PhD Dissertation

International Doctorate School in Information and Communication Technologies

DISI - University of Trento

HOLISTIC SECURITY REQUIREMENTS ENGINEERING FOR SOCIO-TECHNICAL SYSTEMS

Tong Li

Advisor:

Prof. John Mylopoulos

University of Trento

April 2016

Abstract

Security has been a growing concern for large organizations, especially financial and governmental institutions, as security breaches in the systems they depend have repeatedly resulted in losses of billions per year, and this cost is on the rise. A primary reason for these breaches is the "socio-technical" nature of today's systems that consist of an amalgam of social and human actors, processes, technology and infrastructure. We refer to such systems as Socio-Technical Systems (STSs). Finding secure solutions for STSs is a difficult and error-prone task because of their heterogeneity and complexity.

The thesis proposes a holistic security requirements analysis framework which categorizes system security concerns into three layers, including a social layer (social actors and business processes), a software layer (software applications that support the social layer) and an infrastructure layer (physical infrastructure, hardware, and devices). Within each layer, security requirements are elicited, and security mechanisms are designed to satisfy the security requirements. In particular, a cross-layer support link is defined to capture how security mechanisms deployed at one layer influence security requirements of the next layer down, allowing us to systematically and iteratively analyze security for all three layers and eventually produce holistic security solutions for the systems.

To ensure the quality of the analysis of our approach and to promote practical adoption of the three-layer approach, the thesis includes two additional components. Firstly, we propose a holistic attack analysis, which takes an attacker's perspective to explore realistic attacks that can happen to a system and thus contributes to the identification of critical security requirements. This approach consists of an attack strategy identification method which analyzes attacker's alternative malicious intentions, and an attack strategy operationalization method which analyzes realistic attack actions that can be performed by attackers. Secondly, the thesis proposes a systematic approach for selecting and applying security patterns, which describe proven security solutions to known security problems. As such, analysts with little security knowledge can efficiently leverage reusable security knowledge to operationalize security requirements in terms of security mechanisms. This approach also allows us to systematically analyze and enforce the impact of deployed security mechanisms on system functional specifications.

We have developed a prototype tool, which implements the formalized analysis methods of our three-layer framework and enables the semi-automatic application of our proposal. With the help of the tool, we apply our framework to two large-scale case studies so as to *validate the efficacy of our approach.*

Keywords[Security Requirements Engineering, Socio-Technical Systems, Goal Models, Security Patterns, Security Attacks]

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor John Mylopoulos, who has provided terrific on my research and have helped me to uncover my interests in research. In particular, I am much impressed with the way he carries out research, patient, rigorous, open-minded, for all of which I have been endeavoring to train myself. Moreover, I would like to thanks Professor John Mylopoulos for providing me many opportunities to participate various conferences and summer schools, in which I am able to communicate with excellent researchers in my field of research.

I am grateful for Professor Fabio Massacci, not only because of serving on my thesis committee but also because of co-supervising the early stage of my PhD study. He has provided me many insightful comments on my research, which have been rooted in my mind and guide my research all the time. Especially, I appreciate and have been influenced by his empirical and rigorous perspective on security research.

I would like to sincerely thank Professor Lin Liu and Professor Diego Calvanese for being my committee members. They have provided excellent comments and advice on my thesis, based on which I am able to further improve the thesis. In addition they have helped me to uncover additional limitations of my work, shedding light on my future research directions.

Many thanks to Jennifer Horkoff for her tremendous help on my work. I have learned a lot from our discussions and from all your detailed comments on my work. Thanks for showing me how to be an excellent postdoctoral researcher, which is important for a late-stage PhD student.

I would also like to extend my thanks to all my colleagues in Trento. It is a great opportunity for me to work and exchange ideas with all of you. All the talks, discussions, and seminars are invaluable experiences for me. Especially, I so much appreciate the friendships with all of you, which make my life joyful.

Special thanks to my parents for their continuous support throughout my PhD stage, helping me to easily concentrate on my research.

Lastly, I would like to share all my achievements with my wife, Nan Yang, who definitely deserves half of them, if not more than that.

Tong (童)

The work compiled in this thesis has been supported by the ERC advanced grant 267856, titled "Lucretius: Foundations for Software Evolution".

Contents

1 Introduction 1
1.1 Complexity of Socio-Technical Systems . . . 1
1.2 Challenges in Designing Secure STSs . . . 2
1.3 Research Objectives and Research Questions . . . 4
1.4 Research Overview and Contribution . . . 6
1.4.1 A Three-Layer Security Requirements Modeling Framework . . . 6
1.4.2 A Three-Layer Security Requirements Analysis Framework . . . 8
1.4.3 Prototype Tool . . . 11
1.4.4 Case Studies . . . 12
1.5 Structure of the Thesis . . . 13
1.6 Published Work . . . 14

2 State of the Art 17
2.1 Security Requirements Engineering . . . 17
2.1.1 Security Requirements Engineering Processes . . . 18
2.1.2 Security Requirements Engineering Approaches . . . 21
2.2 Holistic Security . . . 28
2.2.1 Security Analysis beyond Software Systems . . . 28
2.2.2 Holistic Security Analysis Approaches . . . 30
2.3 Security Attack Analysis . . . 31
2.3.1 Attacker-oriented Analysis . . . 31
2.3.2 Attack Pattern-based Analysis . . . 32
2.3.3 Multistage Attack Analysis . . . 33
2.4 Security Patterns . . . 34
2.4.1 Security Pattern Collections . . . 35
2.4.2 Security Pattern Selection . . . 36
2.4.3 Impact of applying security patterns . . . 38
2.5 Chapter Summary . . . 39

3 Baselines **41**
3.1 Requirements Problem 41
3.2 Goal Modeling Languages 42
3.2.1 i* 43
3.2.2 Techne 44
3.2.3 A Contextual Goal Modeling Language 45
3.3 Security Patterns 45
3.3.1 Pattern Templates 46
3.4 CAPEC 48
3.4.1 CAPEC Schema 48
3.5 Chapter Summary 50

4 A Three-Layer Security Requirements Analysis Framework **51**
4.1 A Three-Layer Structure for STSs 52
4.2 A Three-Layer Requirements Modeling Language 54
4.2.1 Conceptual Model 54
4.2.2 Formal Definitions 60
4.3 A Holistic Security Requirements Analysis Process 64
4.3.1 Security Goal Refinement 65
4.3.2 Security Goal Simplification 69
4.3.3 Security Goal Operationalization 71
4.3.4 Cross-Layer Security Analysis 73
4.3.5 Holistic Security Solution Generation 75
4.4 Discussions 77
4.5 Related Work 80
4.6 Chapter summary 83

5 A Holistic Security Attack Modeling and Analysis Approach **85**
5.1 Approach Overview 86
5.2 Analyzing Attack Strategies via Anti-Goal Refinement 88
5.2.1 Real Attack Scenario Examination 88
5.2.2 An Anti-Goal Refinement Approach 92
5.2.3 Evaluation 95
5.2.4 Discussion 101
5.3 Operationalizing Anti-Goals with Attack Patterns 102
5.3.1 Model Attack Patterns as Contextual Goal Model 103
5.3.2 Attack Pattern Selection and Application 108
5.4 Validation 112

5.5 Discussion and Related Work . . . 115
5.6 Chapter Summary . . . 119

6 Integrating Security Patterns with Security Requirements Analysis 121
6.1 Security Patterns Complement Security Requirements Operationalization . 122
6.2 Model Security Patterns as Contextual Goal Models . . . 123
6.2.1 A Contextual Goal Modeling Language . . . 123
6.2.2 A Process for Creating Contextual Goal Models from Security Patterns . . . 124
6.2.3 Empirical Observations . . . 128
6.3 Selecting and Applying Security Patterns to Operationalize Security Requirements . . . 128
6.3.1 Generate Security Pattern Candidates . . . 130
6.3.2 Security Pattern Selection . . . 131
6.3.3 Security Pattern Application . . . 132
6.4 Related Work . . . 133
6.5 Chapter Summary . . . 134

7 Analyzing the Impact of Security Mechanisms 137
7.1 Impact of Security Mechanisms . . . 138
7.2 Scenario: The Healthcare Collaborative Network (HCN) . . . 139
7.3 An Enriched Requirements Specification . . . 140
7.4 An Enriched Security Mechanism Specification . . . 141
7.5 A Systematic Process for Analyzing the Impact of Security Mechanisms . . 144
7.6 Evaluation . . . 149
7.7 Related Work . . . 154
7.8 Chapter Summary . . . 156

8 A Prototype Tool 157
8.1 Architecture . . . 157
8.2 Features . . . 159
8.3 Use Cases . . . 163
8.4 Chapter Summary . . . 168

9 Validation 169
9.1 Case Study 1: A Smart Grid Real-Time Pricing Scenario . . . 170
9.1.1 Scenario Description . . . 171
9.1.2 Building Three-Layer Requirements Models . . . 171

9.1.3 Analyze Security Requirements in Three Layers 174
9.1.4 Threats to Validity 181
9.2 Case Study 2: Medical Emergency Response System 183
9.2.1 Scenario Description 183
9.2.2 Modeling the Medical Emergency Response System 184
9.2.3 Security Requirements Analysis Results 188
9.3 Chapter Summary 193

10 Conclusions and Future Work **195**
10.1 Conclusions 195
10.2 Discussion 197
10.2.1 Limitations 198
10.2.2 Ongoing Work and Future Research Directions 201

Appendix A Attack Scenario Studies **205**

Appendix B Attack Pattern Hierarchy **209**

Appendix C Goal Models Built from Attack Patterns **215**

Appendix D Context Inference Rules for Attack Patterns **231**

Appendix E Attack Pattern Validation Results **247**

Appendix F Goal Models Built from Security Patterns **251**

Bibliography **267**

List of Tables

2.1 A comparison of security requirements engineering processes 20
2.2 A comparison of security requirements engineering techniques 27
2.3 A comparison of security pattern selection techniques 37

3.1 An exemplary security pattern in [Fernandez-Buglioni, 2013] 47
3.2 An exemplary CAPEC attack pattern 49

4.1 Formal predicates of the three-layer requirements modeling language 63
4.2 Formal predicates for related constructs 64
4.3 Inference rules for refinement methods 66
4.4 Inference rules of applicability analysis 70
4.5 Inference rules of threat analysis 71
4.6 Inference rules of cross-layer analysis 74

5.1 The EBNF syntax of the structured description language 91
5.2 Summarized refinement patterns 92
5.3 Pattern-related conceptual mappings 103
5.4 Mappings between attack pattern impact and STRIDE threats categories . 105
5.5 Typical targets of attack domains 105
5.6 Inference rules for attack operationalization 110

6.1 The specification of the Intrusion Detection System pattern [Fernandez-Buglioni, 2013] 125

7.1 Security constraint rules 143
7.2 Enforcement measures for the six security constraints 148
7.3 Statistics of applying the conceptual model to 20 security mechanisms . . . 149

8.1 The use case specification for refining security goals 163
8.2 The use case specification for identifying critical security goals 164
8.3 The use case specification for identifying security pattern candidates 164

8.4 The use case specification for checking applicability of security pattern candidates . . . 165
8.5 The use case specification for propagating security requirements across layers 165
8.6 The use case specification for generating holistic security solutions 166
8.7 The use case specification for operationalizing anti-goals 166
8.8 The use case specification for checking applicability of attack pattern candidates . . . 166
8.9 The use case specification for generating alternative (multistage) attacks . 167
8.10 The use case specification for identifying constrained requirements tasks . . 167
8.11 The use case specification for enforcing impact of security constraints . . . 167

9.1 Statistics of the three-layer requirements model (smart grid) 174
9.2 Statistics of the three-layer requirements model (medical emergency) . . . 187
9.3 Two exemplary holistic security solutions 192

List of Figures

1.1 An overview of our holistic security requirements analysis framework . . . 7

3.1 An illustration of the requirements problem from [Zave and Jackson, 1997] 42
3.2 Goal models with contexts 45

4.1 The requirements problem extended for STSs 52
4.2 An overview of the three-layer security requirements analysis framework . . 53
4.3 Meta-model of the three-layer security requirements framework 54
4.4 An excerpt of the three-layer requirements model of the smart grid scenario 56
4.5 Hierarchy of security properties 59
4.6 Overview of assets 59
4.7 An overview of the three-layer security requirements analysis process . . . 65
4.8 Security goals refinements 67
4.9 Exhaustive refinements for the root security goal 68
4.10 Selected security patterns in three layers 72
4.11 An example of the operationalization of critical security goals 73
4.12 Security mechanisms cross-layer analysis 75
4.13 Cross-layer security analysis for security goals 75
4.14 Partial view of a holistic security goal model 76

5.1 An overview of the holistic attack analysis framework 87
5.2 Anti-goal models that are built from the “Easy Money” attack scenario . . 90
5.3 An analysis process of anti-goal refinement 93
5.4 Applying anti-goal refinement analysis to the scenario of credit card theft (excerpt) 97
5.5 Apply the attacker strategy to the credit card stolen scenario (full) 99
5.6 An example attack pattern model 104
5.7 A systematic process for attack strategy operationalization 108
5.8 Operationalize a leaf anti-goal using attack patterns 109
5.9 An excerpt of the final attack model 114

6.1 Concept mappings between contextual goal models and security patterns . 124
6.2 The contextual goal model of the IDS pattern 126
6.3 Security pattern analysis process 129
6.4 A part of the security goal model of the smart grid scenario 130
6.5 Applying IDS pattern to the goal model of the smart grid scenario 132

7.1 A snippet of requirements goal model of HCN 140
7.2 An example of the enriched requirements elements 141
7.3 Modeling security mechanism — virtual private network (VPN) 142
7.4 The process for analyzing impact of security mechanisms 145
7.5 Impact of the application of VPN (part) 146
7.6 Examples of knowledge models 147
7.7 The complete three-layer requirements goal model of the HCN scenario . . 151
7.8 Impact of the application of VPN over HCN scenario 152

8.1 An overall architecture of MUSER 158
8.2 The modeling interface of MUSER 160
8.3 Perform analysis using MUSER control panel 162

9.1 The requirements model in the social layer 172
9.2 A three-layer requirements model of the smart grid realtime pricing scenario 173
9.3 A full resource schema considered in this case study 175
9.4 Refine security goals in the social layer 176
9.5 The entire security goal model across three layers 178
9.6 The business process model of the medical emergency scenario 185
9.7 The three-layer requirements goal model for the medical emergency scenario 186
9.8 The resource model related to the medical emergency system 189
9.9 The entire holistic security goal model for the medical emergency system . 190

E.1 Attack strategy model 248
E.2 Operationalize attack strategies into relevant attack patterns 249
E.3 Final attack model 250

Chapter 1

Introduction

> *Security is a chain; it's only as secure as the weakest link.*
>
> Bruce Schneier

Security has been a growing concern for large organizations, especially financial and governmental institutions, as security breaches in the systems they depend have repeatedly resulted in losses of billions per year, and this cost is on the rise. According to Ponemon Institute, the average total cost of a data breach for the 350 companies participating in their study was 3.79 million dollars in 2015 [Ponemon, 2015]. The breaches are caused by a broad scope of factors, including trusted insiders (inadvertent or malicious), malware, SQL injections, and hijacked devices, etc. A primary reason for these breaches is the "socio-technical" nature of today's systems that consist of social and human actors, processes, technology and infrastructure. We refer to such systems as Socio-Technical Systems (STSs). In this chapter, we discuss the complexity of STSs, as well as the challenges which arise when designing secure STSs. Then, we present a three-layer security requirements analysis framework, which addresses those challenges and eventually generates holistic security solutions that satisfy security requirements of STSs.

1.1 Complexity of Socio-Technical Systems

Socio-Technical Systems (STSs) are an amalgam of people and technology. In contrast to vanilla software systems, STSs extend their system boundary to include additional components, such as social and human actors, business processes and physical infrastructure, to better support the achievement of strategic and tactical objectives.

Consider a smart grid advanced metering infrastructure as a typical STS. In order to dynamically regulate the energy price according to the energy load, we need not only an energy management application, but also an effective business process for generating and distributing energy prices, as well as a well-deployed physical infrastructure. Incorrect design of any of these components will impair the satisfaction of stakeholder requirements.

In addition to a broader scope of concerns, the design of an STS must not ignore interactions among different components, since the design of one system component can influence another. In the smart grid example, the design of the price generation process determines the features of the energy management application, and which physical devices should be deployed according to the applications they host (e.g., if an application needs to communicate with other applications, then the host device should be deployed to connect to a network).

1.2 Challenges in Designing Secure STSs

Expanded system boundary. Thanks to an expanded system boundary, STSs are able to provide advanced functionality catering to more challenging requirements. However, on the security side, an expanded system boundary actually presents a larger attack surface than before, and thus introduces more challenges to the protection of STSs. A common cause for many breaches in STSs is that security solutions are not designed in a holistic fashion. Rather, they are dealt with in a piecemeal fashion, by different analysts, using different analysis techniques, and focusing on different components of STS (e.g., processes, software, hardware). For example, Mitnick and Simon [2005, 2011] focus on analyzing social attacks and corresponding countermeasures; Jürjens [2002] proposes UMLsec for designing secure software applications; Weingart [2000] surveys and discuss known physical attacks and present corresponding security methods. Such approaches cannot guarantee the overall security of an STS, and may lead to security gaps and vulnerabilities for parts of the system. It is worth noting that providing holistic protection for STSs does not imply putting all related security mechanisms into the system design, although such design can probably satisfy the system security requirements. Security is not free: the more security mechanisms a system applies, the more expensive the system is, with system performance and usability likely suffering as well. Regarding information security, good enough always beats perfect, and the big challenge is determining what is good enough [Sandhu, 2003]. Consequently, the challenges are not only producing security specifications for each component of STSs, but also orchestrating them in an appropriate way in order to produce a cost-effective security solution.

Dealing with security issues during the requirements phase has been recognized as an efficient way of reducing security cost, as the earlier security flaws are detected, the less money is paid for security remedies. In particular, Hoo et al. [2001] have reported that introducing security analysis in the early stage of the system development cycle can reduce costs related to software development and maintenance from 12-21%. Security Requirements Engineering, as a research branch, has been founded on the need to analyze security requirements early on, along with other requirements, instead of as an afterthought. Over the last two decades, many approaches have been proposed to model, elicit, and evaluate security requirements. Some of these approaches focus on the social and organizational aspect of a system [Liu et al., 2003; Giorgini et al., 2005a; Mouratidis and Giorgini, 2007a; Paja et al., 2013]; some others investigate security requirements regarding the business processes of a system [Herrmann and Herrmann, 2006; Rodríguez et al., 2011; Salnitri et al., 2014a]; most of these approaches analyze software security requirements [Sindre and Opdahl, 2005; Van Lamsweerde et al., 2007; Haley et al., 2008; Hatebur et al., 2006] However, when designing secure STSs, we need a holistic approach that spans all layers of an STS, rather than just one.

Multistage attacks. Multistage attacks have been recognized as a growing threat for any complex systems, including STSs. Multistage attacks are comprised of less dangerous attack actions (or even harmless ones) and are difficult to be detected, imposing their challenges to system security [Ourston et al., 2003]. Due to their complex nature, STSs have become the ideal target of multistage attacks. As a result, there has been a substantial increase of multistage attacks on STSs [Mitnick and Simon, 2011, Ch. 11]. For example, to break into an intranet of a company, an attacker might first harvest employee email addresses via social information gathering attacks (dumpster diving, pretexting, etc.); then the attacker sends emails to employees to convince them to install malware in their computers; with the malware, the attacker is able to penetrate into the intranet of the company. To deal with such multistage attacks in STSs, we need to not only capture all attack techniques to different components of STSs, but also analyze all potential attack strategies that present specific ways of composing atomic attack actions.

Much work has been done to analyze multistage attacks for network security by using attack graphs [Phillips and Swiler, 1998; Sheyner et al., 2002]. In particular, such approaches leverage model checking techniques to automatically examine all possible penetration paths that attackers may have. A key factor to the success of these approaches is that computers in the network have homogeneous settings. As such, the states of the computers (nodes in the attack graph) and the atomic attacks on these computers (transitions in the attack graph) can be enumerated. Similarly, a recent proposal applies the attack graph approach to analyze multistage attacks of social engineering, where the states of people are modeled as nodes and social engineering attacks are captured as transitions between nodes [Beckers et al., 2015]. However, as the attack graph approach only applies to systems that have simple and homogeneous components, it is inappropriate for security analysis of complex STSs that have heterogeneous components.

Availability of security knowledge. As the components of STSs are by their nature heterogeneous, each one of them raises different security concerns. This implies that designers need to have considerable security knowledge. For example, in order to design a secure business process, analysts should ensure safe task assignment, no conflicts of interest, etc.; while for a secure software application, analysts should consider software technical phenomena, e.g., encryption, suitable access controls, regular software patches; when designing secure infrastructure, analysts should take into account physical issues (e.g, locks, uninterrupted energy suppliers) . Ensuring that such broad security knowledge is available during the design process constitutes yet another challenge.

Failing to solve this challenge will render the holistic security analysis too time consuming and laborious, preventing the practical adoption of our framework. A recent study acknowledges such a challenge, especially because security knowledge is hard to acquire for software designers in reality [Souag et al., 2015]. As a result, the authors of the study advocate for reuse of provable security knowledge, which can make security requirements analysis much easier, more in-depth, and faster. In addition, the authors also emphasize that the knowledge sources have to be of high quality and must be applied in a correct and effective manner. Otherwise, the security analysis results may not be reliable, and may even introduce new security problems.

1.3 Research Objectives and Research Questions

Based on the context and challenges we have discussed in the previous sections, we now present the overall research objective of this thesis.

Research Objective: *develop a comprehensive framework that assists security analysts in analyzing security requirements and generating security solutions for socio-technical systems in a holistic manner.*

We decompose this research objective into the following specific research questions, all of which are addressed by this thesis.

RQ1: How can we *holistically* analyze the security requirements of STSs?

The holistic analysis needs to appropriately deal with the challenges concerning complexity and heterogeneity. Due to the increased system boundary of STSs, analyzing security requirements of STSs becomes much more complex. For one thing, analysts have to take into account all components of STSs, and their heterogeneity, in order to provide comprehensive protection. For another, analysts also need to appropriately orchestrate security requirements of different components to obtain the most cost-effective solution. Moreover, the heterogeneity of STSs entails that different system components involve different security concerns. Thus, a general conceptual framework is required to unify security requirements analysis for all the components.

RQ2: How can we identify *realistic* threats to an STS?

Threat identification has been recognized as an important step for engineering security requirements by many researchers [Mead and Stehney, 2005; Mellado et al., 2007], because it helps analysts to determine the criticality of security requirements, so that they can design suitable security solutions.

As any single vulnerability or exposure of any component of an STS can lead to serious security breaches, identifying all potential threats to an STS becomes more important and at the same time challenging. This challenge is exacerbated by the increasing prevalence of multistage attacks, which adopt more sophisticated strategies and consist of multiple attack actions, exploiting vulnerabilities of different system components. Moreover, a lack of knowledge about impending attacks introduces another challenge to threat analysis, which can result in unrealistic threats and further introduce false positives or false negatives into security analysis [Barnum and Sethi, 2007].

RQ3: How can we *efficiently* operationalize security requirements?

Once obtaining critical security requirements and corresponding threats, we further investigate how to operationalize the security requirements in terms of security mechanisms which can tackle the threats. Since such analysis requires intensive security knowledge, we base our analysis on security patterns, which document proven security solutions to known security problems. Specifically, we leverage security patterns from different repositories which contain more than 100 security patterns, accommodating security analysis in different layers [Asnar et al., 2011a; Yskout et al., 2006; Fernandez-Buglioni, 2013]. However, such security patterns are normally specified in text which can only be manually selected and applied by analysts. Given the large number of patterns, a systematic methodology is required to efficiently leverage security patterns to operationalize security requirements. In addition, after a security requirement has been operationalized by one or several security mechanisms, a subsequent challenge is to efficiently identify and enforce the impact of these mechanisms imposed on system functional requirements, otherwise the resulting requirements specification is incomplete.

RQ4: Can we apply the developed approach to *effectively* analyze security requirements of STSs in realistic settings?

As a design artifact, our proposal needs to be appropriately validated, which is an imperative part of achieving methodological soundness [Wieringa and Heerkens, 2006]. As our framework is intended to provide automated reasoning support for the holistic security requirements analysis, we need to first implement corresponding analysis methods in a prototype tool. With the support of the tool, we need to collect empirical evidence to validate the efficacy of our framework when it is applied to realistic large-scale STSs. In particular, we focus on evaluating whether the entire approach can effectively identify holistic security requirements of STSs and efficiently deal with the complexity of large-scale STSs. Furthermore, we should also evaluate the utility of the supporting tool.

1.4 Research Overview and Contribution

This thesis proposes a three-layer security requirements analysis framework, which can holistically analyze security requirements of STSs and eventually generate holistic security solutions that satisfy the security requirements. The contributions of the thesis consist of four parts, shown schematically in Fig. 1.1. Specifically, we propose a three-layer security requirements modeling framework to holistically capture phenomena of STSs and a companion analysis framework which analyzes security requirements across all three layers. We have developed a prototype tool to implement the proposed analysis methods so as to semi-automate the overall analysis process. Finally, we perform two comprehensive cases studies to validate our proposal. In the remaining part of this section, we describe the above contributions in detail. For each contribution, we list relevant publications and research questions addressed.

1.4.1 A Three-Layer Security Requirements Modeling Framework

In order to deal with the inherit complexity of STSs, we structure an STS into three conceptual layers, each of which accounts for particular artifacts that need to be designed in the STS. At the most abstract level, we consider a social layer that conceptualized in terms of social actors, social dependencies, and business process activities. At the next layer, we consider software applications that support the social layer, conceptualized in terms of architectural components. Finally, we consider an infrastructure layer which focuses on physical infrastructure that supports deployment of software applications and business processes. We argue that each layer involves specific security solutions that deal with particular security requirements of that layer. For example, *separation of duty* ensures the service integrity in the social layer, while *secure pipe* is deployed to tackle confidentiality issues in the application layer. Thus, a holistic security requirements analysis of

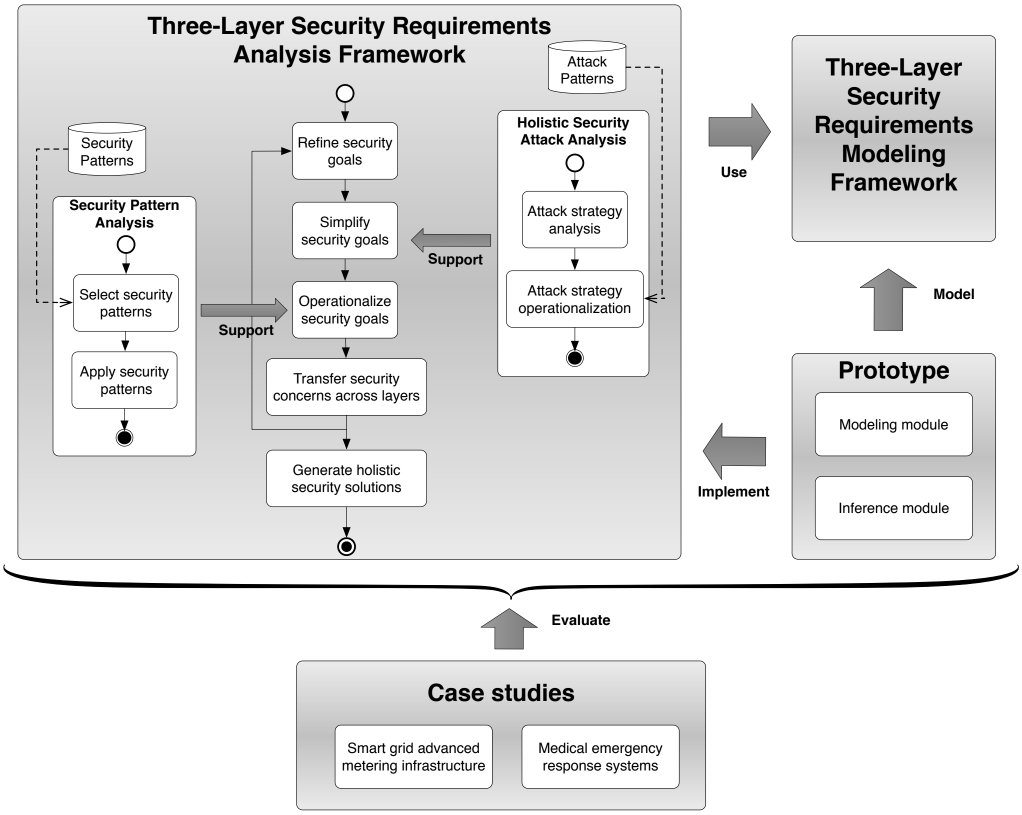

Three-Layer Security Requirements Analysis Framework
Security Patterns
Security Pattern Analysis
Select security patterns
Apply security patterns
Support
Refine security goals
Simplify security goals
Operationalize security goals
Transfer security concerns across layers
Generate holistic security solutions
Support
Attack Patterns
Holistic Security Attack Analysis
Attack strategy analysis
Attack strategy operationalization
Use
Three-Layer Security Requirements Modeling Framework
Model
Prototype
Modeling module
Inference module
Implement
Evaluate
Case studies
Smart grid advanced metering infrastructure
Medical emergency response systems

Figure 1.1: An overview of our holistic security requirements analysis framework

STSs requires us to identify security requirements and generate security solutions in all the three layers.

To deal with the requirements of STSs, we base our framework on the requirements problem defined by Zave and Jackson [1997], i.e., finding a collection of specifications, which can satisfy all requirements under certain domain assumptions. In particular, we extend the Zava and Jackson's requirements ontology and re-formulate the requirements problem to account for requirements of STSs based on the three-layer structure. Thus, each layer has its own requirements, which are satisfied by layer-specific specifications under corresponding domain assumptions. To model and analyze the extended requirements problem of STSs, we propose a three-layer security requirements modeling language based on existing goal modeling languages i* [Yu, 1997] and Techne [Jureta et al., 2008]. The proposed language can model not only the layer-specific security requirements and security solutions, but also the dependencies across layers. As such, it provides the foundation for holistically analyzing security requirements throughout three layers.

Contributions to literature. Existing requirements modeling languages, such as [Dardenne et al., 1993; Yu, 1997; Jureta et al., 2008], do not explicitly account for requirements of different artifacts of STSs (business processes, physical infrastructures, etc.). We extend Zave and Jackson's core RE ontology [Zave and Jackson, 1997] and re-formulate the requirements problem, providing the theoretical foundation for dealing with requirements of STSs. The proposed modeling language can be used to model security requirements and security mechanisms for various artifacts involved in STSs, while capturing the interrelationships among them. As such, the modeling language enables us to deal with requirements of STSs using a divide and conquer approach [Knuth, 1998, p.159], which can reduce the complexity of security requirements analysis of STSs.

Publications: Li and Horkoff [2014].

Addressed research questions: *RQ1.*

1.4.2 A Three-Layer Security Requirements Analysis Framework

Based on the three-layer requirements modeling framework, we propose a comprehensive set of analysis methods to holistically analyze security requirements of STSs in three layers. In the top-left part in Fig. 1.1, an overall process of the three-layer analysis framework is presented, which iteratively refines, simplifies, operationalizes security goals in each layer, and transfers security concerns from one layer to the next layer down. In subsequent work, two analysis steps have been further elaborated to tackle particular challenges that we encountered during the evaluation of the holistic analysis framework. Firstly, a holistic attack analysis approach is proposed to holistically identify all potential attacks that are related to security goals. Secondly, we propose a security pattern-based analysis approach to support security goal operationalization. This framework takes security goals as input, then effectively selects and applies appropriate security patterns to operationalize the security goals. In the rest of this section, we introduce the overall analysis process and the two supporting analysis approaches, respectively.

Holistic security requirements analysis process

Based on the three-layer structure, we propose a systematic process and a set of analysis methods to guide security analysis both within one and across layers. Within each layer, we first iteratively refine and concretize security goals so as to identify critical security goals, which need to be satisfied. After that the identified critical security goals are operationalized in terms of security mechanisms. After the operationalization analysis, we propagate security concerns from one layer to the next layer down based on the connections between layers. After iteratively performing such security analysis for all layers, finally, we can generate a collection of alternative security solutions, which satisfy critical security requirements across all the three layers. A set of formal predicates and inference rules have been defined to (semi-)automate the above analysis. We have implemented such rules in Disjunctive Datalog [Eiter et al., 1997].

Contributions to literature. As different artifacts involve different security issues, existing approaches focus on analyzing security requirements of specific artifacts, e.g., Lamsweerde [2004] analyzes software security requirements, while Rodríguez et al. [2007b] investigate security requirements for business processes. Our approach, built on top of the three-layer structure, holistically analyzes the security requirements of various artifacts of STSs in a divide-and-conquer manner. In this way, our approach is able to appropriately deal with the inherent complexity of STSs.

Publications: Li and Horkoff [2014].

Addressed research questions: *RQ1.*

Holistic attack analysis

During the three-layer security requirements analysis, in order to identify critical security goals, we need to know possible attacks on the target systems. However, compared to typical software systems, STSs suffer from a broader scope of attacks due to their complexity and heterogeneity. Especially, the increasing types of attacks can lead to exponentially more multistage attacks which compose atomic attack actions from different parts of STSs. Precisely detecting all the possible attacks on an STS is essential for determining the criticality of security goals during the holistic security requirements analysis.

We propose a holistic attack analysis approach to detect (multistage) attacks on STSs from an attacker's viewpoint. In particular, this approach takes the three-layer security requirements goal model as input and holistically detect attacks related to the security requirements. As indicated in the top-left corner in Fig. 1.1, the approach consists of two parts: firstly, we identify an attacker's strategies by systematically elaborating an attacker's malicious intentions. To this end, we systematically examine three real comprehensive attack scenarios, through which we investigate how attackers elaborate their malicious intentions to produce attack strategies. Grounded in such real evidence, we propose an attacker strategy analysis framework which supports systematic exploration of attack strategies. Secondly, we analyze how attackers implement identified attack strategies in terms of realistic attack behaviors. For this purpose, we depend on the CAPEC attack patterns [Barnum and Sethi, 2007] for realistic attack knowledge, which helps us to operationalize attackers' malicious intentions in terms of realistic attack behaviors. Specifically, we propose to model attack patterns in terms of contextual goal models, allowing us to semi-automatically identify applicable attack patterns with tool support. We have pragmatically processed and modeled 102 attack patterns in the CAPEC repository, and have seamlessly integrated such attack knowledge into our holistic attack analysis.

Once we finish the attack operationalization analysis, the resulting identified attacks will be used in the three-layer security requirements analysis for identifying critical security goals.

Contributions to literature. Many approaches have advocated for analyzing security requirements from an attacker's viewpoint [Elahi et al., 2010; Lin et al., 2003b]. However, due to the knowledge gap between attackers and defenders [Mansourov and Campara, 2010], it is very difficult for analysts to realistically analyze attacker's intentions and behaviors. Our approach deals with this challenge by practically examining realistic attack scenarios and leveraging reusable attack patterns. In the meantime, our approach contributes to the detection of multistage attacks, as we are able to understand how an attacker composes atomic attack behaviors based on the attacker's intentional model. Lastly, our proposal not only serves our attack analysis, but also contributes to the practical adoption of attack patterns. We have pragmatically followed our method to model 102 real attack patterns, which can thus be semi-automatically selected and applied by analysts in our attack analysis.

Publications: Li et al. [2015a], Li et al. [2015d], [Li et al., 2015c], [Li et al., 2016] (accepted).

Addressed research questions: *RQ2.*

Security pattern analysis

Operationalizing security requirements in terms of security mechanisms is a laborious and knowledge-intensive process for large-scale systems, e.g., STSs. Such operationalization analysis plays an imperative role in the holistic security requirements analysis, as the results of this analysis (i.e., security mechanisms) at one layer can affect security requirements in the next layer down. Therefore, if the operationalization results are incomplete, the subsequent security analysis in lower layers will be affected accordingly.

In order to enhance the efficacy of the operationalization analysis, we practically leverage reusable security patterns to bridges the knowledge gap between security requirements (i.e., security problems) and security mechanisms (i.e., security solutions). In particular, we model security patterns in terms of the contextual goal model [Ali et al., 2010] in order to seamlessly integrate security patterns analysis with our goal-oriented security requirements analysis. To this end, we have first defined a conceptual mapping between the primary concepts of security patterns and contextual goal models, as well as a process for building contextual goal models from security patterns. Moreover, we propose a detailed analysis process, which helps analysts to systematically and semi-automatically select and apply security patterns to operationalize security requirements. To promote the practical adoption of this approach, we have pragmatically modeled 20 security patterns in the repository [Fernandez-Buglioni, 2013].

During the application of security patterns, i.e., operationalizing security goals into security mechanisms, we noticed that the deployed security mechanisms can affect not only security requirements in the next layer down, but also the system functional requirements. Ignoring such impact will result in incomplete requirements specifications. As such, we have proposed a systematic method to capture and enforce the impact of deployed security mechanisms.

Contributions to literature. Compared to existing approaches [Araujo and Weiss, 2002; Mouratidis et al., 2006; Asnar et al., 2011a], our approach first contributes to the practical integration of security patterns and goal-based security requirements analysis. Specifically, we propose a systematic guideline for modeling security patterns in terms of the contextual goal model, and have pragmatically modeled 20 security patterns from existing pattern repository. with the tool support, our approach can be semi-automatically applied to select and apply security patterns. Lastly, our approach can capture the impact that security mechanisms imposed on functional requirements, and can semi-automatically enforce such impact.

Publications: [Li and Mylopoulos, 2014], [Li et al., 2014a], [Li et al., 2015b].

Addressed research questions: *RQ3.*

1.4.3 Prototype Tool

We have developed a prototype tool MUSER in order to semi-automate our proposal in this thesis, facilitating its practical adoption. The tool was initially designed to support the three-layer security requirements modeling and analysis, and has been incrementally enhanced to support the elaborated holistic attack analysis and security pattern analysis. In particular, this tool consists of two modules, as indicated in Fig. 1.1. The modeling module supports constructing different analysis models used by our framework, while the reasoning module (semi-)automates analysis methods that are defined in the framework.

As the complexity of STSs imposes challenges on both the modeling module and the reasoning module, we have purposely optimized the prototype to tackle such challenges. In particular, we developed the prototype tool on top of a powerful diagramming application OmniGraffle[1], allowing us to easily build large-scale models and avoid pragmatic problems of graphical modeling, such as mentioned in [Massacci and Paci, 2012]. Moreover, we implemented the inference module of the prototype tool based on the DLV inference engine, which can efficiently reason over large amounts of data[2].

---

[1] `http://www.omnigroup.com/omnigraffle`

[2] `http://www.dlvsystem.com`

Contributions to literature. The prototype tool plays an important role in dealing with the complexity of security requirements analysis of STSs. On one hand, the tool allows us to realistically apply our proposal to analyze security requirements of STSs, serving as the precondition for conducting large-scale case studies. On the other hand, the tool contributes to the practical adoption of our approach.

Publications: [Li et al., 2014b].

Addressed research questions: *RQ4*.

1.4.4 Case Studies

To validate our approach, we need to evaluate the efficacy of the approach when applying it to realistic systems. As such, we need to first choose appropriate methods to collect and analyze empirical data. Easterbrook et al. [2008] summarize five empirical methods for software engineering research, including controlled experiments, case studies, survey research, ethnographies, and action research. Due to the complexity of STSs, a full application of our approach requires analysts to have comprehensive knowledge about phenomena in all three layers of STSs and perform a series of analysis steps using such knowledge (as indicated in Fig. 1.1), which can take several person-months. Considering the amount of time required to apply our approach, we choose to use **case studies** to validate our approach, as "*case study research is most appropriate for cases where effects are expected to be wide ranging, or take a long to appear*" [Easterbrook et al., 2008].

When performing case studies, it is essential to select appropriate cases that are most relevant to our research problem, because the properties of our approach that are validated by such case studies are likely to hold for many other cases. We have purposely selected two representative large-scale STSs to perform case studies, which intensively involve phenomena across all the three layers of STSs and have high demand for security.

Firstly, I applied our approach (by myself) to smart grid advanced metering infrastructure based on realistic specifications [NIST, 2012; Cuellar and Suppan, 2013], and holistically analyzed security requirements for protecting the metering data. In this study, we focus on evaluating the efficiency and expressiveness of the proposed modeling language, as well as the efficiency and effectiveness of the proposed analysis framework.

Secondly, we applied our approach to a large-scale medical emergency response system, which is enriched from an European project case study [Serenity-Consortium, March 2007], and address the system's holistic security concerns. This study is grounded on a Master thesis [Robin, 2015]. The Master student was first taught the three-layer security requirements modeling language, and then used the prototype to perform the holistic security requirements analysis under the supervision of Prof. Mylopoulos and myself. Different from the first case study which focuses on evaluating the efficacy of our approach, this study is intended to preliminarily assess whether our approach has the potential to be adopted in reality by people who were not involved in the development of the approach.

Contributions to literature. The two case studies validate the efficacy of our approach when applied to realistic large-scale systems. In addition, they help us to better understand the advantages and limitations of our approach.

Addressed research questions: *RQ4*.

1.5 Structure of the Thesis

- Chapter 2 presents the state of art of this thesis. We first review approaches for engineering security requirements and designing holistic security. Then, we examine research in the area of security attack analysis, which plays an important role in identifying security requirements. Finally, we survey the subject of security patterns which bridge the knowledge gap between security problems and security solutions.

- Chapter 3 describes baseline approaches, which have been used or extended in this thesis, including the requirements problem, the goal modeling languages, security patterns and attack patterns.

- Chapter 4 first introduces the three-layer security requirements modeling framework which can model phenomena in different layers of STSs. Based on the three-layer model, we then describe the overall analysis framework that is used to holistically analyze security requirements of STSs. Finally, we compare the three-layer security requirements modeling and analysis framework with the state of the art.

- Chapter 5 presents the holistic attack analysis approach, which supports the security goal simplification analysis during the three-layer security requirements analysis. We first introduce an attacker strategy analysis method which explores potential attack strategies, and then describe an attack pattern-based approach that operationalizes those attack strategies. Lastly, we discuss the related work of this attack analysis approach.

- Chapter 6 presents the security pattern-based operationalization analysis, which integrates security patterns into our three-layer framework and assists analysts with little security knowledge in operationalizing security goals. We first present the conceptual mapping between security patterns and contextual goal models. Then we describe the systematic process for selecting and applying security patterns. Finally, we discuss related security pattern analysis approaches.

- Chapter 7 describes the analysis framework for analyzing and enforcing the impact of security mechanisms. We first describe the proposed enriched requirements specification and security mechanism specification, respectively. Then, we introduce the systematic analysis process for detecting and analyzing the impact. Finally, we compare our proposal with approaches related to the security impact analysis.
- Chapter 8 describes the prototype tool MUSER we have developed to support modeling and analyzing security requirements of STSs in three layers. In particular, We not only present the architecture and features of the prototype tool, but also describe detailed use case specifications for each feature in order to guide potential users.
- Chapter 9 presents the two case studies we have performed for validation of our approach. For each case study, we first introduce the scenario, and then describe the three-layer model we have built based on the scenario, and finally present and evaluate the analysis results.
- Chapter 10 concludes the thesis, discusses limitations of the holistic security requirements analysis framework, and clarifies subsequent work that needs to be done to improve the approach. As inspired by the proposal in this thesis, we have identified a number of future research directions, which are introduced at the end of this thesis.

1.6 Published Work

All the publications related to this thesis are listed here, which are divided into three categories: journals, conferences, workshops and demos. All the publications are refereed.

Journals

- Horkoff, Jennifer; Li, Tong; Li, Feng-Lin; Salnitri, Mattia; Cardoso, Evellin; Giorgini, Paolo, and Mylopoulos, John. Using goal models downstream: A systematic roadmap and literature review. *International Journal of Information System Modeling and Design (IJISMD)*, 6(2):1–42, 2015

Conferences

- Li, Tong; Horkoff, Jennifer; Paja, Elda; Beckers, Kristian, and Mylopoulos, John. Security attack analysis using attack patterns. In *The IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*. IEEE, 2016 (accepted)

- Li, Tong; Horkoff, Jennifer; Paja, Elda; Beckers, Kristian, and Mylopoulos, John. Analyzing attack strategies through anti-goal refinement. In *The Practice of Enterprise Modeling (PoEM 2015)*, pages 75–90. Springer International Publishing, 2015c
- Li, Tong; Horkoff, Jennifer, and Mylopoulos, John. Analyzing and enforcing security mechanisms on requirements specification. In *Requirements Engineering: Foundation for Software Quality (REFSQ 2015)*. Springer International Publishing, 2015b
- Li, Tong and Horkoff, Jennifer. Dealing with security requirements for socio-technical systems: A holistic approach. In *Advanced Information Systems Engineering (CAiSE 2014)*, pages 185–200. Springer International Publishing, 2014
- Li, Tong; Horkoff, Jennifer, and Mylopoulos, John. Integrating security patterns with security requirements analysis using contextual goal models. In *The Practice of Enterprise Modeling (PoEM 2014)*, pages 208–223. Springer Berlin Heidelberg, 2014a
- Horkoff, Jennifer; Li, Tong; Li, Feng-Lin; Salnitri, Mattia; Cardoso, Evellin; Giorgini, Paolo; Mylopoulos, John, and Pimentel, João. Taking goal models downstream: A systematic roadmap. In *Research Challenges in Information Science (RCIS), 2014 IEEE Eighth International Conference on*, pages 1–12. IEEE, 2014b
- Horkoff, Jennifer; Aydemir, Fatma Başak; Li, Feng-Lin; Li, Tong, and Mylopoulos, John. Evaluating modeling languages: An example from the requirements domain. In *Conceptual Modeling (ER 2014)*, pages 260–274. Springer International Publishing, 2014a

Workshops and Demos

- Li, Tong; Paja, Elda; Mylopoulos, John; Horkoff, Jennifer, and Beckers, Kristian. Holistic security requirements analysis: An attacker's perspective. In *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*, pages 282–283. IEEE, 2015d
- Li, Tong; Horkoff, Jennifer; Beckers, Kristian; Paja, Elda, and Mylopoulos, John. A holistic approach to security attack modeling and analysis. In *Proceedings of the Eighth International i* Workshop*, pages 49–54, 2015a
- Li, Tong and Mylopoulos, John. Modeling and applying security patterns using contextual goal models. In *The 7th International i* Workshop (iStar14)*, pages 208–223, 2014

- Li, Tong; Horkoff, Jennifer, and Mylopoulos, John. A prototype tool for modeling and analyzing security requirements from a holistic viewpoint. In *The CAiSE'14 Forum at the 26th International Conference on Advanced Information Systems Engineering*, pages 185–192, 2014b

Chapter 2

State of the Art

If I have seen further it is by standing on the shoulders of giants.

Isaac Newton

In this chapter, we first review the state of the art in the area of security requirements engineering which deals with security requirements elicitation and analysis in the early phase of software development (Section 2.1). Moreover, we survey several related research areas that contribute to the security requirements analysis of STSs, corresponding to the research challenges we have introduced in Section 1.2. In particular, we review the approaches that deal with security beyond software systems and holistically analyze system security (Section 2.2); the approaches that identify and analyze attacks (Section 2.3); and the approaches that select and apply security patterns (Section 2.4). Note that in this chapter we review and discuss the advantages and disadvantages of existing approaches according to the research questions and challenges we have described in the last chapter. Detailed comparison between our proposal and the existing approaches will be presented in the following chapters after introducing our proposal.

2.1 Security Requirements Engineering

Security requirements engineering is motivated by the fact that analyzing security in the early stage of the system development cycle can significantly save costs for later system development and maintenance [Hoo et al., 2001], which has been investigated for more than two decades. Many approaches have been proposed to produce high-quality security requirements. In this section, we first review and compare approaches that define processes for systematically engineering security requirements, based on which we summarize a *generic process of security requirements engineering* that cover all important analysis steps of security requirements engineering. After that we review existing typical security requirements engineering approaches on the basis of several detailed survey papers [Nhlabatsi et al., 2010; Fabian et al., 2010; Elahi et al., 2011; Souag et al., 2015]. All the reviewed approaches are then systematically compared based on the topics of this thesis presented in Section 1.1, e.g., focused system layers.

2.1.1 Security Requirements Engineering Processes

Although analyzing security requirements upfront has been widely accepted, Mead and Stehney [2005] have argued that without a systematic process of engineering security requirements the obtained security requirements can be ambiguous and incomplete. Specifically, the ad hoc way of analyzing security requirements is tended to simply describe general security mechanisms instead of analyzing what stakeholders need for their systems' security [Firesmith, 2003a; Mead, 2006b]. In order to help analysts to engineer high quality security requirements, Mead [2006b] has developed the SQUARE (Security QUAlity Requirements Engineering) methodology, which provides a nine-step security requirements elicitation and analysis process (shown in Table 2.1). This process involves stakeholders, requirements engineers, risk experts, and inspection teams, and eventually delivers a collection of categorized and prioritized security requirements. Specifically, the author has developed a prototype tool to support such analysis process. In subsequent research, [Chen et al., 2004] iteratively evaluated and improved the SQUARE process through several case studies, which involved real-world clients that were developing large-scale IT projects. Moreover, with the aim of improving the effectiveness of SQUARE, Mead et al. [2008] propose a method to fit the SQUARE methodology into the Rational Unified Process (RUP) [Kruchten, 2004]. Note that the SQUARE process presents general steps that need to be performed by security analysts, while does not limit itself to specific techniques. Thus, analysts should select appropriate techniques to implement each step of the process. In particular, Mead [2006a] compared difference security requirements elicitation techniques, shedding light on how such techniques can be used in the SQUARE process.

Mellado et al. [2007] propose a Common Criteria (CC) centered and reuse-based Security Requirements Engineering Process (SREP), which is partially based on SQUARE. CC is an international security standard (ISO/IEC 15408) for evaluating the security properties of information systems, and it has been incorporated in SREP to facilitate security requirements analysis in two ways: first, it contains a collection of well-documented security functional requirements, assisting analysts in eliciting and specifying security requirements. Secondly, CC includes assurance requirements which can be used to evaluate the security of the system and thus help to inspect the elicited security requirements. The reusability of SREP is derived from a Security Resources Repository (SRR). The repository stores reusable artifacts (e.g., threat specifications and security requirements specifications) that are specified using different techniques, such as Misuse cases [Sindre and Opdahl, 2005] and UMLsec [Jürjens, 2002]. The entire analysis process of SREP consists of nine steps (as shown in Table 2.1), which account for the two new features of SREP as introduced above. Note that SREP is an iterative and incremental process, which is built on top of the Unified Process (UP) [Jacobson et al., 1999]. Mellado et al. [2006] have performed a case study to demonstrate how the security requirements for a security critical system can be systematically obtained by applying SREP.

Another related piece of work is carried out by Wang et al. [2009], who propose a more fine-grained process for dealing with security requirements in the early stage of system development. The process consists of nine steps (Table 2.1), which has an emphasis on the threat and risk analysis but does not consider the prioritization and inspection of security requirements after they have been elicited.

A recent study proposes a dedicated security requirements engineering process for web applications (MOSRE-WebApp) [Salini and Kanmani, 2012]. MOSRE-WebApp involves 16 specific analysis steps (Table 2.1), which are presented using a spiral process model in order to cover all phases of RE. The authors have performed a comprehensive case study for an e-voting system, and the evaluation results of this study shows that the effectiveness and performance of this process are comparatively better than existing approaches [Salini and Kanmani, 2015].

Comparison of processes

Given the four security requirements engineering processes we have reviewed above, we summarize a generic process which sheds light on detailed and important analysis steps that should be performed in order to systematically engineer security requirements. Based on such insights, we further discuss similarities and differences among the four reviewed processes. Note that this generic process we presented here serves as a theoretical foundation for our holistic security requirements analysis framework.

Basically, we identify 11 generic steps of security requirement engineering, which are derived from elaborating two steps of SQUARE. Firstly, we add a step "*Identify assets*" before the second step of SQUARE "*Identify security goals*". This new step is a non-trivial step, which is an imperative precondition for identifying security goals and has been recognized by the other three proposals. Secondly, we elaborate the third steps of SQUARE "*Develop artifacts to support security requirements definition*" into two sub-steps "*Domain analysis*" and "*Identify threats/vulnerabilities*". The former sub-step focuses on developing functional models that character the system, while the latter one identifies threats and vulnerabilities in the system. Apart from these two modifications, the rest of

Table 2.1: A comparison of security requirements engineering processes

Generic Steps	**SQUARE**	**SREP**	**Wang et al. [2009]**	**MORSE-WebApp**
(1) Agree on definition	(1) Agree on definitions	(1) Agree on definitions	(1) Consistency of Definitions	
(2) Identify assets		(2) Identify vulnerable and critical assets	(3) Identify critical assets and process	(3) Identify the assets
(3) Identify security goals	(2) Identify security goals	(3) Identify security objectives and dependencies	(6) Security goals and dependencies	(8) Identify security objectives/goals
(4) Domain analysis	(3) Develop artifacts to support security requirements definition		(2) Analyzing characters of system	(1) Identify the objective of the software system (2) Identify the stakeholder (5) Obtain high level architecture diagram (6) Elicit non-security goals and requirements (7) Generate use cases diagram
(5) Identify threats/vulnerabilities	(3) Develop artifacts to support security requirements definition	(4) Identify threats and develop artifacts	(4) Identify system vulnerability (5) Identify threats (7) Generating threat model	(9) Identify threats and vulnerabilities
(6) Perform risk analysis	(4) Perform risk assessment	(5) Risk assessment	(8) Risk assessment	(10) Perform risk analysis (11) Categorize and prioritize threats and vulnerabilities for mitigation (12) Generate misuse case diagram
(7) Select elicitation techniques	(5) Select elicitation techniques			(4) Select elicitation techniques
(8) Elicit security requirements	(6) Elicit security requirements	(6) Elicit security requirements	(9) Elicit security requirements	(13) Identify security requirements (14) Generate use cases diagram considering security requirements
(9) Categorize requirements	(7) Categorize requirements as to level and whether they are requirements or other kinds of constraints	(7) Categorize and prioritize requirements		
(10) Prioritize requirements	(8) Prioritize requirements	(7) Categorize and prioritize requirements		
(11) Requirements inspection	(9) Requirements inspection	(8) Requirements inspection		
		(9) Repository improvement		(15) Generate structural analysis models (16) Develop UML diagrams

the generic analysis steps are the same with corresponding steps in SQUARE.

In Table 2.1, we presented the 11 generic steps of security requirement engineering in the first column. From the second column to the fifth column, we list detailed analysis steps of the four security requirements engineering processes, which have been categorized into corresponding generic steps. As SREP is proposed partially based on SQUARE, there are only a few differences between them. Specifically, SREP emphasizes the step of asset identification, while omits the step of selecting elicitation techniques. Moreover, SREP has a particular step "*Repository improvement*" as it proposes to reuse analytical artifacts. As shown in Table 2.1, the process which is proposed by Wang et al. [2009] covers the first 8 generic analysis steps except for step 7. This process proposal has an emphasis on the threat and vulnerability analysis, but does not further process security requirements after they are elicited. For MOSRE-WebApp, it has a strong focus on both domain analysis and risk analysis. However, similar to [Wang et al., 2009], MOSRE-WebApp does not categorize nor prioritize elicited security requirements. To be noted that the last two steps of MOSRE-WebApp have stepped into system security design and thus are beyond our discussion.

2.1.2 Security Requirements Engineering Approaches

Goal-oriented approaches

Chung [1993] proposes to treat security requirements as a class of Non-Functional Requirements (NFRs), which are used for selecting among system design decisions and justifying the overall design. The author also defines a process-oriented approach to analyze security requirements, which is adapted from the work done by Mylopoulos et al. [1992]. This approach emphasizes the importance of reusing security design knowledge. In particular, the approach incorporates reusable taxonomies of security properties and security goal satisficing methods, so as to support security goal refinement and operationalization. Oladimeji et al. [2006a] extend NFR framework with the notations of negative softgoals for representing threats and inverse contributions in order to model and analyze threats. In addition, another extension of NFR proposes to unify security goals and their associated security policies with UML functional models [Oladimeji et al., 2006b], which can discover conflicts and inconsistencies in security policies at the early stage of system development.

Lamsweerde extends KAOS by introducing the notions of obstacle [Van Lamsweerde and Letier, 2000] and anti-goal [Lamsweerde, 2004] to analyze security requirements of a system. KAOS obstacles capture undesired states of affairs that prevent stakeholder goals from being satisfied, while anti-goals analyze obstacles that are intentionally imposed by attackers. KAOS leverages formal methods to systematically refine (security) goals that are specified in Linear Temporal Logic (LTL), using a set of predefined refinement patterns [Darimont and Van Lamsweerde, 1996]. After identifying specific threats via the anti-goal refinement, security requirements (i.e., countermeasures to the threats) are elicited accordingly. In addition, De Landtsheer and Van Lamsweerde [2005] have proposed a collection of KAOS specification patterns that codify families of confidentiality requirements, enabling automatic check of confidentiality violation.

Liu et al. [2003] propose a methodological framework for dealing with security and privacy requirements, which extends the *i** framework [Yu, 1997]. This approach places an emphasis on the security of organizational settings. In particular, their proposal can detect vulnerabilities in organizational relationships, can identify potential system abusers, and eventually can generate countermeasures so as to protect the system. Elahi et al. [2009] also advocate for incorporating vulnerability into security requirements analysis. They have proposed a vulnerability-centric security analysis approach [Elahi et al., 2010], which is also built on the *i** framework. This approach links empirical knowledge of vulnerability to system requirements models, and proposes a method to assess system risks. Specifically, the authors propose to take advantage of available vulnerability knowledge, such as CWE [MITRE-CWE] and CVE [MITRE-CVE]. In addition, Elahi and Yu [2007] have also developed an approach for analyzing security trade-offs and selecting the best design alternative once risk assessment results are available.

Mouratidis and Giorgini [2002] extend Tropos [Castro et al., 2001, 2002] with security concepts in order to integrate security concerns into the entire life cycle of system development, especially the early requirements stage. The resulting approach, Secure Tropos, has been continuously elaborated and extended for more than ten years, such as reported in [Mouratidis et al., 2004; Mouratidis and Giorgini, 2007b; Matulevičius et al., 2008; Mouratidis and Jurjens, 2010]. Recently, Mouratidis [2011] summarizes Security Tropos as an approach, which can be used to identify security requirements, transform the security requirements to design, and validate the designed system. In particular, a series of security concepts have been proposed and used to model security enhanced diagrams, including security enhanced actor diagram, security enhanced goal diagram, architectural style selection diagram, security attack scenarios diagram, and security reference diagram. A computer-aided tool (SecTro) has been developed to support graphical modeling of the above diagrams and to automatically generate some templates and diagrams that are required by the methodology [Pavlidis and Islam, 2011]. In addition, Mouratidis et al. [2006] use the extended security concepts to model security patterns, encapsulating reusable security knowledge. In another recent work, Mouratidis et al. [2013] enrich Secure Tropos with new concepts (e.g., cloud actor) to support the selection of cloud providers based on security and privacy requirements.

Giorgini et al. [2005b, 2006] emphasize the importance of analyzing organizational security requirements without getting into security protocols or security techniques. They have proposed a formal modeling framework SI* to capture the such security concerns, which extends Tropos [Bresciani et al., 2004] with notions of trust, permission, and ownership. On the basis of this modeling framework, they define a collection of formal reasoning rules for automatically checking security properties at the organizational level, e.g., need-to-know policies and conflicts of trust. A CASE tool (ST-Tool) has been developed to support modeling and verifying security requirements at the organizational level. This approach has been revised according to experiences gained from several industry case studies, the latest version of the SI* modeling framework and the Secure Tropos methodology[1] are presented in [Massacci et al., 2010]. Based on the SI* framework, Dalpiaz et al. [2011] propose SecCo, which explicitly models security needs via commitments and relates them to security requirements. Such commitment specifications are then used for designing secure business processes [Paja et al., 2012]. Paja et al. [2013] also propose a formal framework to automatically detect conflicts among security requirements that have been captured in SecCo. Another related work is performed by Asnar et al. [2011b], which extends SI* with a reasoning technique that identifies potential security threats on system assets.

UML-based approaches

Sindre and Opdahl [2005] extend traditional use cases to cover misuse cases, which describe behaviors that stakeholders do not want to occur. Based on the extended concept, the authors propose a systematic process for eliciting security requirements. To facilitate the analysis process, the authors have also defined an approach that constructs and reuses a repository of generic misuse cases [Sindre et al., 2003]. Similar to this approach, McDermott and Fox [1999] use *abuse cases* to capture harmful interactions to a system. As compared in [Wei, 2005], misuse cases can model a wider range of threats than abuse cases. In addition, misuse cases are modeled together with use cases instead of modeling separately like abuse cases. Lastly, misuse cases have been used by Firesmith [2003b] to identify security use cases. Several approaches have extended or applied misuse cases, such as executable misuse cases [Whittle et al., 2008], combining misuse cases with security goals [Okubo et al., 2009], and combining misuse cases with the Common Criteria [Ware et al., 2005].

UMLsec extends UML with security constructs (stereotypes, constraints, and tagged values), allowing analysts to express security concerns within the UML diagrams (e.g., use case diagram and activity diagram) [Jürjens, 2002, 2005]. As such, security can be

---

[1] This is different from the *Secure Tropos* approach proposed by Mouratidis et al.

designed and evaluated throughout the entire software development process. In particular, the author has defined formal semantics for each security construct in order to automatically verify security in a given specification. Although this approach mainly focuses on engineering security design rather than security requirements, it has been integrated with Secure Tropos so as to support the entire secure software development lifecycle [Mouratidis and Jurjens, 2010]. Similarly, another model-driven security design approach, SecureUML, has defined a security modeling language to formally design and verify role-based access control policies in UML diagrams [Lodderstedt et al., 2002].

Problem frame-based approaches

Crook et al. [2002] introduce the concept of anti-requirements, which are the requirements of malicious users and subvert existing system requirements. The authors then incorporate such anti-requirements into abuse frames [Lin et al., 2003b,a], which represent a set of undesirable phenomena (i.e., security threats) in Problem Frames [Jackson, 2001]. As such, analysts can elicit security requirements according to the threats captured in the abuse frames.

Hatebur et al. [2006] specialize Problem Frames into two kinds: Security Problem Frame (SPF) and Concertized Security Problem Frame (CSPF), in order to engineer system security. In particular, SPFs document general security problems, which are selected and instantiated to identify security requirements; CSPFs specify general security solutions to the security problems, which are selected and instantiated to design appropriate security mechanisms that satisfy the identified security requirements. In such a way, this approach explicitly distinguishes security problems from their solutions and leverage reusable security knowledge to facilitate analysis in both parts. On the basis of such extended frames, Hatebur et al. [2007] propose a systematic process for analyzing security requirements and solutions. Later on Schmidt [2010] further elaborates this analysis process by incorporating threat and risk analysis.

Haley et al. [2008] present a Security Requirements Engineering Framework (SREF) for security requirements elicitation and analysis. The authors use problem frames to represent system functional requirements, and model security constraints on top of the problem diagrams. The elicited security requirements are verified by using both outer (formal) and inner (informal) satisfaction arguments. Note that this framework clearly distinguishes security goals from security requirements, where the security goals are stakeholder's security concerns (e.g., confidentiality, integrity, etc.) and the security requirements are constraints on system functions. As the authors acknowledge the mutual impact between requirements and architecture, they define their security requirements analysis as an iterative process.

Summary of techniques

Table 2.2 summarizes and compares the security requirements engineering techniques we have reviewed. Although the approaches are designed for analyzing security requirements of software applications, some of them have already pointed out the importance of designing security at the organizational level. In particular, the approaches that are designed based on i*/Tropos inherit the capability of modeling social interactions and thus are suitable for analyzing organizational security. Apart from organizational security, UMLsec can analyze infrastructure security using the extended UML deployment diagram. To deal with security of STSs, we need to take into account security concerns of all components of STSs and to holistically select the best solution, which cannot be accommodated by existing approaches.

Most of the existing approaches incorporate threat analysis, but none of them can systematically analyze multistage attacks, which are an emerging challenge in designing secure STSs. By further examining existing approaches, we find out that goal-oriented approaches have inherent advantages in analyzing multistage attacks. In particular, their hierarchical AND/OR refinement structure allows analysts to capture different steps of multistage attacks. In other words, goal-oriented approaches are able to model multistage attacks when corresponding knowledge is available. However, currently there is limited knowledge about multistage attacks on STSs, especially since there are no approaches can analyze multistage attacks in a socio-technical setting. Therefore we need an approach that can not only import and represent multistage attacks in the threat analysis, but can also systematically **discover all** possible multistage attacks on STSs. As indicated in Table 2.2, the KAOS anti-goal approach can be used to identify multistage attacks via anti-goal refinement using requirements refinement patterns. However, as those refinement patterns are not initially designed for refining malicious intentions from an attacker's viewpoint, they cannot guarantee the completeness of the analysis results.

Reusing security knowledge has recently received a substantial increase of attention, as it can significantly ease the knowledge-intensive security analysis, such as threat identification and security requirement elicitation. As shown in Table 2.2, various ways for reusing security knowledge have been proposed. For example, Mouratidis et al. [2006] propose to reuse security requirements knowledge via security patterns; Sindre et al. [2003] propose to construct and reuse a repository of generic misuse cases for security requirements analysis. Although these proposals contribute to the methods of reusing security knowledge, less efforts have been made to promote the practical use of such methods. The *Source* column in Table 2.2 shows knowledge sources to be reused by each approach. In particular, most reuse-based approaches rely on *literature* in general for security knowledge, but do not investigate how to effectively use the knowledge. Instead, researchers normally make assumptions to simplify the problem, e.g., "*the proposed framework assumes that analysts have knowledge about vulnerabilities, potential attacks, and proper countermeasure or can obtain such information*" [Elahi et al., 2010]. However, comprehensive security knowledge repositories normally have an impressive scale, e.g., CAPEC has 504 attack patterns [Barnum and Sethi, 2007] and CWE include 719 weaknesses [MITRE-CWE]. Given the large body of security knowledge, analysts are reluctant to adopt them in practice without an efficient method [Shostack, 2014, p.106]. Thus, we argue that a systematic method and automated support are required in order to practically reuse security knowledge, which has also been concluded from a mapping study [Souag et al., 2015].

Because of the complexity of STSs, a holistic security analysis can result in large-scale models, consisting of hundreds of elements. As such, manual analysis is time-consuming and error-prone, and there is a strong need for automated reasoning and analysis. Although the surveyed existing approaches all have tool support for graphically modeling analysis models, only half of them can automate the security analysis on top of the models. In addition, the usability of tools remains as a challenge to the practical adoption of security analysis techniques [Massacci and Paci, 2012].

Table 2.2: A comparison of security requirements engineering techniques

Technique	Analyzed Artifact	Formal	Tool Support		Threat	Multistage Attack		Security Knowledge Reuse		
			Model	Infer		Rep.	Ide.	Proposal	Source	Support
NFR	Software	✓	✓	✓				1) Reuse taxonomy of security properties to support security goal refinement 2) Reuse taxonomy of security goal satisficing methods	1-2) Literature	1-2) The tool can support the application of the taxonomies
KAOS	Software	✓	✓	✓	✓	✓	✓			
Secure i*	Organization; Software	✓	✓	✓	✓	✓				
Elahi et al. [2010]	Organization; Software	✓	✓		✓	✓		Reuse empirical knowledge of vulnerabilities	CVE, CWE, SANS list	
Secure Tropos	Organization; Software		✓		✓	✓		1) Define reusable security patterns using the secure tropos modeling language 2) Develop a security catalog that can be reused to identify security requirements and security mechanisms	1) Four specific security patterns 2) Literature	1) Guidelines for applying the patterns 2) A tool for graphically model social settings security reference catalog
Secure Tropos (SI*)	Organization; Software	✓	✓	✓						
Misuse Cases	Software		✓		✓			Construct and reuse a repository of generic misuse cases for security requirements analysis	Literature	Guidelines for reusing the repository
UMLsec	Software; Infrastructure	✓	✓	✓	✓					
SecureUML	Software	✓	✓	✓						
Abuse Frame	Machine	✓	✓		✓			Abuse frame diagrams can be reused for threat analysis	Literature	
SEPP	Machine	✓	✓		✓			Security problem frames and concretized security problem frames can be reused for security requirements analysis	Three specific security problem frames	Guidelines for applying the security problem frames
SREF	Machine	✓	✓		✓					

2.2 Holistic Security

To holistically design secure STSs, security requirements of all components of STSs should be elicited and analyzed. As we have reviewed in the last section, typical security requirements analysis techniques focus on analyzing security requirements of software, some of which can also analyze security requirements of organizational settings. In this section, we first review approaches that focus on designing secure business processes and secure physical infrastructure, which are beyond the software-oriented security requirements analysis approaches. To be noted that some software-oriented approaches, e.g., Misuse cases, can also be applied in general to analyze security for different artifacts. However, we here focus on specialized approaches that can provide in-depth understanding about security in business processes and physical infrastructure. After that we examine approaches that holistically deal with system security.

2.2.1 Security Analysis beyond Software Systems

Business process security

Providing security in business process design has been recognized as important [Herrmann and Pernul, 1998]. Different researchers have been investigating what kind of security requirements need to be enforced during business management, what are the semantics of those security requirements, and how to verify and enforce those requirements. However, none of these approaches can capture alternative security requirements and perform trade-off analysis.

Rodríguez et al. [2007a] extend BPMN [Group, 2011] with security notations so as to model security requirements within business processes. The authors define seven security requirements, which can be annotated to specific BPMN constructs. In particular, they use OCL (Object Constraint Language) to unambiguously specify restrictions imposed by the security requirements. In subsequent research [Rodríguez et al., 2011], the authors extends UML 2.0 activity diagrams with similar security requirements concepts. Moreover, they propose a model-driven approach M-BPsec, which transforms the extended activity diagrams (Computation Independent Model – CIM) into a set of UML artifacts used in software development (Platform Independent Model – PIM).

Altuhhova et al. [2012] analyze BPMN with respect to Information System Security Risk Management (ISSRM) model [Dubois et al., 2010], investigating how to express security concepts (e.g., assets, threats) with original BPMN constructs. The authors provide an alignment of the ISSRM concepts and the BPMN constructs, allowing them to annotate security issues on top of BPMN models. As such, they are able to analyze and design secure business processes by following the ISSRM process.

Meland and Gjære [2012] emphasize the need of expressing threats and unwanted incidents on BPMN 2.0 models. They propose a process-centric threat modeling approach that can analyze which threat events trigger a deviation from the typical business process flow. A triggered threat will provoke a re-composition and/or reduced functionality of the composite services. To keep the number of BPMN constructs to a minimum, they use existing constructs to define unwanted events in business processes instead of inventing new constructs.

Salnitri et al. [2015] propose SecBPMN-ml as an extension of BPMN for modeling security aspects in business processes. Several security annotations are included in SecBPMN-ml, the semantics of which have been formally specified. In addition, the authors define the SecBPMN-Q query language for representing security policies, which can be automatically checked against SecBPMN-ml specifications.

Physical infrastructure security

Physical security protections highly depend on the specific physical devices. Different devices involve different vulnerabilities and thus can be attacked in different ways, such as attacks against smart meters [Flick and Morehouse, 2010, Chap.12] and advanced metering infrastructure [Carpenter et al., 2009]. As such, instead of modeling and abstracting common features of different physical devices, most physical security analysis is performed in an ad-hoc way that focuses on analyzing particular devices.

There are a few model-based approaches that model and analyze the infrastructure security. Jürjens [2002] considers physical layer in UMLsec, in which the author specifies and verifies secure concerns in deployment diagrams. Specifically, this work focuses on the dependencies between components deployed in different nodes, as well as the location of the nodes.

Ustun et al. [2006] propose an agent-based conceptual design of a physical system security simulation, which allows analysts to identify potential threats to physical security systems. According to the simulation results, a security configuration will be generated to minimize risks, which includes the physical structure of the facility, the set of sensors included in the facility, and the set of guards and their respective operating/patrol strategies.

Fernandez et al. [2007b] examined existing systems, industry standards and government regulations, based on which they have summarized a core set of features that a physical access control system should have. Such knowledge is specified in terms of security patterns, which can be reused to develop physical security. In particular, they have created and illustrated three specific patterns: *Alarm Monitoring*, *Relays*, *Access Control to Physical Structures.*

2.2.2 Holistic Security Analysis Approaches

Mouratidis and Jurjens [2010] analyze security from both organizational and technical perspectives by combining existing approaches Secure Tropos and UMLsec. The authors provide a systematic process to transfer the elicited organizational security requirements (i.e., Security Tropos models) to secure software design models (i.e., UMLsec class diagrams and deployment diagrams). Salnitri et al. [2014b] propose an incremental and iterative process to align business process policies with organizational security requirements. By automatically verifying such policies, this approach ensures the organizational security requirements are preserved in the business layer. Other approaches [Menzel et al., 2009; Rodríguez et al., 2010] apply model-driven techniques to transform security requirements captured in business processes into concrete security configurations in software design. All the above approaches cover multiple aspects of system security and align security requirements in a top-down manner. However, these approaches do not holistically analyze alternative security requirements across different aspects. As such, the analysis result may not be the optimal solution.

Apart from the alignment-based approach, several studies have been done to structure information systems into multiple conceptual layers. May and Dhillon [2010] propose a holistic conceptual framework to identify security issues from both social and technical perspectives. The authors contend that information security is becoming a multidimensional discipline, where particular models can only address part of the security issues and a meta-theoretical basis is required to achieve a holistic understanding of information security. As such, they base their framework on the theory of semiotics, which consists of six layers of abstraction. By mapping information security issues into these six layers, their framework can theoretically and intuitively guide holistic security analysis process. Similarly, Wimmer and Von Bredow [2002] propose a holistic approach to generate security solutions in e-government. This proposal consists of four levels, including community level, process level, interaction level, and infrastructure level. The authors argue that each of these four levels has different security requirements, and they exemplify security solutions at each level. However, these approaches do not systematically analyze interactions among different layers. In addition, the approaches do not have any automated analysis support, and thus cannot be pragmatically used to analyze large-scale systems.

Zuccato [2004] introduces a holistic security requirements engineering approach for electronic commerce. The approach synthesizes three paradigms that are commonly used to elicit security requirements in order to have a complete set of requirements. In such a way, the author claims this approach can analyze holistic security requirements. In particular, the approach first collects security requirements from risk analysis, business process analysis, and workshops with stakeholders, respectively; and then compiles such requirements by solving conflicts among them. The author provides detailed instructions for performing this holistic security requirements analysis. Instead of analyzing different components of a system, this approach implements *holistic analysis* from another dimension, i.e., using multiple elicitation paradigms.

Spears [2005] describes a systematic process for holistically analyzing security risks. This proposal starts from identifying business functions, and then consecutively identifies corresponding critical business processes, information systems, and supporting infrastructure. After that the author proposes to analyze threats and vulnerabilities and to develop risk scenarios for each of the critical assets, in order to perform a holistic analysis. Although this approach can cover different security concerns pertain to the information systems, it cannot holistically select the best security solution. In addition, this approach does not have any automated analysis support.

2.3 Security Attack Analysis

In this section, we review approaches that analyze security attacks from three perspectives. Firstly, we focus on approaches that are based on attacker analysis. Secondly, we review papers that leverage reusable attack patterns. Thirdly, we survey approaches that are designed to analyze multistage attacks.

2.3.1 Attacker-oriented Analysis

Steele and Jia [2008] leverage User-Centered Design techniques to develop personas, describing the archetypal behavior of possible attackers. Apart from this speculative proposal, Atzeni et al. [2011] propose a grounded approach for developing attacker personas. They collect realistic data from people who have been known to attack systems, based on which they develop personas. In addition, their approach includes an argumentation model, which associates grounded arguments to specific characteristics of attackers. As such, the attacker personas can be better understood by analysts and can be revised in light of different contexts. These personas-based approaches help to easily comprehend attackers and identify attack scenarios, especially for non-security analysts.

Another branch of attacker analysis advocates analyzing attacks from an attacker's viewpoint. Several approaches have been proposed to analyze attacks at the operational level, i.e., modeling behaviors of attackers. Such as the attack tree [Schneier, 1999], misuse cases [Sindre and Opdahl, 2001, 2005], and abuse cases McDermott and Fox [1999]. Beyond the operational level, later studies capture intention of attackers in order to understand why the malicious behaviors are performed by attackers. Crook et al. [2002]

capture the requirements of a malicious user that subvert an existing requirement as anti-requirements. Later on, Lin et al. [2003a,b] incorporate such anti-requirements into abuse frames to represent threats and analyze security requirements.

Lamsweerde [2004] proposes to use anti-goals to model an attacker's malicious intention, by refining which alternative attacks can be identified. Specifically, the author leverages formal goal refinement patterns [Darimont and Van Lamsweerde, 1996], which were initially designed for refining requirements goals, to refine anti-goals. By operationalizing each leaf anti-goal in terms of specific attack actions, the approach eventually can discover alternative attack scenarios. This approach is developed in the context of the goal modeling approach KAOS, and it has been formalized with automated analysis support. Similar approaches have also been proposed to model an attacker's intention using goal modeling languages [Liu et al., 2003; Mouratidis et al., 2004; Elahi et al., 2010]. Instead of identifying attacks, these approaches focus on analyzing the influences of modeled alternative attacks on a system using goal satisfaction analysis techniques. However, all of these intention-oriented approaches are not grounded in realistic security knowledge, which requires analysts to have in-depth understanding about attackers in order to effectively apply these approaches.

2.3.2 Attack Pattern-based Analysis

Moore et al. [2001] emphasize the importance of reusing known attack knowledge, which significantly affects the practicality of attack analysis methods (e.g., attack tree). As such, they first define attack patterns to support the knowledge reuse. In particular, each pattern consists of four sections: goal, precondition, attack steps, and post-condition. In subsequent research, Bozic and Wotawa [2014] use this structure to define attack patterns and formalize the malicious actions of the attack patterns. In this way, the approach can automate the execution of attack patterns and test system security. Apart from this attack pattern template, several researchers have defined attack patterns using their own templates. Gegick and Williams [2005] define software attack patterns in term of a sequence of events, using regular expressions. Specifically, each event is expressed by its associated component, such as user, server, hard disk, etc. By automatically matching such patterns with system design, the approach can assist analysts in identifying system vulnerabilities. Fernandez et al. [2007a, 2009] specify attack patterns (i.e., misuse pattern) based on POSA template [Buschmann et al., 1996]. The POSA template includes much more sections than the initial one defined in [Moore et al., 2001], such as context, known uses, countermeasures, etc., which contribute to the practicality of attack pattern-based analysis. Although the above approaches contribute to the theoretical foundation of attack patterns, they have not been pragmatically applied to develop attack patterns.

For example, Moore et al. [2001] illustrate their approach with four patterns and we are unaware of subsequent work has been done for developing patterns; Fernandez-Buglioni [2013] has only developed three misuse patterns, as reported in his recent book.

Compared to the above theoretical approaches, the Common Attack Pattern Enumeration and Classification (CAPEC) is initiated as a baseline catalog of attack patterns along with a comprehensive schema and classification taxonomy [Barnum and Sethi, 2007]. CAPEC was launched in 2007 and has been consecutively developed, which currently includes 504 attack patterns[2]. Since CAPEC provides a significant amount of practical security knowledge, it is receiving an increase of attention from both academia and industry. Especially, a main research question is how to effectively use CAPEC, given its impressive size. Kaiya et al. [2014] define term-maps, which link terms in requirements specifications to specific security terms used in CAPEC. As such, this approach can automatically identify related attack patterns based on requirements specifications, and thus further obtains corresponding security requirements. Engebretson and Pauli [2009] enrich the CAPEC attack patterns with the concepts *parent threat* and *parent mitigation* in order to facilitate the navigation among the large number of attack patterns. Yuan et al. [2014] map CAPEC patterns to STRIDE [Shostack, 2014], based on which they develop a tool to facilitate the retrieval of CAPEC patterns. However, all the above approaches do not use context to check the applicability of attack patterns. In addition, the CAPEC patterns focus on documenting operational attack knowledge, and currently there are no approaches that associate such patterns with attacker intention analysis, limiting the utility of CAPEC patterns.

2.3.3 Multistage Attack Analysis

An attack graph shows all paths through a system that end in a state where an attacker achieves his malicious intention. Each attack path indicates a potential attack that consists of one or several steps, allowing analysts to analyze multistage attacks. Different techniques have been proposed to automatically generate such graphs. Phillips and Swiler [1998] first use attack graphs to analyze network security. Because of the homogeneous settings of machines in the network, the states of machines (i.e., nodes in the attack graph) and the atomic attacks on machines (i.e., transitions in the attack graph) can be enumerated. As such, it is possible to automatically generate all the attack paths by following a comparatively simple attack strategy. Take the approach of [Sheyner et al., 2002], for example: an attacker starts from a machine with the root permission, he then iteratively detects the next vulnerable machine in the network, logs into that machine, and gets the root permission of that machine until reaching his target machine. In a

---

[2] `https://capec.mitre.org`

recent study, Beckers et al. [2015] apply the attack graph technique to analyze social engineering attacks, where the states of people are modeled as nodes and social engineering attacks are captured as transitions between nodes. However, the attack graph approach only applies to systems that have simple and homogeneous components, where the states of components and relevant actions can be enumerated, i.e., without combinatorial explosions. Thus, it is not suitable for security analysis of STSs which can involve a large number of heterogeneous components, such as people, software, and hardware.

Attack trees are a typical way of representing alternative attacks, which can also capture multistage attacks due to their hierarchical tree structure. Thus, the systematic construction of such trees contributes to the identification of multistage attacks. Although there is no unique way of creating attack trees, different researchers have proposed several guidelines/methods. Morais et al. [2013] advocate a guideline for creating attack trees, which starts from modeling the general attack description; and then identifies the violated security properties and the security mechanisms to be exploited, respectively; and finally models the concrete attack actions. Paul [2014] proposes a layer-per-layer approach to automatically generate skeletons of attack trees using information derived from system architecture, risk assessment study, and related security knowledge base. However, as the attack tree approaches only focus on operational attack actions and do not capture an attacker's malicious intentions, they fail to identify the variety of attacks at the strategic level.

Several studies have been proposed to capture attacker's malicious intentions as (anti-)goals using goal-oriented modeling language, e.g., [Lamsweerde, 2004; Liu et al., 2003; Mouratidis et al., 2004]. As such, analysts can well understand both *how* and *why* attacks are performed, and thus better explore the space of attack alternatives. In addition, the tree structure of goal models allows analysts to capture multistage attacks. However, these approaches are not grounded in realistic security knowledge, requiring analysts to have a strong security background in order to effectively model and analyze attacks.

2.4 Security Patterns

Security patterns encapsulate reusable security knowledge which can assist analysts with little security knowledge in identifying security solutions to satisfying security requirements. We here first review existing security pattern repositories with the aim of understanding the situation of security pattern development. In particular, we try to know the total number of security patterns that have been developed. After that we survey existing techniques to investigate how to effectively select the best security pattern among alternatives. Finally, we review approaches that analyze the impact of applied security mechanisms (i.e., solutions of security patterns).

2.4.1 Security Pattern Collections

Yoder and Barcalow [1997] wrote the first paper on security patterns, which includes seven patterns. The authors specify these patterns in a structured manner using the GoF template [Gamma et al., 1994] and describe interactions among patterns in order to fit the seven patterns all together. After that different researchers have been working on the discovery, documentation, and application of security patterns, resulting in several security pattern collections that include a significant number of patterns.

Schumacher et al. [2006] published a book in 2006, which advocates using security patterns to integrate security and software engineering. In particular, the authors introduce 44 security patterns in details, which are specified using the POSA template [Buschmann et al., 2007], and illustrate the usage of security patterns via case studies. In a recent textbook, Fernandez-Buglioni [2013] summarizes in total 68 security patterns that have been written by himself, which are also specified with the POSA template. Some of these patterns have appeared in [Schumacher et al., 2006], but have been revised by Fernandez based on his experiences and security knowledge.

Yskout et al. [2006] gather a collection of security patterns based on an extensive survey of security patterns in literature in order to form a system of security patterns. The collected patterns are screened based on their complexity, quality, and abstraction level, leading to 35 core security patterns that concentrate on software architecture and detailed design. The authors document the patterns in a specific security pattern template, which is extended from the GoF template.

Asnar et al. [2011a] present a process-oriented approach for capturing, validating, and applying security and dependability organizational patterns. By applying this approach within an industry lead EU project, an organizational pattern library has been established based on a number of case studies of the project. The library covers a broad spectrum of security management issues, e.g., legality, privacy, security, etc., totally including 49 organizational patterns. Each pattern is defined as a triple <*Context; Requirement; Solution*>, the elements of which are specified by using the SI* framework [Massacci et al., 2010].

Given the different collections of security patterns, it is difficult to tell the exact number of security patterns that have been developed. For one thing, the above reviewed security pattern repositories cannot be exhaustive. For another thing, as existing security patterns can cover different levels of abstraction, there is no consensus on the definition of security pattern. For example, Heyman et al. [2007] argue that some existing security patterns are too abstract to be considered as actual patterns (e.g., *Asset Valuation*), which are closer to security guidelines or principles. They have performed an extensive survey over

220 patterns, among which 55% are classified as core patterns, 35% are guidelines and principles, and 10% are process activities. However, the definition of security patterns is out of the scope of this thesis. Apart from this survey, in a recent mapping study, Ito et al. [2015] acknowledge that currently there are more than 200 security patterns.

2.4.2 Security Pattern Selection

Scandariato et al. [2008] propose a methodology which can help analysts to systematically select security patterns to develop secure systems. This methodology is built on 35 well-documented security patterns [Yskout et al., 2006]. In particular, the 35 patterns have been classified in two ways: development phases (architecture, design, etc.) and security objectives (confidentiality, availability, etc.). Using such classifications, analysts are able to navigate among the 35 patterns and identify candidates for application. Moreover, the authors also associate quality labels with each security pattern, in order to assist analysts in selecting the best security patterns among candidates, if there are more than one. Lastly, the approach captures interrelations among patterns, such as *depend* and *impair*, etc., which further complement the selection of security patterns. However, there is no tool has been developed to support the application of this methodology, requiring analysts to manually go through the entire analysis process.

Araujo and Weiss [2002] argue that selecting the best security patterns among all the applicable ones is a knowledge-intensive process, as analysts have to thoroughly read and understand all the alternative patterns. Therefore, the authors apply the Non-Functional Requirement (NFR) framework as a complementary representation for security patterns in order to help analysts to analyze trade-offs among candidate patterns. Their approach captures *forces* of security patterns with contribution links to softgoals. In particular, they have defined *force hierarchy* to capture the interactions among forces at different level. As the root force is imposed on the entire system, analysts are able to compare forces of all candidate security patterns within a particular force hierarchy, and assess the total satisfaction of the root force so as to choose the best alternative. The authors have practically applied their approach to enrich 14 security patterns with the force hierarchy. In subsequent research, Mussbacher et al. [2006] formalize this approach and enable automatic trade-off analysis with tool support. However, this approach does not support analyzing the application context of security patterns, and thus analysts have to first manually identify applicable patterns among a large number of security patterns.

Hafiz and Johnson [2006] survey existing security patterns and their classification schemata. Specifically, the authors compare different classification schemata (e.g., logical tiers, system viewpoint, and security concepts, etc.), and discuss their advantages and disadvantages. Considering the evidence from the comparison, the authors construct

Table 2.3: A comparison of security pattern selection techniques

Technique	**Pattern**	**Identify Applicable Patterns**				**Trade-off**	
	Number	**Classification**	**Interrelationship**	**Applicability**	**Tool**	**Method**	**Tool**
[Scandariato et al., 2008]	35	1) development phase 2) security objective	five inter-pattern relationships	manual assessment	no	quality trade-off labels	no
[Araujo and Weiss, 2002]	14	–	–	–	–	NFR trade-off analysis	yes
[Hafiz et al., 2012]	96	1) threat 2) system levels	a full pattern language for all patterns	manual assessment	no	–	–
[Fernandez-Buglioni, 2013]	68	multiple-dimensional classification	separate pattern diagrams for specific patterns	manual assessment	no	incorporate risk analysis techniques	no

an organization of security patterns based on threat model (STRIDE) and system levels, [Hafiz et al., 2007]. In subsequent work, Hafiz et al. [2012] further improve their pattern schema by defining a comprehensive pattern language, involving all the 96 patterns in the pattern organization. A pattern language describes the interrelations among patterns and offers analysts a guidance in selecting the next pattern to consider.

Fernandez-Buglioni [2013] has proposed a systematic process for selecting and applying security patterns during the entire lifecycle of software development. To select appropriate patterns to apply, he leverages a multi-dimensional classification schema to categorize patterns, e.g., via architectural levels, lifecycle stages, and domains, etc. [VanHilst et al., 2009]. In addition, the approach also captures interactions among security patterns using pattern diagrams, in order to facilitate the pattern selection. Specifically, the *abstract/instantiate* relations among security patterns have been emphasized to play an important role when building pattern diagrams [Fernandez et al., 2008].

Summary

Table 2.3 summarizes and compares the four approaches that we reviewed before. To select the best security pattern, we have identified two main steps: firstly, identify applicable patterns from the pattern catalog; secondly, perform trade-off analysis on all the applicable patterns in order to select the best one.

In the first step of analysis, all the approaches rely on pattern classifications to navigate through the entire pattern catalog. In addition, the interrelationships among patterns have been investigated to different extents, assisting analysts in identify relevant patterns that can be applied. The above two types of support can help to reduce the range of the pattern catalog, as they actually capture part of context of security patterns. However, such supporting techniques cannot include all kinds of context, and thus analysts still have to manually assess the applicability of security patterns. In addition, none of the reviewed approaches has tool support for identifying applicable patterns.

In the second step of analysis, the NFR-based approach is dedicated to analyze trade-offs among candidate patterns, which can be automated with a support tool. On the contrary, other approaches acknowledge the need of performing such trade-off analysis, but do not specify analysis methods nor have tool support.

2.4.3 Impact of applying security patterns

The application of a security pattern amounts to applying a security mechanism (i.e., the solution of the security pattern) to fulfill a security requirement (i.e., the problem of the security pattern). However, as a side effect of such application, the applied security mechanism will inevitably impact the original system requirements (both security and non-security requirements), which has been acknowledged and taken into account by several security analysis approaches. For example, Hatebur et al. [2007] deal with the impact of security mechanisms by iteratively performing the security requirements analysis and security mechanism analysis. In particular, if the precondition of an applied security mechanism (encapsulated in concretized security problem frames) cannot be satisfied, then new security requirements are derived from that precondition, demanding a new round of security analysis.

Nuseibeh [2001] first proposes a requirements *Twin Peaks* model to demonstrate the interactions between requirements and architecture at an abstract level. Heyman et al. [2011] specialize the twin peaks model in the security area, leading to a *Security Twin Peaks* model that emphasizes the bi-directional impact between security requirements and security architectural design. Building on this conceptual model, the authors propose a constructive process for co-developing secure software architectures and security requirements using security patterns. In particular, they identify three key notations for co-development, which should be incorporated in the specification of security patterns: firstly, the *components and behavioral requirements* that are introduced by a security pattern; secondly, the *roles and expectations* that explain how the newly introduced components interact with existing ones; thirdly, the residual goals that are considered by a security pattern and need to be taken into account when instantiating the pattern.

Similarly, built on the Twin Peaks model, Okubo et al. [2012] propose a method *TMP-SA* (Twin Peaks model Application for Security Analysis) which elicits security requirements during the elaboration of software architecture. In particular, the authors implement the mutual refinement process on top of their previous security analysis framework *MASG* (Misuse case with Assets and Security Goals) [Okubo et al., 2009], which involves security concepts *asset*, *security goal*, *threat*, *attack*, and *countermeasure*. To unify the analysis within both peaks, the authors consider not only *architecture-independent* artifacts but also *architecture-specific* artifacts. For example, the approach can detect architecture-specific threats, from which countermeasures are derived.

Although the above reviewed approaches contribute to the analysis of impact of security mechanisms, they focus on only the impact imposed on security requirements and omit the impact on non-security requirements (either functional or non-functional requirements). In addition, all theses proposals are performed manually, which cannot be applied to analyze large-scale systems.

2.5 Chapter Summary

In this chapter, we first present an in-depth survey on the state of the art in the area of security requirements engineering in Section 2.1, from which we disclose several gaps in analyzing security requirements of STSs. In light of such gaps, we specifically survey related techniques. In Section 2.2 we review security analysis techniques beyond software which can be incorporated in the holistic approach in order to tackle security issues for particular artifacts, as well as approaches that are intended to holistically analyze system security. Next, we examine existing approaches which profile attackers, reuse attack knowledge, or deal with multistage attacks in Section 2.3. Such examination helps us to get a deep understanding of the challenges in holistically analyzing attacks of STSs. Finally, in Section 2.4, we survey existing security pattern repositories, as well as techniques that select security patterns, shedding light on how to effectively reuse existing knowledge. We also pay attention to the impact of security mechanisms imposed on system requirements and discuss related approaches.

Chapter 3

Baselines

> *Creation always involves building upon something else. There is no art that doesn't reuse. And there will be less art if every reuse is taxed by the appropriator.*
>
> Lawrence Lessig

In this chapter, we describe existing techniques, based on which we develop our framework. We first introduce the requirements problem, which specifies fundamental tasks that need to be addressed during requirements analysis (Section 3.1). Then, we describe the goal-oriented requirements modeling languages i* [Yu, 1997] and Techne [Jureta et al., 2010], based on which we develop the three-layer requirements modeling language (Section 3.2). In addition, we also introduce a contextual goal modeling language [Ali et al., 2010], which is used for modeling attack patterns and security patterns. After that we present existing security pattern repositories, which help analysts to reuse provable security knowledge (Section 3.3). Lastly, we describe CAPEC attack patterns which we use to identify realistic attacks (Section 3.4).

3.1 Requirements Problem

Zave and Jackson [1997] define a Requirements Engineering (RE) ontology in order to specify the requirements problem. Their definition consists of three concepts: a *Requirement* is an optative property that specifies stakeholder needs, expected to be satisfied by the system-to-be; a *Domain Assumption* is an indicative property that is relevant to the system-to-be; a *Specification* is an optative property, which can be directly implemented by the system-to-be in order to satisfy stakeholder needs. On the basis of these three concepts, the requirements problem amounts to finding a collection of specifications S, which can satisfy all requirements R under domain assumptions D. Thus, the requirements problem is represented as $D, S \vdash R$, meaning that domain assumptions and specifications together entail the requirements.

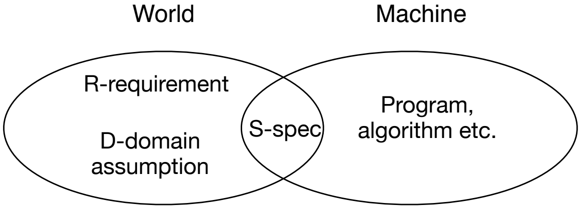


Figure 3.1: An illustration of the requirements problem from [Zave and Jackson, 1997]

As illustrated in Fig. 3.1, requirements are phenomena in the problem world, and specifications offer a way for phenomena in the system-to-be to satisfy the requirements. Due to the complexity of STSs, the requirements problem needs to be extended to account for phenomena in all the three conceptual layers. We will describe such extensions in detail in Chapter 4.

3.2 Goal Modeling Languages

Goal-Oriented Requirements Engineering has received much attention in RE research, as an intuitive means of capturing stakeholder goals and understanding underlying motivations for system requirements [Horkoff and Yu, 2011]. Over the last two decades, many Goal Modeling Languages (GML) have been proposed to model and analyze requirements, such as NFR [Chung, 1991], i* [Yu, 1997], KAOS [Dardenne et al., 1993], Techne [Jureta et al., 2010], etc. Specifically, several advantages are offered by such GMLs, which fit well our needs of dealing with holistic security requirements of STSs.

- GMLs capture stakeholder requirements as goals which can be and/or refined to detailed ones. Such *and-or tree* structure can capture alternative solutions that satisfy stakeholder requirements. In particular, many satisfaction analysis techniques have been proposed in order to select the best alternative for fulfilling root level goals [Horkoff and Yu, 2013]. By inheriting such a feature from GMLs, we are able to capture and analyze alternative holistic security solutions for STSs.
- The social aspect of security requirements analysis has been paid an increasing attention in the past fifteen years, which is particularly important for STSs. The i* modeling language [Yu, 1997] has emphasized the need of social modeling in requirements analysis. In particular, it offers concepts and relations to model social actors and their dependency. Many security requirements analysis approaches have been

proposed based on i* so as to analyze social and organizational security issues, such as [Mouratidis and Giorgini, 2002; Liu et al., 2003; Giorgini et al., 2005b]. In this thesis, we also base our approach on such social modeling constructs.

- Techne [Jureta et al., 2010] is a recently proposed requirements modeling language, which maps Zave and Jackson's requirements ontology into goal model concepts. As such, it provides the theoretical foundation of using GMLs to solve the requirements problem. In addition, it extends the requirements problem with priorities among stakeholder requirements, which further enhance the selection among alternative requirements specifications [Horkoff et al., 2014a]. Since we intend to tackle the requirements problem of STSs, we adopt the mapping and corresponding constructs defined in Techne.

In the remainder of this section, we first introduce the two particular GMLs (i.e., i* and Techne), on the basis of which we develop our three-layer requirements modeling language (Chapter 4). After that, we introduce a contextual goal modeling language, which extends a goal modeling language (Tropos[Bresciani et al., 2004]) with notions of context in order to model and analyze requirements in different contexts [Ali et al., 2010]. In particular, we leverage this extended modeling language to model attack patterns (Chapter 5) and security patterns (Chapter 6) in terms of goal models with context, enabling semi-automatic context analysis for those patterns.

3.2.1 i*

i* was developed by Yu [1997] as a goal- and agent-oriented modeling and reasoning framework. Different from other GMLs, such as NFR or KAOS, i* promotes the notion of *actor* as a first-class citizen, where an actor is an active and autonomous entity involved in a system. In particular, i* can model and analyze requirement goals for system actors from their perspectives via Strategic Rationale (SR) diagrams, as well as capture the social interactions among actors via Strategic Dependency (SD) diagrams.

In order to model an actor's strategic rationale, four intentional elements have been defined, i.e., *goal*, *task*, *softgoal*, and *resource*, which are essential to the *i** modeling language. A *goal* is a state of affairs that is desired by an actor and has clear-cut criteria of achievement; a *softgoal* is similar to a goal but does not have clear-cut criteria for its satisfaction; a *task* represents an action that is executed by an actor in order for achieving some goal; a *resource* is an entity (either physical or informational) that is required by an actor in order to perform a task. In particular, the achievement of such intentional elements is captured using *means-end links*, which shows how a *means* (i.e., a task) can be used to reach an *end*. In particular, the *end* can be a goal to achieve, a task to accomplish, a resource to produce, or a softgoal to satisfice. In addition, a task can be decomposed into one or several sub-elements via the *task decomposition links*. Lastly, all such intentional elements can be linked to a softgoal using *contribution links*, indicating to what extent the intentional element contributes to the fulfillment of the softgoal. Such contribution links are typically used to evaluate and compare alternative requirements specifications.

An actor's goals can be fulfilled either by herself or by other actors. The former case is captured by using the decomposition and means-end links described above, while the later case is modeled by using *dependency links*. In particular, i* defines four types of dependencies, each of which corresponds to one particular intentional element, such as goal-dependency. The i* strategic dependency diagram is intended to capture such dependency relationships among actors, presenting the interaction network in an organization.

The i* models are typically developed during early requirements stage, helping to understand why a new system is needed. In addition, on top of the i* models, techniques have been developed to capture and select solutions in goal models, such as [Horkoff and Yu, 2010]. The i* framework has been widely adopted by the research community in fields such as requirements engineering and business modeling. Specifically, Tropos [Bresciani et al., 2004], as an agent-oriented software engineering methodology, has been built on the i* meta-model and supports the entire system development lifecycle using intentional goal models.

3.2.2 Techne

Jureta et al. [2010] propose Techne as a new generation of requirements modeling language, which is defined based on a revised requirements core ontology [Jureta et al., 2008]. The concepts defined in Techne can not only represent the requirements problem defined by Zave and Jackson [1997] (introduced in Section 3.1), but also extend it with quality and preference in order to better compare alternative solutions. In particular, stakeholder's requirements R is captured as *goals* and *softgoals*, while system specification S includes *tasks* and *quality constraints* which operationalize *goals* and *softgoals*, respectively. Moreover, Techne explicitly captures *domain assumptions* D.

To better deal with the selection of alternatives, stakeholder's *preferences* are taken into account, which are modeled by preference relations between requirements. Specifically, a *preference* relation means one requirement is strictly more desirable than another. As reported in [Horkoff et al., 2014a], by capturing stakeholder's preferences, Techne reasoning is able to choose between alternatives in more cases than i* reasoning, providing enhanced reasoning power.

It is worth noting that Techne does not include graphical modeling notations but an abstract requirements modeling language, which provides the formal foundations for new modeling languages applicable during early phases of the requirements engineering process. Thus, when defining our modeling framework in Chapter 4, we borrow concepts from Techne and define corresponding graphical notations by ourselves (mainly based on the i* modeling notations).

3.2.3 A Contextual Goal Modeling Language

Ali et al. [2010] argue that requirements should be analyzed in a way that reflects context settings, as stakeholder's requirements can vary from context to context. As such, they have proposed a goal-based framework for contextual requirements modeling and analysis, in which they relate goals and contexts. In particular, contexts are treated as labels that can be attached to specific goal model elements, such as shown in Fig. 3.2. Moreover, they define semantics for contexts that are modeled within goal models. For example, in the first case of Fig. 3.2, goal *G* represents a requirement if and only if context *C* holds. In the other two parts of the figure, a link between two goals is part of the goal model only when context *C* holds.

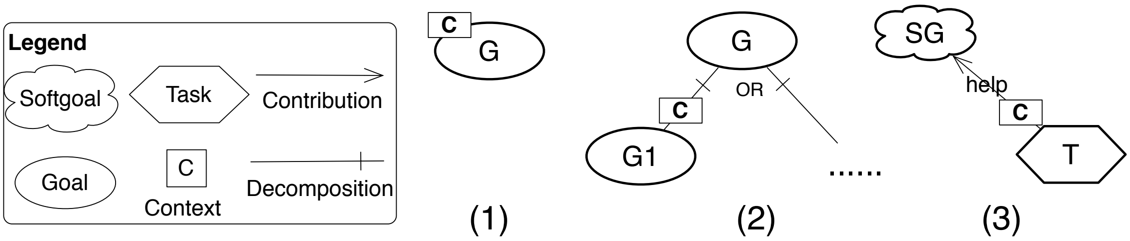


Figure 3.2: Goal models with contexts

In this thesis, we use this contextual goal modeling language to model attack patterns and security patterns. More specifically, we follow their approach to model contexts, as illustrated in the Fig. 3.2. It is worth noting that this modeling language extends Tropos, which is built on the i* meta-model. Since we have decided to also base our three-layer requirements modeling language on i*, the constructed contextual goal models can be easily and seamlessly integrated with the three-layer requirements models.

3.3 Security Patterns

Security patterns consist of <security requirement (problem), security mechanism (solution)> pairs that capture particular ways of solving known security problems. Since patterns constitute an effective way to encapsulate security expertise, they can significantly help analysts who have little security knowledge to perform security analysis. As introduced in Section 2.4.1, many security patterns have been proposed at different abstraction levels and several pattern repositories have been established, such as the work done by Asnar et al. [2011a] and Fernandez-Buglioni [2013].

In this thesis, we propose to make use of security patterns to operationalize security goals in terms of security mechanisms within each of the three layers. In particular, our approach not only refers to the security pattern repositories, but seamlessly integrates those patterns as part of our framework in order to *practically* apply such patterns in an effective manner. In this section, we describe existing security patterns in detail. Specifically, we describe essential concepts of security patterns, which play an important role in our security requirements analysis.

3.3.1 Pattern Templates

Patterns are structured documents that capture proven solutions for recurring problems. Different templates have been proposed for documenting patterns. Alexander et al. [1977] first define a pattern as "*a three-part rule, which expresses a relation between a certain context, a problem, and a solution*". In the same spirit of Alexander's proposal, Gamma et al. [1994] develop design patterns to support software design, in which they define Gang of Four (GoF) template for specifying patterns. Similarly, Buschmann et al. [1996] propose pattern-oriented software architecture and define POSA template; Coplien [1996] define software patterns with his own template (i.e., Coplien template), which has been further used for specifying organizational patterns [Coplien and Harrison, 2004]. These subsequent approaches have enriched the initial pattern templates (defined by Alexander) with specific sections according to their own purposes. For example, the POSA template has an *Implementation* section to support the downstream application of a pattern, while Coplien template has a *Resulting Context* section which describes the influences of a pattern in detail.

As different templates have their own focuses, they are used by different security pattern developers for particular purposes. For example, Fernandez-Buglioni [2013] specifies security patterns based on the POSA template, while Scandariato et al. [2008] use the GoF template and extend it with security aspects. As pointed out by Mowbray and Malveau [1997], the core concepts of a pattern include *Name*, *Context*, *Problem*, *Forces*, and *Solution*, which should be specified in all patterns. In particular, *Context* describes the situations in which the pattern may apply; the *Problem* presents the problem for which the pattern offers a solution; *Forces* are concerns, often contradictory, which have to be taken into account when determining the applicability of a pattern; *Solution* describes fundamental principles underlying the pattern, which address the problem of the pattern.

Table 3.1: An exemplary security pattern in [Fernandez-Buglioni, 2013]

Name: Abstract Intrusion Detection System
Context: Nodes for local systems that need to communicate with each other using the Internet.
Problem: An attacker may try to infiltrate our system through the Internet. We need to know when an attack is happening and take appropriate response.
Force: • *Communication.* The system is usually more secure if we have a closed network. However, in today's world it is better and more realistic to use the Internet or other insecure network to reduce costs, which may subject our network to security threats. • *Real time behavior.* Attacks should be detected before the attack completes its purpose, so that we can preserve our assets and save time and money. It is difficult to detect an attack when it is happening, but such detection is imperative if we are to react timely and appropriately. • *Incomplete security.* Security measures such as encryption, authentication and so on may not protect all our systems, because they do not cover all possible attacks. • *Non-suspicious users.* Request coming from a non-suspicious address (permitted by a firewall) could still be harmful and should be monitored further. • *Flexibility.* Hard-coding the type of attack can be done easily. But it will be hard and time-consuming to adapt to attack patterns that change constantly.
Solution: Each request to access the network is analyzed to check whether it conforms to the definition of an attack. If we detect an attack, an alert is raised and some countermeasures may be taken.

In this thesis, when integrating security patterns into our framework (Chapter 6), we exclusively focus on analyzing and reusing knowledge related to these core concepts.

Table 3.1 shows the Abstract Intrusion Detection System pattern [Fernandez-Buglioni, 2013], which has been specified with these core concepts. Specifically, this pattern is intended to prevent attacker from infiltrating into the target system (*Problem*), which communicates with other nodes via Internet (*Context*). Additionally, a list of considerations should be taken into account when dealing with this security problem, e.g., detecting attacks when it is happening is imperative for timely reactions, but it is difficult to achieve (*Forces*). After balancing such forces for the problem in this context, one solution can be checking each request to access the network whether it conforms to the definition of an attack (*Solution*).

3.4 CAPEC

Attack patterns document reusable attack knowledge, helping analysts with attack analysis. CAPEC (Common Attack Pattern Enumeration and Classification) is a comprehensive and well-documented attack knowledge repository, which has been incrementally built starting from 2007 and includes 504 attack patterns thus far[1]. In this thesis, we propose to leverage the reusable attack knowledge of CAPEC in order to practically identify possible attacks for STSs. Specifically, we consider the following advantages of CAPEC:

- CAPEC involves a wide range of attack categories, from social engineering attacks, to software attacks, to physical attacks, perfectly meeting our needs for holistic security analysis of STSs.
- CAPEC attack patterns are well-documented in a structural way, covering different aspects of an attack. Such detailed and comprehensive information can help analysts to better understand and apply attack patterns. We will further describe and illustrate the essential parts of an attack pattern in the following subsection.
- Apart from attack patterns, CAPEC also offers useful pattern categories (e.g., mechanisms of attack and domains of attack) that help to navigate through the large number of patterns.

Similar to the security patterns, instead of only referring to the CAPEC knowledge repository, we intend to seamlessly integrate the CAPEC attack patterns into our holistic security analysis framework and promote their practical adoption. As such, in the next subsection, we describe in detail the specific attack knowledge that is concerned by our framework.

3.4.1 CAPEC Schema

Attack patterns, as a specific type of pattern, are specified in the same spirit of design patterns [Gamma et al., 1994], but from an attacker's viewpoint. Thus, an attack pattern also specifies the three primary pattern concepts that are proposed by Alexander et al. [1977], i.e., *Context*, *Problem*, *Solution*. It is worth noting that such pattern concepts are described from an attacker's perspective, i.e., what an attacker wants to attack (*Problem*), how does the attacker perform the attack (*Solution*), under what situation (*Context*). Beyond these concepts, a CAPEC attack pattern also includes a lot of detailed attack-related information, such as *Mitigations*, *Severity*, *Likelihood of Exploit*, etc. A full schema of the CAPEC attack pattern can be found online[2].

---

[1] `https://capec.mitre.org`

[2] `https://capec.mitre.org/data/xsd/apschemav2.7.xsd`

Table 3.2: An exemplary CAPEC attack pattern

Name: SQL Injection (CAPEC-66)
Attack Motivation-Consequences: • *Integrity.* Modify application data • *Confidentiality.* Read application data • *Confidentiality/Integrity/Availability.* Execute unauthorized code or commands • *Confidentiality/Access_Control/Authorization.* Gain privileges / assume identity
Attack Prerequisites: • SQL queries used by the application to store, retrieve or modify data • User-controllable input that is not properly validated by the application as part of SQL queries
Technical Context: • *Architectural Paradigms.* All • *Frameworks.* All • *Platforms.* All • *Languages.* All
Attack Execution Flow: • Survey application • Determine user-controllable input susceptible to injection • Experiment and try to exploit SQL Injection vulnerability

Table 3.2 presents the SQL Injection pattern (CAPEC-66), showing the attack knowledge that is related to the primary pattern concepts. A full specification of this pattern can be found online[3], which also contains other detailed attack knowledge. In particular, this attack pattern can be used to gain privileges to an application (*Problem*) which use SQL queries to operate data (*Context*). To this end, an attacker should first survey the application and then determine user-controllable input susceptible to injection, based on which she can experiment and conduct SQL injection attacks (*Solution*). Note that this pattern can also deal with other *Problems* as specified in the *Attack Motivation-Consequences* section, i.e., modify application data.

In our framework, we mainly focus on identifying and selecting attack patterns. Thus, we exclusively concern the attack knowledge that can be mapped to the three primary pattern concepts. Specifically, we extract the context of an attack from two pattern sections, *Attack Prerequisite* and *Technical Context*; we analyze the problem of an attack based on section *Attack Motivation-Consequences*; and the solution of an attack is described in section *Attack Execution Flow.* In Chapter 5, we will describe a detailed approach to

---

[3] `https://capec.mitre.org/data/definitions/66.html`

process CAPEC patterns and effectively make use of the corresponding attack knowledge to support our analysis.

3.5 Chapter Summary

In this chapter, we describe several existing techniques, based on which we develop our holistic security requirements analysis framework. We also explain the reasons why we choose such techniques, and associate them with specific parts of our framework. In particular, we introduce the Requirements Problem as the theoretical foundation of our holistic requirements framework (Section 3.1). We base our modeling language on two goal-oriented modeling languages, i.e., i* and Techne, the advantages and core concepts of which are introduced in Section 3.2. Moreover, we also introduce a contextual goal modeling language, which is used for modeling attack patterns and security patterns. In Section 3.3, we introduce existing security patterns, which we intend to integrate into our framework in order to practically support analysts with little security knowledge. Especially, we present different templates that have been used by existing security patterns, on top of which we have described and illustrated the essential concepts of security patterns that are used by our approach, including *Context*, *Problem*, *Forces*, and *Solution*. In line with the integration of security pattern, we aim to seamlessly incorporate practical attack knowledge into our framework, helping analysts to identify realistic attacks. Specifically, we choose CAPEC attack patterns as the attack knowledge source because of their wide coverage and detailed documentation, and introduce such patterns in detail in Section 3.4. We also introduce the CAPEC pattern schema and highlight the sections that are used in our framework.

Chapter 4

A Three-Layer Security Requirements Analysis Framework

> *The most fundamental problem in software development is complexity. There is only one basic way of dealing with complexity: divide and conquer.*
>
> ---
>
> Bjarne Stroustrup

In this chapter, we present a holistic security requirements analysis framework which divides an STS into three layers: a social layer (business processes and social actors), a software layer (software applications that support the social layer) and an infrastructure layer (physical and technological infrastructure). Within this framework, each layer focuses on particular concerns and has its own requirements and specifications, which are captured by goal-oriented requirements models. In particular, specifications in one layer dictate requirements in lower layers, and we use cross-layer links to capture such dependencies. As such, the framework not only performs security requirements analysis at each of the three layers, but also takes into account the connections among layers. Eventually, the framework generates holistic security solutions that can satisfy security requirements at all layers, achieving holistic security protection.

Specifically, we first describe the rationale of having such a three-layer structure and how it supports the analysis of the requirements problem in Section 4.1. Next, we propose a goal-oriented modeling language based on existing techniques (i* and Techne) in Section 4.2, which models requirements of STSs based on the proposed three-layer structure. In Section 4.3, we introduce a systematic process which guides holistic security requirements analysis across three layers. The process takes a system's functional requirements and high-level security requirements as inputs, iteratively performs security requirements analysis throughout the three layers in order to generate holistic security solutions. In particular, we have proposed a collection of analysis methods that have been formalized so as to (semi-)automate each analysis step in the process. After that we discuss several aspects of the proposed framework in more detail, such as potential and applicability, in Section 4.4. In the meanwhile, such discussions shed light on the motivation of other parts of the thesis (i.e., Chapter 5-7). In the end, we compare our three-layer framework with related work (Section 4.5).

4.1 A Three-Layer Structure for STSs

When dealing with STSs, compared to traditional software systems, there is a plethora of artifacts that need to be accounted for in order to make the entire system work effectively. In particular, we focus on three important aspects of STSs which have received much attention from the security community, and structure them into three layers. At the most abstract level, we consider a social layer that conceptualized in terms of social actors, social dependencies, and business processes. At the next layer, we consider software applications that support the social layer, conceptualized in terms of architectural components. Finally, we consider an infrastructure layer that focuses on the technological and physical infrastructure that supports deployment of software applications and business processes.

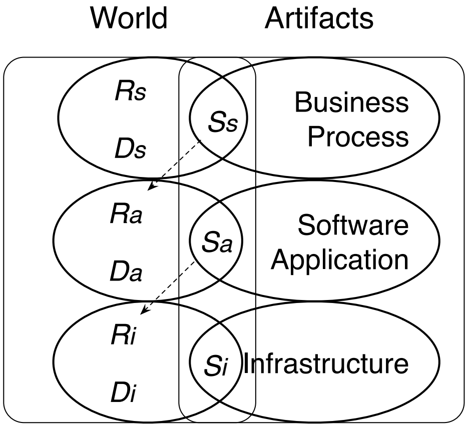


Figure 4.1: The requirements problem extended for STSs

Once structuring STSs into the three layers, we argue that each layer involves specific phenomena, which compose solutions to deal with security requirements of STSs. Thus, each layer has its own requirements, which are satisfied by layer-specific specifications under corresponding domain assumptions. We contend that the original requirements problem, which was described in Fig. 3.1 (Section 3.1), cannot accommodate the requirements analysis of STSs, as it exclusively focuses on how a machine interacts with the world. Therefore, we extend the original requirements problem to *deal with the requirements problem of each layer*, respectively (as shown in Fig. 4.1). In particular, the *Machine* in the original requirements problem has been generalized into *Artifacts*. Each of these *Artifacts* is designed by layer-specific phenomena and satisfies corresponding requirements under specific domain assumptions in the *World*. For example, the social-layer requirements (R_s) are satisfied by business process specifications (S_s) under the social-layer domain assumptions (D_s), i.e., $S_s, D_s \vdash R_s$, and other layers have their own requirements problem, likewise.

Given the three-layer structure, the specifications of one layer determine the requirements of lower layers. In particular, as shown in Fig. 4.1, social-layer specifications (S_s) determine the requirements of software applications (R_a), while application specifications (S_a) affect the requirements of the infrastructure (R_i) . The semantics of these dependencies is that the achievement of the specification in one layer requires the satisfaction of requirements in its lower layer. Such dependencies are captured manually, through which we are able to connect security requirements across layers and analyze them from a holistic viewpoint.

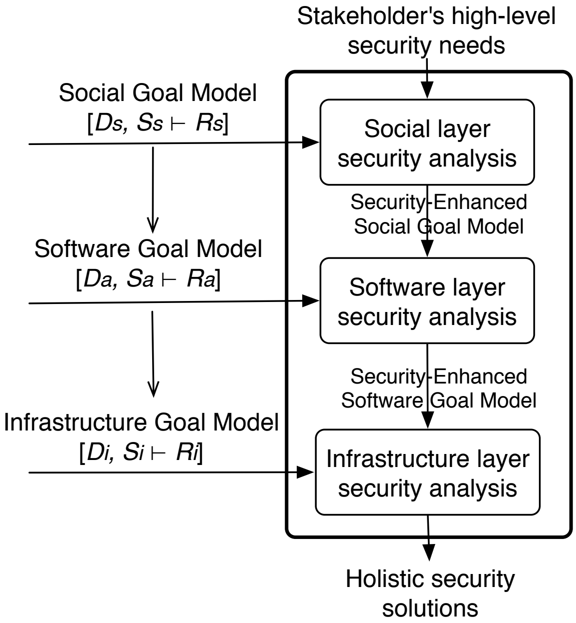

Stakeholder's high-level security needs
Social Goal Model [Ds, Ss ⊢ Rs]
Social layer security analysis
Security-Enhanced Social Goal Model
Software Goal Model [Da, Sa ⊢ Ra]
Software layer security analysis
Security-Enhanced Software Goal Model
Infrastructure Goal Model [Di, Si ⊢ Ri]
Infrastructure layer security analysis
Holistic security solutions

Figure 4.2: An overview of the three-layer security requirements analysis framework

The three-layer structure and the extended requirements problem serve as the basis for holistically dealing with security requirements of STS. As shown in Fig. 4.2, our approach starts with stakeholder security requirements, and analyzes them across layers with regard to layer-specific goals and specifications, resulting in a set of holistic security solutions encompassing security issues in all three layers. We will present details of the entire analysis process in Section 4.3.

4.2 A Three-Layer Requirements Modeling Language

In this section, we propose a three-layer requirements modeling language, which we use to model requirements of STSs in three layers. We first describe the meta-model of this language. In particular, we focus on explaining the extended concepts used in this language. After that, we provide the formal definition of each concept, based on which we define inference rules that support security analysis (we will describe in the next section).

4.2.1 Conceptual Model

We base our modeling language on *i** and *Techne*, and extend them from two dimensions. Firstly, we extend *goals* and *tasks* in order to model requirements of STSs in three layers. Secondly, we define security concepts that can be used to model and analyze security requirements. Fig. 4.3 shows an overview of the meta-model of the proposed language, where the newly introduced concepts are highlighted with dashed rectangles. For the original concepts, we reuse their definitions as presented in [Yu, 1997; Jureta et al., 2010], which have been introduced in Section 3.2. For the new concepts, we describe each of them in detail below.

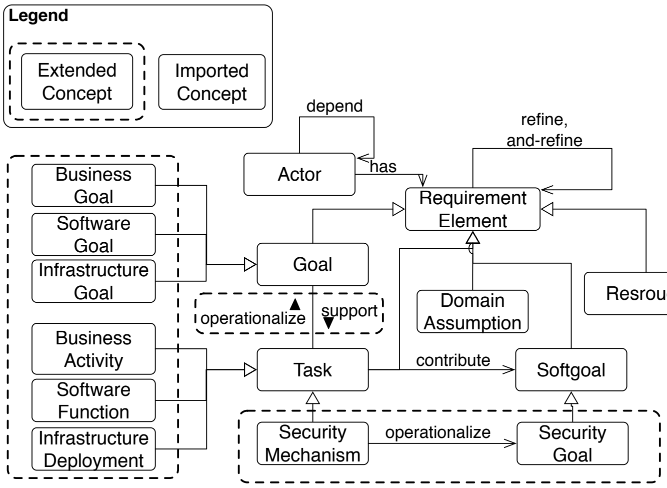


Figure 4.3: Meta-model of the three-layer security requirements framework

Illustrative Example

In a smart grid real-time pricing scenario, the energy supplier collects real-time energy consumption data and balance loads on the power grid. This scenario presents a typical STS. Firstly, it includes a business process for price generation. Specifically, the energy supplier will periodically collect load information about the power grid, and generate appropriate energy prices accordingly in order to regulate the load of the power grid. On the other side, the energy consumers will adjust their energy usage based on the real-time price. Secondly, a number of applications are involved in this scenario to support the interactive process. In particular, a home energy management system is used by the energy consumer to communicate with the energy supplier and control the smart appliances in her apartment. Thirdly, physical devices (e.g., energy management server) are required to deploy the software applications, and networks need to be appropriately configured to support communications.

We here use a part of the scenario as an example to illustrate our three-layer framework, the full details of this scenario will be presented in Section 9.1, where we report the case study performed based on this scenario. In particular, Fig. 4.4 presents the part of scenario we use for illustration, which focuses on *Energy Supplier* and related applications and infrastructure. This figure shows how requirements of STSs are structured and modeled in three layers, and will be used for demonstrating relevant concepts of the three-layer requirements modeling language, as indicated in Fig. 4.3.

Extended Requirement Concepts

As we build goal models for different layers capturing different concerns, we specialize *Goal* into layer-specific goals that focus on a particular aspect of stakeholder needs. Specifically, in the social layer, *Business Goals* represent a stakeholder's high-level requirements for his business. For example, the energy supplier has a business goal about applying real-time pricing strategy to regulate loads on the power grid (i.e., *BG1* in Fig. 4.4). *Software Goals* represent stakeholder requirements in respect of software applications that are used to perform corresponding business activities. For example, the energy control application needs to implement functions to support price calculation (i.e., *AG1* in Fig. 4.4). *Infrastructure Goals* represent stakeholder requirements on technical and physical infrastructure that supports the execution of software applications. For example, the energy supplier server needs to set up a channel to enable communications between the energy control application and the smart meter firmware (i.e., *IG2* in Fig. 4.4).

Accordingly, we assign *Tasks* at different layers with specific operational definitions reflecting the layer. In particular, a task amounts to a *Business Activity* in the social layer,

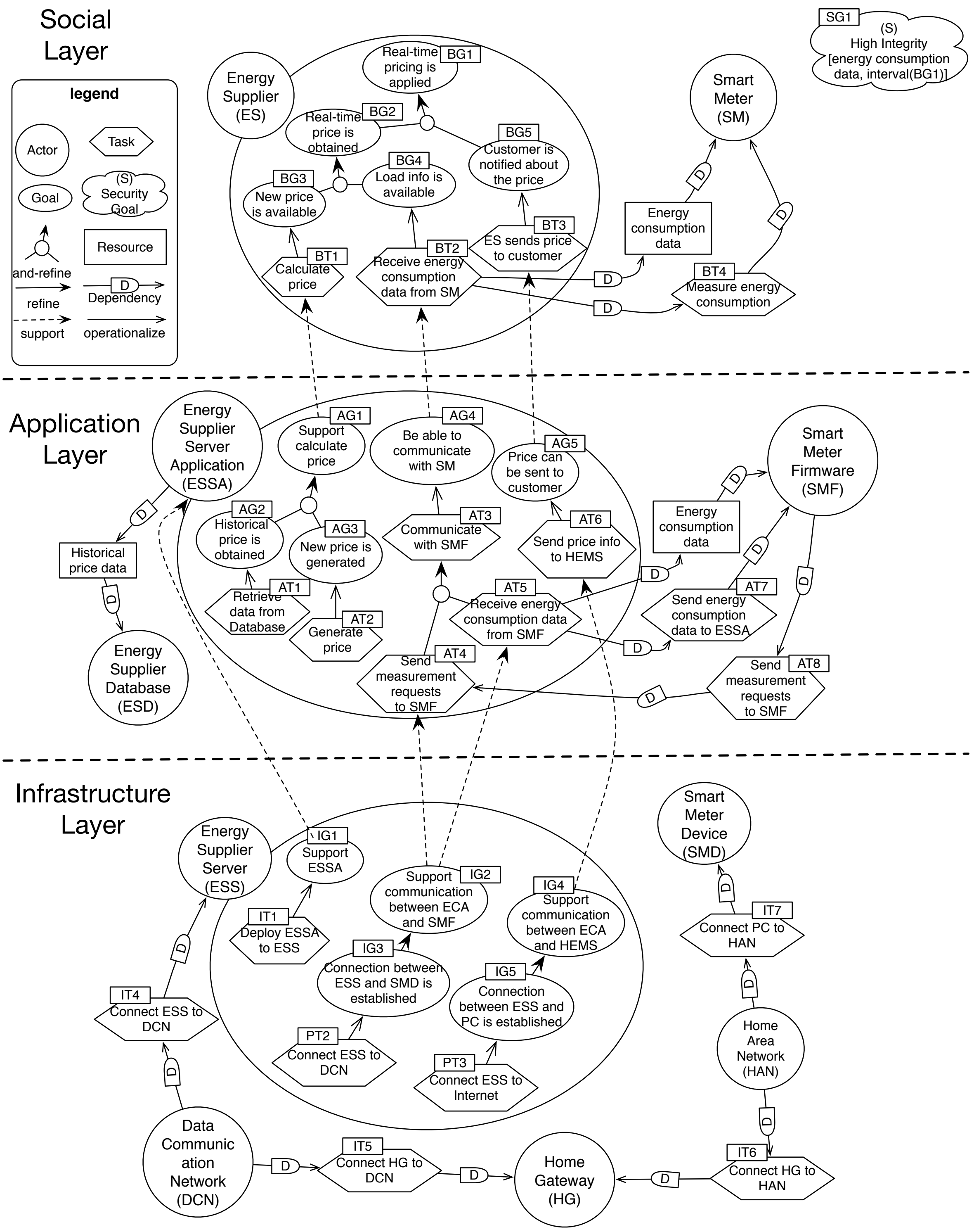

Social Layer
legend
Actor
Task
Goal
(S) Security Goal
and-refine
Resource
refine
D Dependency
support
operationalize
SG1
(S) High Integrity [energy consumption data, interval(BG1)]
Energy Supplier (ES)
BG1
Real-time pricing is applied
BG2
Real-time price is obtained
BG5
Customer is notified about the price
BG3
New price is available
BG4
Load info is available
BT1
Calculate price
BT2
Receive energy consumption data from SM
BT3
ES sends price to customer
Smart Meter (SM)
Energy consumption data
BT4
Measure energy consumption
D
Application Layer
Energy Supplier Server Application (ESSA)
AG1
Support calculate price
AG4
Be able to communicate with SM
AG5
Price can be sent to customer
AG2
Historical price is obtained
AG3
New price is generated
AT3
Communicate with SMF
AT6
Send price info to HEMS
AT1
Retrieve data from Database
AT2
Generate price
AT5
Receive energy consumption data from SMF
AT4
Send measurement requests to SMF
Historical price data
Energy Supplier Database (ESD)
Smart Meter Firmware (SMF)
Energy consumption data
AT7
Send energy consumption data to ESSA
AT8
Send measurement requests to SMF
Infrastructure Layer
Energy Supplier Server (ESS)
IG1
Support ESSA
IG2
Support communication between ECA and SMF
IG4
Support communication between ECA and HEMS
IT1
Deploy ESSA to ESS
IG3
Connection between ESS and SMD is established
IG5
Connection between ESS and PC is established
IT4
Connect ESS to DCN
PT2
Connect ESS to DCN
PT3
Connect ESS to Internet
Smart Meter Device (SMD)
IT7
Connect PC to HAN
Home Area Network (HAN)
Data Communication Network (DCN)
IT5
Connect HG to DCN
Home Gateway (HG)
IT6
Connect HG to HAN
D

Figure 4.4: An excerpt of the three-layer requirements model of the smart grid scenario

while in the software layer it is a *Software Function*, and in the infrastructure layer it is a *Deployment* setting. For example, in the social layer, to satisfy the business goal *BG1*, a collection of business activities should be performed (e.g., *BT1-3*), which are captured as tasks in this layer. Note that when identifying and modeling tasks in the social, analysts should only focus on layer-specific phenomena, while ignore lower-layer concerns. In this example, analysts only consider what business activities are required to achieve the business goal, regardless of whether and how software applications are used within the activities. The benefits of assigning the operational definitions to tasks at different layers are twofold. Firstly, the operational definitions can help analysts to distinguish tasks among layers and determine the granularity of tasks; secondly, tasks with operational definitions can be connected to concepts within additional design models, such as the tasks of the social layer correspond to the business process activities of BPMN (Business Process Modeling Notation). As such, the layer-specific goal models are easy to be transformed into corresponding design models by using available techniques (e.g., [Pimentel et al., 2012; Halleux et al., 2008]), facilitating system development in later lifecycle stages. To simplify the modeling constructs, we use the same notion (ellipse/hexagon) to model goals/tasks in different layers, the layer-specific meaning of these concepts is implied by the layer to which they belong.

Apart from the above concepts, two relations are also included in the proposed framework. *Operationalize* is a relation that relates a goal to a task that operationalizes it within the same layer. For example, in Fig. 4.4, business goal *Customer is notified about the price (BG5)* is operationalized by business activity *ES sends price to customer (BT3)*. *Support* is a cross-layer relation, which indicates that a goal in one layer supports a task in the above layer. Thus, the satisfaction of the task requires the achievement of the goal, while the achievement of the goal cannot imply the satisfaction of the task. Formally speaking, the satisfaction of the goal is a *necessary condition* for satisfying the task, which is defined below:

$$support(G, T) \land satisfied(T) \rightarrow satisfied(G)$$

Take business activity *BT1* as an example, to execute the activity *calculate price*, it requires support from *Energy Supplier Server Application (ESSA)*, where the application goal *AG1* must be satisfied. On the other hand, the satisfaction of *AG1* is not enough to guarantee the successful execution of *BT1*, as it also depends on how *Energy Supplier (ES)* use the application to perform the task, i.e., it can also be affected by specific phenomena in the social layer. The support relation is used likewise between the application layer and the physical layer.

When determining whether a task requires support from the next layer down, analysts should take into account the layer-specific operational definition of the task. In particular, as a task is a business activity in the social layer, analysts should determine whether the execution of the activity involves any software applications. If so, a support link should be added to capture such relation. For example, in Fig. 4.4, as *ES* intends to send price data to the customer using the application *ESSA*, the requirements goal *AG5* is introduced to the application layer which supports the business activity *BT3*, i.e., the satisfaction of the task in the business layer requires the satisfaction of the goal in the application layer. The application will, in turn, refine the goal into more detailed tasks and goals within the application layer.. At some point in the application layer refinement, tasks, which are application functions, will require support from the physical hardware that deploys the software application. As shown in Fig. 4.4, to deal with the modeling scalability problem, instead of modeling the support link for each task in the application layer from the infrastructure layer, we graphically model the support link from the lower-layer goal *IG1* to the application agent *ESSA*. Semantically, that support link indicates that all the tasks of the application *ESSA* are supported by the goal *IG1*. In addition to this default support link, for each application task, within or out of the ESSA actor, analysts also need to identify whether the task requires additional support from the infrastructure layer. For instance, the application task *Receive and store energy consumption data from SMF (AT5)* requires the *Energy Supplier Server* to be connected with SMF, otherwise this function cannot be correctly executed. This is captured in Fig. 4.4 via the supports link from *IG2* to *AT5*.

Extended Security Requirement Concepts.

Security goals have been treated as a specialization of softgoal in several goal-oriented security analysis approaches, such as [Chung, 1993; Oladimeji et al., 2006b; Elahi et al., 2010]. We follow such paradigm and express more detailed and specific security requirements. In particular, we define a security goal as a specialization of softgoal, which represents stakeholder's security needs with regard to specific assets and time intervals.

Since security requirements can interact notoriously with system functional requirements, we denote security goals with a graphical separation from their functional counterparts, as shown in Fig. 4.4. However, security goals are semantically connected with the goals/tasks via an *Interval* attribute (explained below). Each security goal is represented by a template: *<importance><security property>[<asset>, <interval>].* Take security goal *SG1* as an example (Fig. 4.4), *High Integrity [energy consumption data, interval(BG1)]* represents the security requirement "protecting integrity of energy consumption data during the execution interval of *BG1* to a high degree". We describe the four attributes of a security goal in detail as follows:

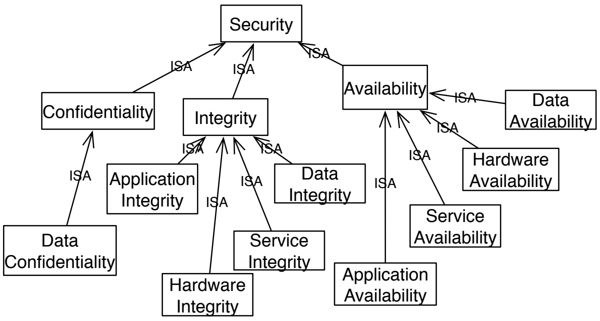


Figure 4.5: Hierarchy of security properties

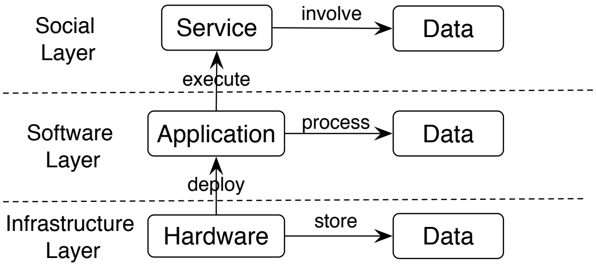


Figure 4.6: Overview of assets

- The *Security Property* specifies a characteristic of security. In this paper, we exclusively focus on confidentiality, integrity, and availability, which are the three main dimensions of information security ISO [2012]. In addition, we detail sub-properties of these three security properties based on the taxonomy defined by Firesmith Firesmith [2004], as shown in Fig. 4.5. For example, under the security property *integrity*, there are four sub-properties *data integrity*, *service integrity*, *application integrity*, and *hardware integrity*. Note that the security properties we consider in our work constitute a starting point and that can be extended in the future.

- The *Asset* is anything that has value to an organization, such as data or services. Fig. 4.6 shows an overview of types of assets accommodated by our framework, as well as the interrelationships among them. In particular, we consider *services* as assets at the social layer; *applications* that execute services are considered as assets in the software layer; in the infrastructure layer, we analyze *hardware* as an asset which deploys applications. Moreover, *data* is an asset that is considered in all three layers.

- The *Interval* of a security goal indicates the time period when the security goal applies. Haley et al. have pointed out that "*Threats can have a 'time' element, stating that the harm will occur only if the violation occurs before or after some point, or within some interval*" Haley et al. [2004]. We agree that the time dimension does affect system security requirements analysis. In particular, we specify an interval in terms of the execution period of a task, i.e., *interval(task)*. Note that a goal can also be used to represent an interval, which is the execution period of all tasks that operationalize this goal. In this way, the security goals are implicitly connected with the system functional requirements.

- The *Importance* of a security goal indicates the priority of a security goal. Possible values include {*very low, low, medium, high, very high*}.

A *Security Mechanism* is a method that operationalizes a security goal. We define the security mechanism as a specialization of task. The operationalization relation between security mechanisms and security goals is a *many-to-many* relation, i.e., one security mechanisms can satisfy multiple security goals, and one security goal may require more than one security mechanisms. In our framework, security mechanisms are also applied in different layers, and become part of the specification of the corresponding layer. For example, *Auditing* is a security mechanism which can be applied in the social layer to ensure the integrity of a business activity, while *Input Guard* is a software-specific security mechanism, which checks all inputs of a software application. It is worth noting that we import the layer-specific security mechanisms from existing security patterns, which will be presented in the next section.

4.2.2 Formal Definitions

Based on the above explanation of concepts of our modeling language, we present their formal definitions. In particular, the formal predicates we defined here will be used for specifying inference rules which (semi-)automate security requirements analysis (in Section 4.3).

Concepts

- **Definition 1**. An *Actor* denotes an entity of an STS, which has goals and should perform tasks to satisfy those goals. In particular, an actor can be a social agent (human), a software agent (application), or a physical agent (hardware), formally, $Actor = Human \cup Application \cup Hardware$. We define $A = \{a_1, ..., a_n\}$ is a set of $Actors$. If $a \in A$, we write $actor(a)$.

- **Definition 2**. A *Requirement* is an abstract concept which represents stakeholder's needs, including goals, softgoals, tasks, resources, and domain assumptions. Formally, $Requirement = Goal \cup Softgoal \cup Task \cup Resource \cup DomainAssumption$. We define $RE = \{re_1, ..., re_n\}$ is a set of $Requirements$. If $re \in RE$, we write $req(re)$.

- **Definition 3**. A *Goal* represent an actor's desire, which has clear-cut criteria for its satisfaction. For our three-layer requirements language, a goal can be a business goal, a software goal, or a infrastructure goal, formally, $Goal = BusinessGoal \cup SoftwareGoal \cup InfrastructureGoal$. We define $G = \{g_1, ..., g_n\}$ is a set of $Goals$. If $g \in G$, we write $goal(g)$.

- **Definition 4**. A $Softgoal$ is an actor's desire, which does not have clear-cut criteria for its satisfaction. Usually, a softgoal is used to capture stakeholder's quality requirements. We define $S = \{s_1, ..., s_n\}$ is a set of $Softgoals$. If $s \in S$, we write $softgoal(s)$.

- **Definition 5**. A $Task$ represents an intention to perform actions, by accomplishing which an actor can achieve her goals. For our three-layer requirements language, a task can be an intention to perform business activities, software functions, or infrastructure deployments, formally, $Task = \mathit{BusinessActivity} \cup \mathit{SoftwareFunction} \cup \mathit{InfrastructureDeployment}$. We define $T = \{t_1, ..., t_n\}$ is a set of $Tasks$. If $t \in T$, we write $task(t)$.

- **Definition 6**. A $\mathit{Resource}$ is a physical or informational entity that an actor requires in order to perform a task. We define $R = \{r_1, ..., r_n\}$ is a set of $\mathit{Resources}$. If $r \in R$, we write $\mathit{resource}(r)$.

- **Definition 7**. A $\mathit{DomainAssumption}$ is an indicative statement that describes phenomena that are considered to be unchanged during system development, within which tasks are performed to achieve the goals. We define $DA = \{da_1, ..., da_n\}$ is a set of $\mathit{DomainAssumptions}$. If $da \in DA$, we write $\mathit{d\_assumption}(da)$.

- **Definition 8**. A $\mathit{SecurityGoal}$ is a specialization of softgoal that focuses on security concerns, i.e., $\mathit{SecurityGoal} \subset \mathit{Softgoal}$. We define $SG = \{sg_1, ..., sg_n\}$ is a set of $\mathit{SecurityGoals}$. If $sg \in SG$, we write $\mathit{sec\_goal}(sg)$. Specifically, we define a security goal as a four-tuple, which expresses stakeholder's security needs with regard to an asset and time interval. $\forall sg \in SG, sg = (imp, sp, as, int), where\ imp \in Importance, sp \in SecurityProperty, as \in Asset, int \in Interval.$

- **Definition 9**. A $\mathit{SecurityMechanism}$ is a specialization of task which operationalizes a security goal, i.e., $\mathit{SecurityMechanism} \subset Task$. We define $SM = \{sm_1, ..., sm_n\}$ is a set of $\mathit{SecurityMechanisms}$. If $sm \in SM$, we write $\mathit{sec\_mechanism}(sm)$.

Relations

- **Definition 10**. We define $REF \subset RE \times RE$ is a set of $refinement$ relations. If $re_1 \in RE, re_2 \in RE$, such that re_1 can be refined into re_2, we write $refine(re_2, re_1) \in REF$. This relation indicates that requirement re_1 can be satisfied as long as requirement re_2 is satisfied.

- **Definition 11**. We define $AND\_REF \subset RE \times RE$ is a set of *and-refinement* relations. If $re_1 \in RE, \ldots, re_n \in RE$, such that re_1 can be and-refined into

$\{re_2, \ldots, re_n\}$, we write $and\_refine(re_2, re_1), \ldots, and\_refine(re_n, re_1) \in AND\_REF$. Such relations indicate that requirement re_1 can only be satisfied when all its and-refinements (i.e., requirement $re_2, \ldots, re_n$) are satisfied.

- **Definition 12**. We define $OPE \subset T \times G$ is a set of *operationalization* relations, which represent how goals can be achieved by tasks. If $t \in T, g \in G$, such that g can be operationalized as t, we write $operationalize(t, g) \in OPE$. Based on this relation, goal g can be satisfied as long as task t is satisfied.

- **Definition 13**. We define $SUP \subset G \times T$ is a set of *support* relations, which capture interactions between tasks in one layer and goals in the next layer down. If $g \in G, t \in T$, such that t is supported by g, we write $support(t, g) \in SUP$. Based on this relation, the satisfaction of g is a necessity condition for the satisfaction of task t, i.e., $satisfied(t) \rightarrow satisfied(g)$

- **Definition 14**. We define $DEL \subset A \times R \times R \times A \times R$ is a set of *dependency* relations, which represent interactions among actors. If $a_1 \in A, re_1 \in RE, re \in RE, a_2 \in A, re_2 \in RE$, such that actor a_1's requirement re_1 depends on actor a_2's requirement re_2 for requirement re, we write $depend(a_1, re_1, re, a_2, re_2)$. In this case, a_1 is a depender who depends for something to be provided; re_1 is the requirement of the depender, which explains why the dependency exists; re is the dependum, which is the object of the dependency; a_2 is the dependee who should provide the dependum; re_2 is the requirement of the dependee, which explains how the dependee intends to provide the dependum. It is worth noting that both the depender's requirement and the dependee's requirement can be omitted in a dependency relation, which is in line with the strategic dependency diagram of i*.

- **Definition 15**. We define $CON \subset RE \times Value \times S$ is a set of *contribution* relations, which represent the influences of requirement elements on softgoals. If $re \in RE, v \in Value, s \in S$, such that re contributes to the satisfaction of s to the extent of v, we write $contribute(re, v, s) \in CON$. In particular, we consider four types of contributions links, each of which represents a particular extent of contributions, $Value = \{Make, Help, Hurt, Break\}$.

- **Definition 16**. We define $HAS \subset A \times RE$ is a set of *has* relations, which associate an actor with her requirements. If $a \in A, re \in RE$, such that actor a requires requirement re, we write $has(a, re) \in HAS$.

Table 4.1 summaries the predicates of all the concepts and relations we have defined above. Such predicates are used for specifying formal inference rules which support our security requirements analysis. More details will be presented in the next section.

Table 4.1: Formal predicates of the three-layer requirements modeling language

Concepts
$actor(Actor : a)$
$req(Requirement : re)$
$goal(Goal : g)$
$softgoal(Softgoal : s)$
$task(Task : t)$
$resource(Resource : r)$
$d\_assumption(DomainAssumption : da)$
$sec\_goal(SecurityGoal : sg)$
$sec\_mechanism(SecurityMechanism : sm)$
Relations
$refine(Requirement : re_2, Requirement : re_2)$
$and\_refine(Requirement : re_2, Requirement : re_2)$
$operationalize(Task : t, Goal : g)$
$support(Goal : g, Task : t)$
$depend(Actor : a_1, Requirement : re_1, Requirement : re, Actor : a_2, Requirement : re_2)$
$contribute(Requirement : re, Value : v, Softgoal : s)$
$has(Actor : a, Requirement : re)$

Apart from the above three-layer requirements constructs, a number of related concepts and relations will also be used in defining inference rules, which are shown in Table 4.2. Specifically, the concepts *data*, *service*, *application*, and *hardware* represent four types of assets, as shown in Fig. 4.6. The concept *threat* specifies a threat to systems, which is used for determining critical security goals. We will describe how to discover threats in detail in Chapter 5.

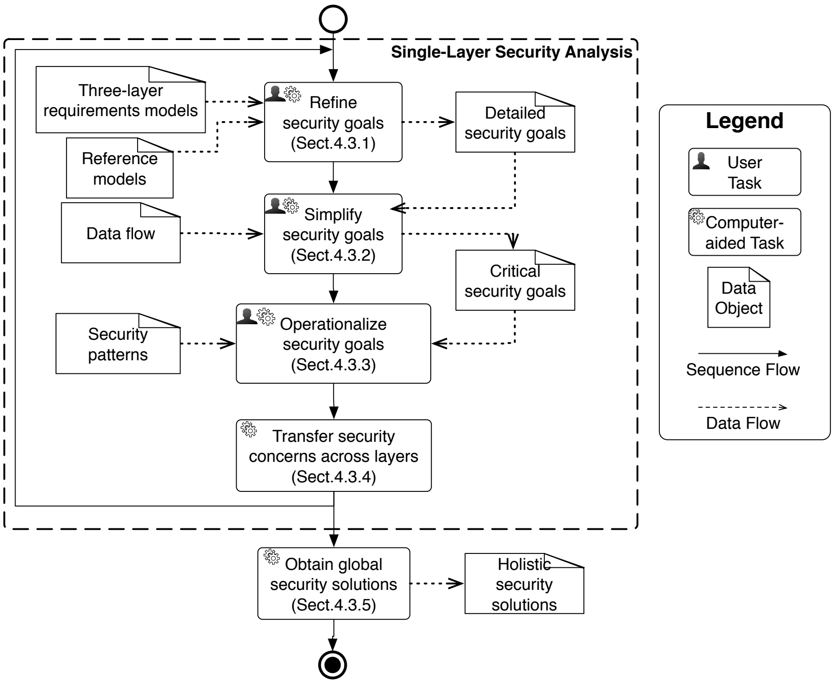
For the relations in Fig. 4.6, *sg_attributes* shows all the detailed attributes of a security goal, such as importance, concerned security properties, etc. The predicate *interval_of* associates a task with corresponding time interval during which the task is performed. The predicate *is_a* describes a specialization relation between two entities. The predicate *part_of* describes the part-of relation between two assets. The predicates *has_input* and *has_output* present the data flow information of a task. The unary predicates *is_applicable* and *is_critical* are assertions on security goals. Finally, *th_attributes* details the attributes of a threat, while *threaten* presents a security goals is threatened by a threat.

Table 4.2: Formal predicates for related constructs

Concepts
$data(Data : d)$
$service(Service : se)$
$application(Application : app)$
$hardware(Hardware : hd)$
$threat(Threat : TH)$
Relations
$sg\_attributes(SecurityGoal : sg, Importance : imp, SecurityProperty : sp, Asset : as, Interval : int)$
$interval\_of(Requirement : re)$
$is\_a(Entity : e_1, Entity : e_2)$
$part\_of(Asset : as_1, Asset : as_2)$
$has\_input(Task : t, Data : d)$
$has\_output(Task : t, Data : d)$
$is\_applicable(SecurityGoal : sg)$
$is\_critical(SecurityGoal : sg)$
$th\_attributes(Threat : th, ThreatType : ty, Asset : as, Interval : int)$
$threaten(Threat : th, SecurityGoal : sg)$

4.3 A Holistic Security Requirements Analysis Process

In this section, we propose a systematic process and a set of security requirements analysis methods to guide security analysis both within one and across layers. Fig. 4.7 shows an overview of the analysis process, which starts from security requirements analysis at the social layer and follows to analyze security requirements at the software layer and the infrastructure layer, respectively. Specifically, within each layer, we refine and concretize security goals to identify possible operationalizations in terms of security mechanisms. Cross-layer analysis aims to propagate the security concerns from one layer to the next layer down. After security analysis has been performed for all layers, we will obtain a holistic security goal model, based on which we can obtain a collection of textual alternative security solutions that are the final output of our approach. A set of inference rules is defined to (semi-)automates the security analysis, which has been implemented in Disjunctive Datalog [Eiter et al., 1997]. All such rules can be automatically inferred by our prototype tool, which will be introduced in Chapter 8. In the remainder of this section, we will go through each analysis step and explain corresponding security analysis methods we have proposed.



Single-Layer Security Analysis
Three-layer requirements models
Reference models
Data flow
Security patterns
Refine security goals (Sect.4.3.1)
Detailed security goals
Simplify security goals (Sect.4.3.2)
Critical security goals
Operationalize security goals (Sect.4.3.3)
Transfer security concerns across layers (Sect.4.3.4)
Obtain global security solutions (Sect.4.3.5)
Holistic security solutions
Legend
User Task
Computer-aided Task
Data Object
Sequence Flow
Data Flow

Figure 4.7: An overview of the three-layer security requirements analysis process

4.3.1 Security Goal Refinement

Having stakeholder's initial security needs as input, analysts should iteratively refine them into more concrete counterparts in order to capture more precise needs of stakeholders and eventually operationalize such needs. Our framework supports three refinement methods, which operate on the three attributes of security goals, respectively. The refinement methods have been formalized as inference rules to better support analysis, as shown in Table 4.3.

Fig. 4.8 presents a series of examples of security goal refinements based on security goal *High Integrity [energy consumption data, interval(BG1)]* (Fig. 4.4), involving all three refinement methods. The type of each refinement is explicitly annotated in the figure, and the reference models that are used for corresponding refinements are presented in the left part of the figure. Using this example, we explain the three refinements methods and their corresponding inference rules (Table 4.3) in detail below.

It is worth noting that Table 4.3 specifies the formal semantics of the proposed inference rules, but does not show the exact implementation in Disjuctive Datalog. This is because Disjuctive Datalog has a couple of restrictions on the representation of rules, and thus not all the proposed inference rules can be directly implemented. For example, it does not allow disjunction expressions in the body part of a rule. Instead, we create an individual rule for each of those disjunctive expressions. Therefore, showing all such specific rules may introduce extra difficulties for understanding the overall semantics of our rules.

Table 4.3: Inference rules for refinement methods

No.	**Content**
REF.P	$\#create(sec\_goal(SG2)) \land sg\_attributes(SG2, IMP, SP2, AS, INT) \land and\_refine(SG2, SG1)$ $\leftarrow is\_a(SP2, SP1) \land sg\_attributes(SG1, IMP, SP1, AS, INT)$
REF.A	$\#create(sec\_goal(SG2)) \land sg\_attributes(SG2, IMP, SP, AS2, INT) \land and\_refine(SG2, SG1)$ $\leftarrow part\_of(AS2, AS1) \land sg\_attributes(SG1, IMP, SP, AS1, INT)$
REF.I.1	$\#create(sec\_goal(SG2)) \land sg\_attributes(SG2, IMP, SP, AS, INT2) \land and\_refine(SG2, SG1)$ $\leftarrow interval\_of(INT1, G1) \land interval\_of(INT2, G2)$ $\land and\_refine(G2, G1) \land sg\_attributes(SG1, IMP, SP, AS, INT1)$
REF.I.2	$\#create(sec\_goal(SG2)) \land sg\_attributes(SG2, IMP, SP, AS, INT2) \land refine(SG2, SG1)$ $\leftarrow interval\_of(INT1, G1) \land interval\_of(INT2, G2)$ $\land refine(G2, G1) \land sg\_attributes(SG1, IMP, SP, AS, INT1)$
REF.I.3	$\#create(sec\_goal(SG2)) \land sg\_attributes(SG2, IMP, SP, AS, INT2) \land refine(SG2, SG1)$ $\leftarrow interval\_of(INT1, G) \land interval\_of(INT2, T)$ $\land operationalize(T, G) \land sg\_attributes(SG1, IMP, SP, AS, INT1)$

Security property-based refinement. Refining security goals via security properties helps the security analysis to cover all possible aspects of security. The rule *REF.P* means: if the security property *SP2* is a specialization of the security property *SP1* (according to the hierarchy of security properties shown in Fig. 4.5), then the security goal *SG1* that concerns the security property *SP1* will be and-refined into a sub security goal *SG2*, which concerns *SP2* and inherits all other attributes from *SG1*. For example, in Fig. 4.8, security goal *SG1* that concerns security property *Integrity* is refined into four sub-security goals by applying rule *REF.P*. In particular, in accordance with the reference model Fig. 4.8-a, the derived security goal *SG2, SG3, SG4, SG5* concern security properties *Application Integrity*, *Data Integrity*, *Service Integrity*, *Hardware Integrity*, respectively.

Asset-based refinement. Refinements of security goals can also be done by refining assets according to the corresponding *part-of* relations. Thus, a security goal can be and-refined to sub-security goals, each of which concerns part of the asset of the original security goal and remains other attributes unchanged (i.e., rule *REF.A*). Note that the *part-of* here stands for an abstract relation, which can apply to various objects, such as data schema, software architecture etc. As shown in Fig. 4.8, security goal *SG3* that concerns asset *Energy consumption data* is refined by applying the rule *REF.A* according to the *part-of* relations (shown in the reference model Fig. 4.8-b). Thus, the derived security goals *SG6, SG7* concern assets *Water consumption data* and *Electricity consumption data*, respectively.

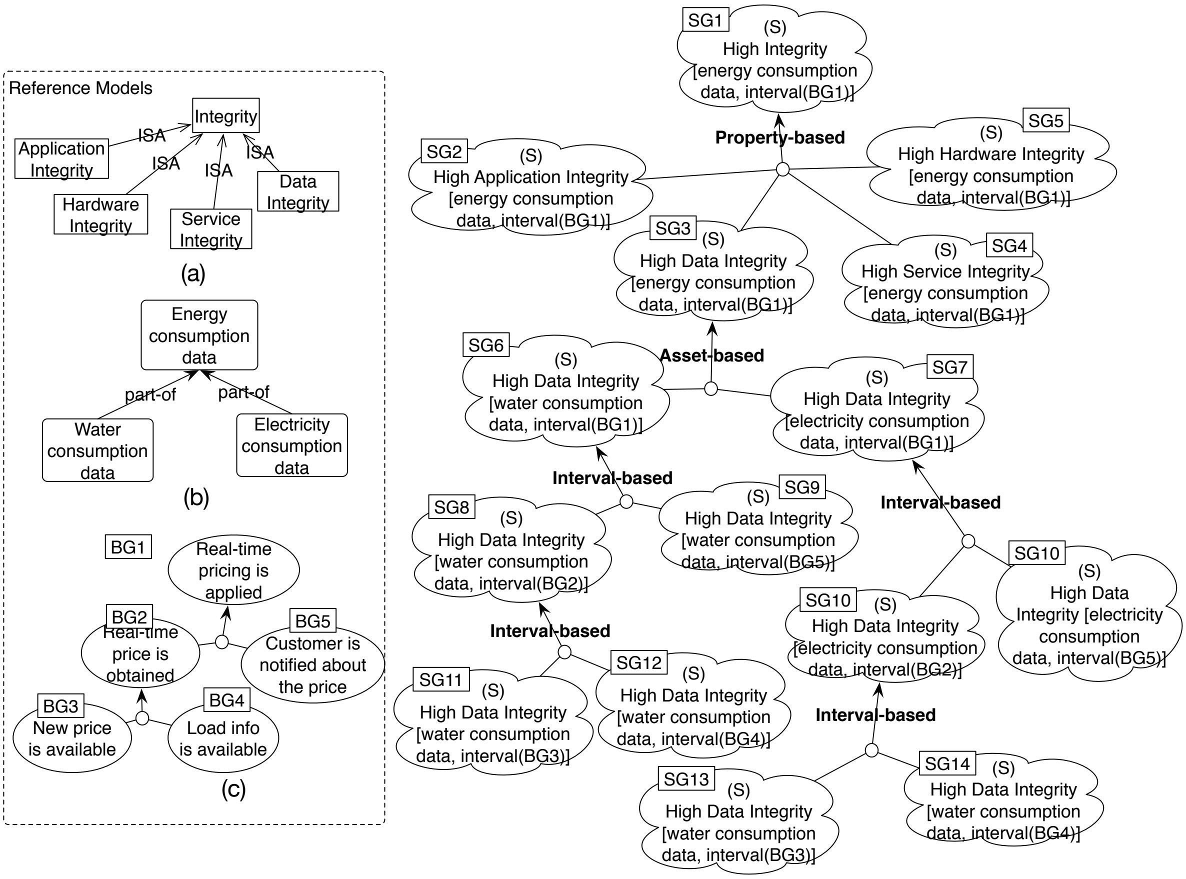

Reference Models
Integrity
Application Integrity
ISA
ISA
ISA
ISA
Hardware Integrity
Service Integrity
Data Integrity
(a)
Energy consumption data
part-of
part-of
Water consumption data
Electricity consumption data
(b)
BG1
Real-time pricing is applied
BG2
Real-time price is obtained
BG5
Customer is notified about the price
BG3
New price is available
BG4
Load info is available
(c)
SG1
(S) High Integrity [energy consumption data, interval(BG1)]
Property-based
SG2
(S) High Application Integrity [energy consumption data, interval(BG1)]
SG5
(S) High Hardware Integrity [energy consumption data, interval(BG1)]
SG3
(S) High Data Integrity [energy consumption data, interval(BG1)]
SG4
(S) High Service Integrity [energy consumption data, interval(BG1)]
Asset-based
SG6
(S) High Data Integrity [water consumption data, interval(BG1)]
SG7
(S) High Data Integrity [electricity consumption data, interval(BG1)]
Interval-based
SG8
(S) High Data Integrity [water consumption data, interval(BG2)]
SG9
(S) High Data Integrity [water consumption data, interval(BG5)]
Interval-based
SG10
(S) High Data Integrity [electricity consumption data, interval(BG5)]
SG10
(S) High Data Integrity [electricity consumption data, interval(BG2)]
Interval-based
SG11
(S) High Data Integrity [water consumption data, interval(BG3)]
SG12
(S) High Data Integrity [water consumption data, interval(BG4)]
Interval-based
SG13
(S) High Data Integrity [water consumption data, interval(BG3)]
SG14
(S) High Data Integrity [water consumption data, interval(BG4)]

Figure 4.8: Security goals refinements

Interval-based refinement. Since an interval specifies the temporal period for which a security goal is concerned, a security analyst can put more detailed constraints on a particular time interval by refining a long interval into short ones. As we use the execution period of requirement goals/tasks to specify intervals, the interval-based security goal refinements are carried out based on the structure of the system requirements model. Specifically, the rules *REF.I.1* and *REF.I.2* say if interval *INT1* of security goal *SG1* can be (and-)refined into *INT2*, then we can obtain a new security goal *SG2* which (and-)refines *SG1* via its interval. Rule *REF.I.3* has similar meaning but focuses on the

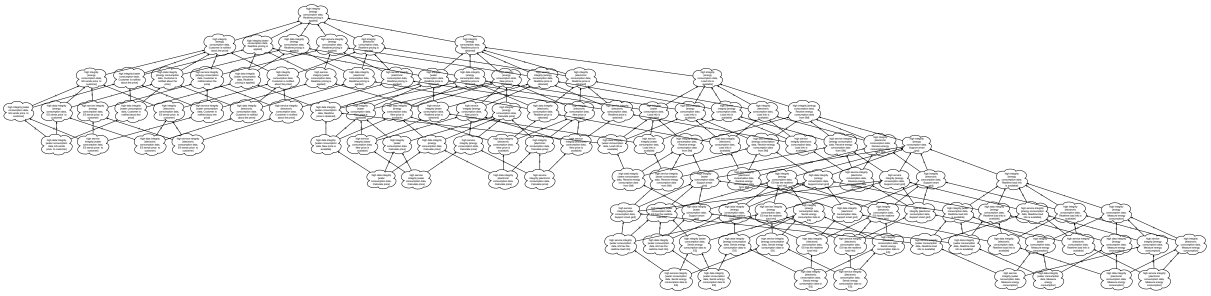


Figure 4.9: Exhaustive refinements for the root security goal

operationalization relation. As shown in Fig. 4.8, the security goals *SG6* and *SG7* have been and-refined via the interval attribute according to the reference model (Fig. 4.8-c).

Refinement strategies. Note that a refinement process can be flexible in the sense that different refinement methods can be applied in any sequence and to any extent. In particular, there are two strategies for refining security goals. Firstly, a step-by-step refinement strategy. After each time of security goal refinement, analysts should check with stakeholders whether all the refined security goals are needed. If a refined security goal is not required by stakeholders, analysts must exclude it from subsequent refinement. In such a way, analysts are able to capture stakeholder's precise security needs which normally cannot be easily expressed during the initial stages of security analysis. Pruning uninteresting goals allows analysts to reduce the refinement space. With the support of our prototype tool (Chapter 8), once analysts choose a refinement dimension for a selected security goal, the tool can automatically perform the refinement analysis and graphically create sub-security goals. For example, the security goal model that is shown on the right side of Fig. 4.8 is derived from a step-by-step refinement analysis, representing one possible way to refine the root security goal *SG1*.

Secondly, an exhaustive refinement strategy, which is intended to explore all the possible refinements of one security goal. Such refinement can be automated by using the prototype tool. For example, considering the same root security goal *SG1* and corresponding reference models in Fig. 4.8, we have created an exhaustive refinement model to show all the possible refinement paths, which contains 117 security goals and 264 refinement links in total (Fig. 4.9). This strategy contributes to the completeness of analysis, but the exhaustive refinements will result in many redundant security goals and complicate the subsequent analysis. Some form of pruning is necessary to manage such complexity.

Considering the advantages and disadvantages of these two strategies, we propose to adopt a hybrid strategy to refine security goals. In particular, analysts should first apply the step-by-step strategy to identify stakeholder's exact security needs, and then perform exhaustive refinement to explore all security goal refinements.

4.3.2 Security Goal Simplification

To release analysts from scrutinizing all the detailed security goals, we define simplification methods to identify critical security goals that need to be further analyzed, allowing us to exclude others. In particular, we perform applicability analysis and threat analysis to determine the criticality of security goals, which will be introduced in detail, respectively. Based on the results of the two types of analysis, we then present how to determine the criticality of a security goal, finishing the simplification analysis.

Applicability analysis. By saying that a security goal is applicable, we mean it is sensible with regard to the meaning of its attributes (security property, asset etc.), otherwise it is inapplicable. To determine the applicability of security goals, firstly, we consider whether the security property is applicable to the type of asset. For example, if a security goal aims to protect the *Application Integrity* of a *Data* asset, then it is inapplicable. Secondly, we check the data-related security property (e.g, *Data Integrity*) based on the involvement of the data during the target interval. For example, if a security goal concerns the *Data Integrity* of a data asset during a specific time interval, but the data is actually not involved in that time interval, then the security goal is inapplicable. To accommodate this analysis, we specify data flow information for each task, i.e., the input and output of each task.

On the basis of the above rationales, we have proposed five inference rules to facilitate the applicability analysis, which are shown in Table 4.4. Take rule *APP.1* as an example: given a security goal *SG*, which considers data-related security properties for an asset *AS* during the execution interval of a task *T*, if the asset is a data asset and it is an input/output of the task *T*, then this security goal is determined as applicable. According to these rules, in Fig. 4.8, security goal *High Data Integrity [water consumption data, new price is available (BG3)]* is inapplicable, because the asset *water consumption data* is not involved in the execution period of *BG3*.

Threat analysis. For each applicable security goal, we identify threats that impair the satisfaction of the goal, which helps us to determine its criticality. To this end, we need to either incorporate existing threat analysis approaches (e.g., [Sindre and Opdahl, 2005; Asnar et al., 2011b]), or import threat knowledge about the target system from existing reports, if available. Regarding this need, in our subsequent research, we have proposed a systematic security attack analysis approach which can holistically identify attacks on

Table 4.4: Inference rules of applicability analysis

No.	**Content**
APP.1	$is\_applicable(SG) \leftarrow (sg\_attributes(SG, \_, data\_confidentiality, AS, INT)$ $\vee sg\_attributes(SG, \_, data\_integrity, AS, INT)$ $\vee sg\_attributes(SG, \_, data\_availability, AS, INT))$ $\wedge interval\_of(INT, T) \wedge data(AS)$ $\wedge(has\_input(T, AS) \vee has\_output(T, AS))$
APP.2	$is\_applicable(SG) \leftarrow service(AS) \wedge (sg\_attributes(SG, \_, service\_integrity, AS, \_)$ $\vee sg\_attributes(SG, \_, service\_availability, AS, \_))$
APP.3	$is\_applicable(SG) \leftarrow application(AS) \wedge (sg\_attributes(SG, \_, application\_integrity, AS, \_)$ $\vee sg\_attributes(SG, \_, application\_availability, AS, \_))$
APP.4	$is\_applicable(SG) \leftarrow hardware(AS) \wedge (sg\_attributes(SG, \_, hardware\_integrity, AS, \_)$ $\vee sg\_attributes(SG, \_, hardware\_availability, AS, \_))$

STSs (detailed in Chapter 5). We will explain the rationale of having this holistic attack analysis approach in Section 4.4.

Each identified threat is specified with name, type, threatened asset, and threatened interval, based on which we can automatically identify security goals that are threatened by the threat. Note that we specify the threat type using the STRIDE threat categories, as each of these categories is matched to a particular type of security property [Hernan et al., 2006], e.g., the threat type *Tampering* is mapped to the security property *Integrity.* We have defined a series of rules $TH.1$-3 to automate such threat analysis. For example, as specified in $TH.1$, if a security goal SG concerns a security property SP which is a type of $confidentiality$, and there is a threat TH that belongs to the type of $information\_disclosure$ and targets the same asset AS and interval INT of SG, then we identify that TH threatens SG.

Simplification analysis. The applicability analysis and the threat analysis can be automated by our prototype tool. Based on the analysis results, we determine whether or not a security goal is critical: 1) if a security goal is applicable and is threatened by certain threats, then it is a critical security goal; 2) if a security goal is applicable but has not been associated with any threats, then the analyst needs to manually determine the criticality of the security goal. Note that in the second case, for analysts with little security knowledge, a conservative solution is to treat all applicable security goals as critical, which ensure the completeness of the analysis but will increase the complexity of

Table 4.5: Inference rules of threat analysis

No.	Content
$TH.1$	$threaten(TH, SG) \leftarrow sg\_attributes(SG, \_, SP, AS, INT) \wedge is\_a(SP, confidentiality)$ $\wedge th\_attributes(TH, information\_disclosure, AS, INT)$
$TH.2$	$threaten(TH, SG) \leftarrow sg\_attributes(SG, \_, SP, AS, INT) \wedge is\_a(SP, integrity)$ $\wedge th\_attributes(TH, tampering, AS, INT)$
$TH.3$	$threaten(TH, SG) \leftarrow sg\_attributes(SG, \_, SP, AS, INT) \wedge is\_a(SP, availability)$ $\wedge th\_attributes(TH, denial\_of\_service, AS, INT)$

subsequent analysis.

As with two strategies for security goal refinements, the simplification analysis can also be applied in two ways. Firstly, analysts can apply the simplification analysis together with the step-by-step security goal refinements, which helps to determine whether a security goal needs to be further refined. In particular, if a refined security goal is not applicable, then it will be excluded from subsequent refinements. Moreover, if a refined security goal is identified as critical, then it does not need to be further refined and will be analyzed for operationalization in the next step of analysis. For a refined security goal which is applicable but not critical, analysts can keep refining it to further evaluate its refinements; if this goal cannot be refined anymore, then analysts have to manually determine whether it is critical.

Secondly, analysts can apply such analysis to all the exhaustively refined security goals with the help of the prototype tool. For example, by applying the simplification analysis to the the aforementioned exhaustively refined security goal model, which contains 117 security goals and 264 refinement links, we identify two critical security goals (i.e., both applicable and threatened) and seven applicable security goals (that have not been threatened). Although this analysis helps to quickly identify critical security goals, the analysts need to manually evaluate all other applicable security goals in order to determine their criticality, which is non-trivial task. Note that since we have proposed to adopt a hybrid refinement strategy, the simplification analysis should be performed accordingly.

4.3.3 Security Goal Operationalization

For each identified critical security goal, we propose operationalization methods to generate possible security mechanisms that can satisfy the critical security goal. In particular, we leverage existing security patterns to help analysts with few security knowledge to operationalize security goals. The security patterns are taken from existing pattern

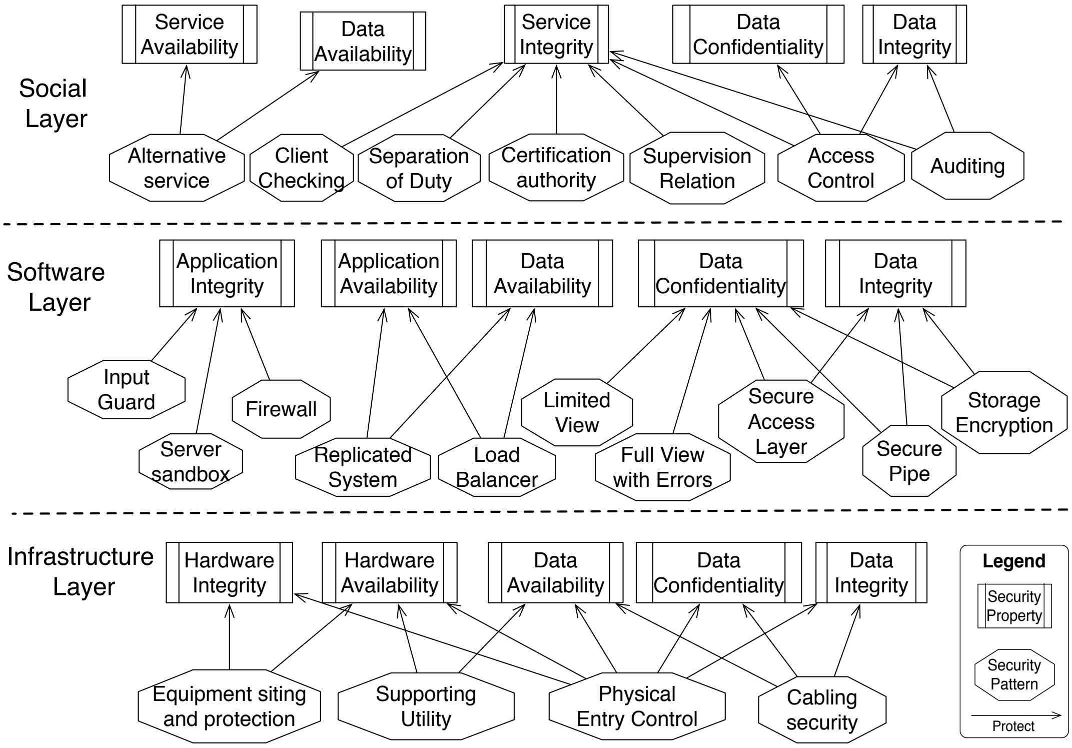


Figure 4.10: Selected security patterns in three layers

repositories [Asnar et al., 2011a; Scandariato et al., 2008; Fernandez et al., 2007b], each of which fits a particular layer in our framework. Fig. 4.10 shows the selected security patterns we used in three layers, as well as the corresponding security properties achieved by those patterns. As indicated by Fig. 4.10, one pattern can be applied to multiple security goals and one security goal can have multiple patterns. In total, we include 21 security patterns in our framework thus far, where 7 patterns are at the social layer, 10 patterns are at the software layer, and 4 patterns are at the infrastructure layer. Note that the selection of security patterns is not intended to be exhaustive, and can evolve and expand over time.

To operationalize a security goal, we first identify all candidate security patterns based on the protected security property of the pattern. For example, as shown in Fig. 4.11, two critical security goals *SG2* and *SG3* concern security property *data integrity*, according to which two candidate security patterns are identified for each security goal (reference to Fig. 4.10). This analysis can be automated by our prototype tool. Having the candidate security patterns, analysts then need to manually check the applicability of such patterns. In particular, the analysts should check the target system and environment against the *context* and *forces* of the candidate patterns, and then determine which pattern to apply.

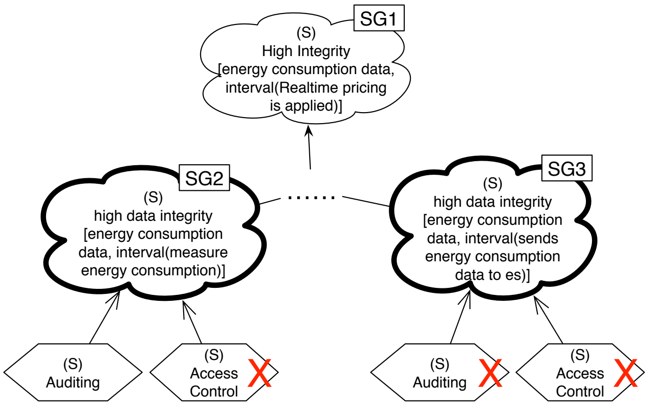


Figure 4.11: An example of the operationalization of critical security goals

In our example (Fig. 4.11), we have determined that *Auditing* is applicable to achieve security goal *SG2*, while others are inapplicable (indicated by red crosses). Note that if a security goal cannot be operationalized in one layer using layer-specific security patterns, there may be two possible reasons. Firstly, potential threats to the security goal may not exist in the current layer, and thus the security goal will be further elaborated and analyzed in lower layers, which will be shown in the next subsection. Secondly, the selected security patterns (shown in Fig. 4.10) may not be sufficient to satisfy the security goal, indicating our current set of patterns is not complete. We will further discuss completeness in Section 4.4.

It is worth noting that, in our subsequent research, we have proposed a systematic and tool-supported approach in Chapter 6-7 in order to facilitate the practical application of security patterns. We will further discuss the rationale of that piece of research in Section 4.4.

4.3.4 Cross-Layer Security Analysis

After finishing the security analysis within one layer, we switch the focus of our security analysis to the next layer down, i.e., generating a set of security goals at the next layer based on the analysis results of this layer. Then, a new round of security analysis will be performed for the next layer using the newly available information there, as indicated in Fig. 4.7. In particular, the cross-layer analysis focuses on analyzing the influences of applied security mechanisms and critical security goals in one layer.

Influences of applied security mechanism When a security goal has been operationalized into a specific security mechanism in one layer, the analyst first needs to manually

Table 4.6: Inference rules of cross-layer analysis

No.	**Content**
CRO.1	$\#create(sec\_goal(SG2)) \wedge sg\_attributes(SG2, IMP, security, A, INT) \wedge support(SG2, SM) \leftarrow operationalize(SM, SG1) \wedge sg\_importance(SG1, IMP) \wedge support(G, SM) \wedge has(A, G) \wedge interval\_of(INT, G)$
CRO.2	$\#create(sec\_goal(SG2)) \wedge sg\_attributes(SG2, IMP, SP, AS, INT2) \wedge \#create(sec\_goal(SG3)) \wedge sg\_attributes(SG3, IMP, security, A, INT2) \wedge and\_refine(SG2, SG1) \wedge and\_refine(SG3, SG1) \leftarrow interval\_of(INT1, T) \wedge sg\_attributes(SG1, IMP, SP, AS, INT1) \wedge support(G, T) \wedge has(A, G) \wedge interval\_of(INT2, G)$

check whether this security mechanism needs support from the lower-layer artifacts based on the specification of the security mechanism. If so, the security mechanism, as a specialization of *Task*, will introduce a functional requirement in the next layer down. Such support analysis is performed based on the instructions we have described in Section 4.2.1. As shown in the left part of Fig. 4.12, in the illustrating example, the applied security mechanism *Auditing* is determined to be implemented by an *Auditing Application* in the software layer. Thus, a new software goal *Measurement is audited (G1)* is introduced to the application correspondingly.

When it comes to cross-layer analysis, a security mechanism is different from a general task for its satisfaction; it requires not only functional support but also security support from the next layer down for its satisfaction. Otherwise, the security mechanism can be impaired and thus fails to protect the system. As such, given the functional support we have identified above, our analysis will also cover security support, introducing a security goal refinement, possibly within another layer, to protect corresponding artifacts. As illustrated in Fig. 4.12, security goal *SG2* is introduced into the software layer in order to ensure the *Security* of the *Auditing Application.* Thus, *SG2*, together with *G1*, supports security mechanism *SM1* in the social layer. The inference rule *CRO.1* (in Table. 4.6) has been defined to automate this analysis.

Note that the above analysis should be performed for all security mechanisms that operationalize security goals. If a security goal is operationalized into several alternative security mechanisms, all of them need to be analyzed. Thus, we are able to holistically analyze alternative security solutions once the security analysis has been performed in all layers.

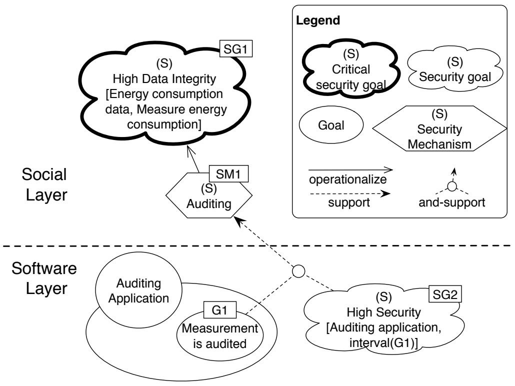


Figure 4.12: Security mechanisms cross-layer analysis

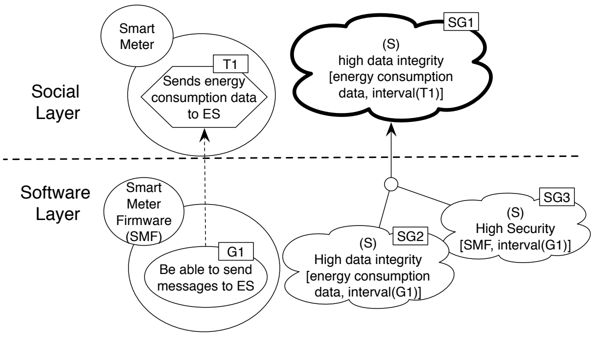

Smart Meter
T1
Sends energy consumption data to ES
Social Layer
SG1
(S) high data integrity [energy consumption data, interval(T1)]
Software Layer
Smart Meter Firmware (SMF)
G1
Be able to send messages to ES
SG2
(S) High data integrity [energy consumption data, interval(G1)]
SG3
(S) High Security [SMF, interval(G1)]

Figure 4.13: Cross-layer security analysis for security goals

Influences of critical security goals For each critical security goal that has been identified in one layer, we check whether it involves security issues in the lower layer. If a critical security goal concerns an interval which involves support from the next layer down, then we further elaborate the critical security goal to cover corresponding security issues. Firstly, we want to analyze layer-specific threats to the original asset in next layer; secondly, we need to take into account security of the supporting artifacts in the next layer (e.g., software applications and hardware devices), as attackers can indirectly impair the original asset by exploiting vulnerabilities of the supporting artifacts. For example, as shown in Fig. 4.13, *SG1* is concerned during the execution interval of *T1*, which is supported by software goal *G1* owned by *Smart Meter Firmware (SMF)*. Thus, *SG1* is and-refined to *SG2* and *SG3*: *SG2* concerns the same asset and security property of *SG1* but focuses on the execution interval of *G1*, while *SG3* concerns *security* of *SMF*. We have defined rule *CRO.2* which can be used to automate such analysis.

Once this cross-layer analysis is performed for one layer, iterative security analysis will be performed for the next layer down until reaching the bottom layer.

4.3.5 Holistic Security Solution Generation

Once security analysis has been performed in all three layers, we can derive a holistic security goal model that involves various security concerns regarding the system. By performing backwards analysis on the model, we can automatically generate a set of holistic security solutions, each of which consists of a number of security mechanisms that vary from the social layer to the infrastructure layer.

An example of a holistic security goal model is presented in Fig. 4.14, which only presents the related security goals and security mechanisms due to space limitation. In

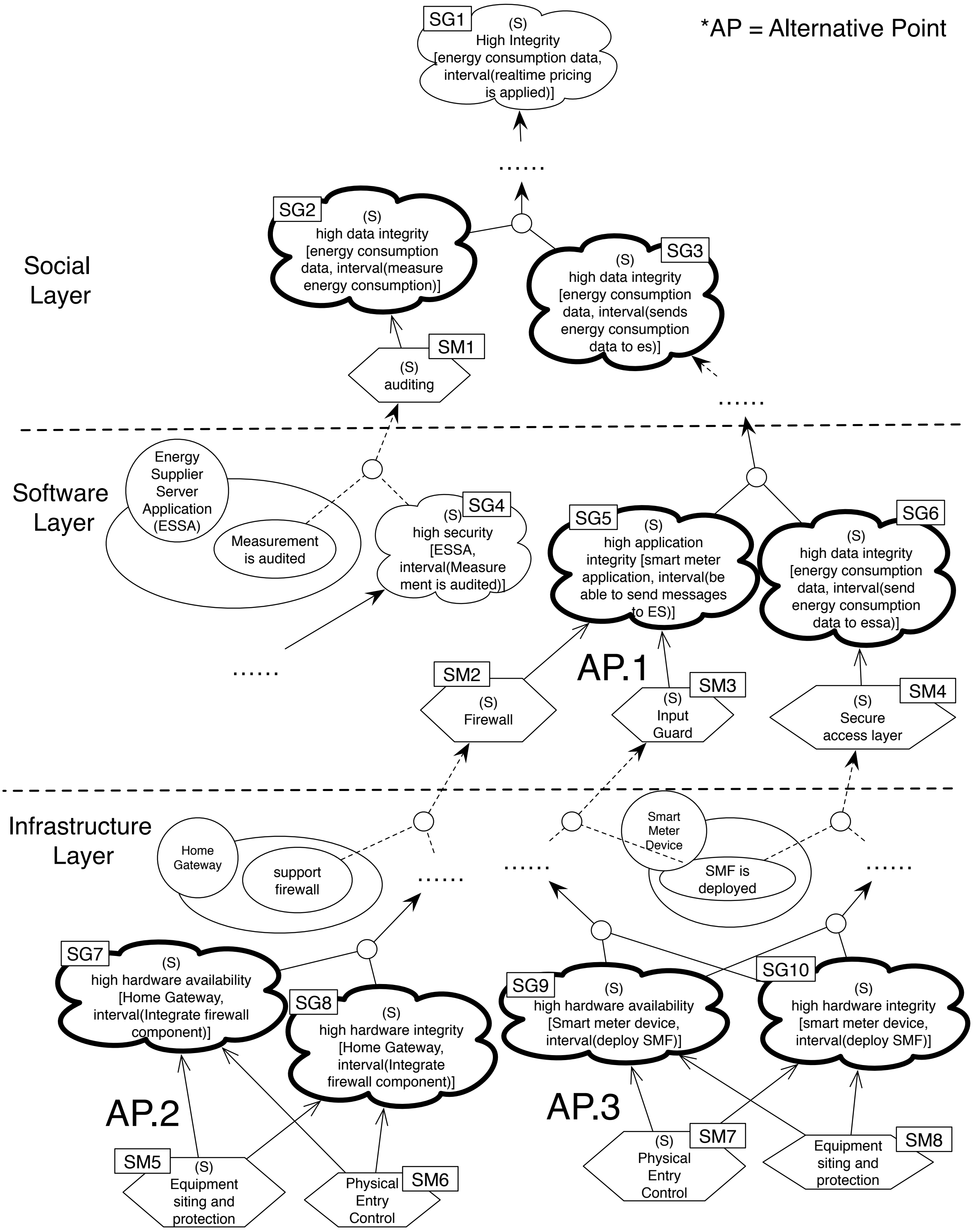

*AP = Alternative Point
SG1 (S) High Integrity [energy consumption data, interval(realtime pricing is applied)]
......
SG2 (S) high data integrity [energy consumption data, interval(measure energy consumption)]
SG3 (S) high data integrity [energy consumption data, interval(sends energy consumption data to es)]
Social Layer
SM1 (S) auditing
......
Software Layer
Energy Supplier Server Application (ESSA)
Measurement is audited
SG4 (S) high security [ESSA, interval(Measure ment is audited)]
SG5 (S) high application integrity [smart meter application, interval(be able to send messages to ES)]
SG6 (S) high data integrity [energy consumption data, interval(send energy consumption data to essa)]
......
AP.1
SM2 (S) Firewall
SM3 (S) Input Guard
SM4 (S) Secure access layer
Infrastructure Layer
Home Gateway
support firewall
......
......
Smart Meter Device
SMF is deployed
......
SG7 (S) high hardware availability [Home Gateway, interval(Integrate firewall component)]
SG8 (S) high hardware integrity [Home Gateway, interval(Integrate firewall component)]
SG9 (S) high hardware availability [Smart meter device, interval(deploy SMF)]
SG10 (S) high hardware integrity [smart meter device, interval(deploy SMF)]
AP.2
AP.3
SM5 (S) Equipment siting and protection
SM6 Physical Entry Control
SM7 (S) Physical Entry Control
SM8 Equipment siting and protection

Figure 4.14: Partial view of a holistic security goal model

particular, each alternative point (AP) in the figure indicates that there are alternative security mechanisms can be applied to satisfy a security goal. Note that choosing security mechanism *SM2* over *SM3* at *AP.1* leads to different influences on security analysis of the infrastructure layer, further resulting in different holistic solutions. In this example, by performing the backwards analysis, we derive in total four holistic security solutions which can achieve the root security goal. For example, one holistic solution is to have *auditing* at the social layer for ensuring the integrity of the *energy consumption data* when measuring energy consumption, to use *firewall* at the software layer to protect the integrity of the *smart meter application*, and to place *physical entry control* at the infrastructure layer in order to protect the *home gateway.*

Overall, our approach eventually produces a collection of alternative holistic security solutions that is specified in text, which will be passed to later stages of system development. Note that our approach currently does not deal with the selection among alternative security solutions, instead, we propose to use existing goal-oriented satisfaction analysis techniques to infer the best solution [Horkoff and Yu, 2013].

4.4 Discussions

In this section, we discuss several aspects of the proposed framework in more detail.

Motivations and benefits. The essence of our framework is to separate (security) requirements analysis of STSs into three different layers, each of which is associated with a specific solution domain. In particular, each layer corresponds to specific artifacts that are involved in STSs, such as business processes, software applications, and physical infrastructure etc., all of which constitute the entire STSs. Especially, the different domains of artifacts involve their own phenomena, which are normally analyzed by different people who have the specific knowledge background. Such phenomena motivate us to apply the *Divide and Conquer* paradigm [Knuth, 1998] to deal with the complexity of the requirements analysis of STSs, i.e., dealing with the requirements problem of each layer separately and merging the analysis results based on the cross-layer relations. As a result, we are able to capture alternative security solutions in individual layers, and analyze them together in a holistic manner and generate holistic security solutions.

Mappings between the three-layer framework and enterprise architecture framework. Enterprise architecture frameworks were first investigated by Zackman with the aim of efficiently aligning business requirements with IT systems. In particular, the Zackman framework provides a comprehensive taxonomy of artifacts of an enterprise, which is specified in terms of an user's perspective (e.g., business owner) and a descriptive focus (e.g., function), forming 36 intersecting cells [Zachman, 1987]. Based on this taxonomy, The Open Group Architectural Framework (TOGAF) is proposed by the open group, which divides an enterprise architecture into four categories, i.e., business architecture, data architecture, application architecture, technical architecture [Haren, 2011]. Although these enterprise architecture models connects enterprise artifacts at different layers, they do not capture and analyze alternative security solutions across layers.

We argue that our framework and TOGAF framework can complement each other. On one hand, the architecture categories defined in TOGAF can be intuitively mapped to our three-layer requirements framework, facilitating the construction of the three-layer requirements framework. In particular, the business architecture can be mapped to the social layer, each business activity in the business architecture will be modeled as a business task; the application architecture corresponds to the software application layer, an application function is then modeled as an application task; the technical architecture is mapped to the infrastructure layer, helping analysts to identify deployment tasks in this layer. On the basis of such mappings, when TOGAF architecture models are available for use, analysts are able to build the three-layer functional requirements models in a bottom-up fashion by asking “why” questions [Yu, 1997]. On the other hand, the holistic security solutions identified by our three-layer framework can be easily updated in the enterprise architecture model via the mappings. It is worth noting that, as The Open Group has provided detailed mappings between the TOGAF architectures and the Zackman framework [TOG, 2002], we can thus indirectly establish the mappings between our three analysis layers and the Zackman framework.

Application. Our approach is applied in the requirements analysis stage of system development lifecycle, which analyzes stakeholder’s security needs and holistically generates a set of security solutions across three layers that satisfy the security needs. Such security solutions will be designed and implemented in later system development stages, which are out of the scope of this paper. Our approach aims at decomposing the complicated requirements problem into small pieces, each of which is tackled by a specialized person or team. In other words, our approach offers a framework of collaborative analysis, involving business analysts, software architects, and infrastructure designers. As a result, to exert the power of our approach, a group of people with specialized domain and security knowledge of all three layers are required.

Security knowledge reuse. As security knowledge is difficult to acquire Souag et al. [2015], our framework proposes to incorporate security patterns from existing repositories. As such, the security analysis results depend on the range of selected security patterns. Although reusing existing security patterns can facilitate our knowledge-intensive security analysis, analysts are still required to grasp a full understanding of a security pattern before applying it, as reported in Araujo and Weiss [2002]. We have noticed this challenge.

In particular, when selecting a security pattern, we need to manually check the context of each candidate security pattern in order to determine whether it can be applied in current system settings. Furthermore, when apply a security pattern, we also need to first understand how the corresponding security mechanism works.

As a result, although there are more than 100 security patterns available from the repositories we use [Scandariato et al., 2008; Fernandez-Buglioni, 2013; Asnar et al., 2011a], our approach does not incorporate all of them. Instead, we currently incorporate 21 patterns from those repositories (as shown in 4.10), covering different layers and different security concerns, which serve as the baseline of our approach. When applying our approach, practitioners can further customize the set of security patterns according to their security expertise. For example, if practitioners have in-depth understanding of a specific set of the security patterns in the repositories, they can expand the set of selected security patterns with this set in order to produce more comprehensive security analysis results.

Regarding the above challenge, we were motivated to dedicatedly develop a systematic and semi-automatic approach to further facilitate the application of security pattern. In particular, we have proposed to model a significant number of security patterns in terms of contextual goal models, which can semi-automatically check the context of security patterns and can be seamlessly integrated into our three-layer framework. More details of this approach will be presented in Chapter 6. In addition, as a security mechanism (i.e., the solution of a security pattern) functions over existing system components, the application of the mechanism inevitably influences existing system functional requirements. Such influences have to be captured and enforced in order to correctly analyze system requirements. In Chapter 7, we will present a novel method which efficiently enforces the impact of security mechanisms imposed on system functional requirements, completing the last step of the application of security patterns.

Holistic threat analysis. As presented in Section 4.3.2, our framework relies on external threat analysis approach to identify threats to STSs. Because the threat analysis results can significantly influence the overall quality of the results of our holistic security analysis, it is important to continuously update our framework with regard to the recent advances in the corresponding research fields in order to timely deal with new challenges. For the security of STSs, it is important to analyze threats from a holistic viewpoint, covering different system components. In particular, multistage attacks which assemble individual attacks from different parts of STSs are imposing a new research challenge, which has not been tackled by existing approaches.

As such we have developed a holistic security attack analysis approach as part of our holistic security requirements framework in order to deal with this particular challenge.

This approach takes an attacker's perspective to analyze attack strategies, and then operationalizes the strategies into specific attack actions based on a collection of attack patterns. Details of this approach will be presented in Chapter 5.

Scalability. [Estrada et al., 2006] have acknowledged the scalability problem of the *i** modeling language based on their empirical evaluation. In particular, they have identified that the lack of mechanisms for modularization is the main cause to this problem. Our framework, as an extension of *i**, also needs to deal with this scalability problem. To relieve this problem, we have produced interventions which optimize the modularity of the entire analysis model. Firstly, the division of the three layers appropriately separates concerns related to different artifacts, and each layer can be modeled separately. With the prototype tool, analysts can determine the visibility of each layer, and thus can either browse the entire three-layer model or focus on a particular layer.

Secondly, we visually separate the security requirements model from the functional requirements model, while maintain the semantic connection between them. Many approaches have proposed to model and analyze security requirements together with functional requirements, e.g., misuse cases [Sindre and Opdahl, 2005] and Secure Tropos [Mouratidis, 2011]. However, considering the complexity of STSs, putting all these models together can further exacerbate the scalability problem. As such, we semantically connect the functional requirements model with the security requirement model via the *interval* attribute of security goals. In other words, each security goal actually targets a particular element in the functional requirements model. Such connections can be easily maintained and analyzed with the support of our prototype tool.

In addition to the above methodological designs, we have also implemented our prototype tool with specific features that contribute to relieving the scalability problem, details of which will be presented in Chapter 8.

4.5 Related Work

In this section, we compare the proposed three-layer security requirements analysis framework with the state of the art we have reviewed in Chapter 2.

NFR-based requirements analysis. Chung [1993] proposes to treat security requirements as a class of NFRs, and apply a process-oriented approach to analyze security requirements. In a subsequent work, Chung and Supakkul [2006] integrate NFRs with FRs in the UML use case model, which enable NFRs to be refined through functional requirement models. Another complementary approach introduced by Gross and Yu [2001] proposes to connect NFRs to designs via patterns. However, all of these NFR-based approaches mainly focus on information system analysis, and do not support requirements analysis in the business layer and the physical layer.

Security requirements analysis. A large number of security requirement analysis approaches have been proposed over last two decades. Most of these approaches focus on analyzing security requirements with regard to a particular aspect of information system. In particular, there are many approaches that focus on the social and organizational aspect: Mouratidis and Giorgini [2002] capture security intentions of stakeholders and interdependence among stakeholders; Giorgini et al. [2005b] investigate social relationships by integrating trusts and ownership into security analysis; Paja et al. [2013] capture and analyze security requirements of STSs in terms of commitments, using three views; Liu et al. [2003] analyze organizational risks by analyzing dependencies among social actors.

Another branch of work deals with security requirements for business processes. Rodríguez et al. [2011] propose an extension of UML activity diagram to model security requirements as part of the business process model, while Altuhhova et al. [2012] use BPMN constructs to represent security-related concepts and model secure business process models. Herrmann and Herrmann [2006] propose a systematic process to elicit and analyze security requirements from business processes models.

Most work is dedicated to analyzing security requirements of software, such as Attack Tree [Schneier, 1999], Misuse case [Sindre and Opdahl, 2005], and obstacle/anti-goal analysis [Van Lamsweerde and Letier, 2000; Lamsweerde, 2004]. All of these approaches are complementary to our proposal, as each of them can be fitted into one layer of the proposed framework. However, none of these approaches take the broad, holistic view of our framework, dealing with security dependencies between the social, software, and infrastructure layers.

Security pattern-based analysis. As security patterns have been recognized as an efficient way of designing system security, over a dozen security methodologies have been proposed based on security patterns [Uzunov et al., 2012]. As a representative of these methodologies, Uzunov et al. [2015] propose a comprehensive pattern-driven security methodology designed for general distributed systems, which is based on a significant number of well-documented security patterns [Fernandez-Buglioni, 2013]. This methodology covers both the requirements analysis stage and the design stage of software development lifecycle. In the requirements analysis stage, they first elicit secure use cases based on misuse activities. To satisfy these secure use cases, they then identify corresponding security solutions by using security patterns. After passing the security solutions to the design phase, they apply security solution frames, which consist of architectural level security patterns and micro process patterns, to generate security system design.

Compared to their methodology, our framework exclusively focuses on the requirements analysis stage and has several advantages. Firstly, our analysis targets STSs, which involve more heterogeneous components and are more complicated than general distributed systems. In particular, both social and physical security concerns are taken into account in our methodology. As such, our analysis can provide more comprehensive security protections. Moreover, building on goal modeling languages, our approach can capture and analyze alternative security solutions from a holistic viewpoint, and thus generates solutions whose effectiveness can be evaluated, helping to select the best possible solution. Lastly, our methodology is supported by a prototype tool, enabling us to deal with the complexity of large-scale STSs; while their methodology is performed manually, and they "*believe it can be used without tool-support on at least small- to mid-size projects*" [Uzunov et al., 2015].

Security requirements transformation. Many approaches have been proposed to transform security requirements captured in a high-abstraction level to the security design in a low-abstraction level in order to maintain security requirements throughout the entire life-cycle of system development. Mouratidis and Jurjens [2010] connect security requirements and security designs by integrating Security Tropos with UMLsec. In particularly, they provide guidelines to transfer Security Tropos models to UMLsec class diagrams and deployment diagrams. Menzel et al. [2009] propose a model-driven approach that transfers security requirements, which are captured at the business process layer, to concrete security implementations/configurations by using patterns. Similarly, Rodríguez et al. [2010] apply MDA techniques to transform secure business process model into analysis class diagram and use case diagram .

The above approaches focus on maintaining security requirements identified in the early stage during later design stages. Orthogonally, our framework looks at security requirements not in terms of development stages, but different solution domains (business, software, infrastructure), each of which will produce specific artifacts that constitute STSs. In addition, these approaches focus on aligning security requirements in a top-down manner, but not analyze all security requirements in different layers together. On the contrary, our approach captures alternative security solutions in each of the three layers and connect them across layers. As such, we are able to generate the best holistic security solutions, taking into account all security concerns in the three layers together.

Multilayer requirement analysis. A number of approaches have been proposed to analyze requirements in multiple levels. Lankhorst et al. [2009] provide an integrated view for enterprise architecture, consisting three layers, to enable impact and change analysis covering all relevant aspects. Within each of the three layers, they consider both external services that are delivered by one layer and internal services that specify how the layer is implemented. Cui and Paige [2012] propose an integrated framework for system requirement developments, which consists of six levels and aligns requirements with business motivations. Specifically, their framework is intended to cover organizational requirements, product requirements, and hardware/software requirements. Ranjan and Misra [2006] argue that the goal-based analytic technique should be applied to different level of abstraction in order to better understand requirements of specific domains.

Although these approaches have conceptually presented the multilayer structure, none of them have pragmatically developed analysis methods for analyzing (security) requirements of STSs. Our approach acknowledges the intention to capture requirements in different layers of STSs. In addition to the above approaches, we delve into influences of security requirements in different layers, and have proposed a systematic process and corresponding analysis methods to pragmatically analyze security requirements in a holistic manner.

4.6 Chapter summary

In this chapter, we present a three-layer security requirements analysis framework, which is the essential part of this thesis. We have introduced the full details of this framework, starting from explaining the rationale of the three-layer structure (Section 4.1). Based on such a structure, we propose a three-layer requirements modeling language which can capture requirements of STSs in three layers, while maintaining connections across layers (Section 4.2). Given a three-layer requirements goal model, we have defined a systematic security requirements analysis process, which iteratively performs security analysis in each of the three layers and eventually generates holistic security solutions that satisfy stakeholder's security requirements. In particular, we propose a set of security requirements analysis methods and define a collection of inference rules to (semi-)automate those analysis methods. In Section 4.4, we discuss a number of issues of the proposed three-layer framework, such as the rationale and the potential of the framework. Specifically, we discuss the challenges in holistic threat analysis and the reuse of security patterns, based on which we explain how other parts of this thesis (i.e., Chapter 5-7) contribute to this three-layer framework. Finally, we compare our framework with related work in Section 4.5.

Chapter 5

A Holistic Security Attack Modeling and Analysis Approach

> *Know your enemies and know yourself, you will not be imperiled in a hundred battles.*
>
> Sun Tzu

In this chapter, we present a holistic security attack analysis framework, which complements our holistic security requirements analysis framework (in Chapter 4). The proposed attack analysis approach is performed from an attacker's perspective, taking the three-layer security requirements goal model as input and exploring possible attacks on the target system. In particular, our attack analysis consists of two parts: firstly, we identify an attacker's strategies by systematically elaborating an attacker's malicious intentions; secondly, we leverage CAPEC attack patterns to analyze how attackers implement an identified attack strategy in terms of realistic attack behaviors. The output of the holistic attack analysis is a set of alternative (multistage) attacks, which are required by our holistic security requirements analysis framework for identifying critical security goals (as described in Section 4.3.2).

In the remaining part of this chapter, we first provide the overview of our approach, explaining the rationale of our approach and presenting challenges that are solved by the approach (Section 5.1). Then we present the two parts of our approach in detail in Section 5.2 and Section 5.3, respectively. In Section 5.4, we evaluate this holistic attack analysis approach using a smart grid scenario. It is worth noting that we here exclusively focus on evaluating this attack analysis approach, while in Chapter 9 we focus on validating the holistic security requirements analysis framework. Lastly, we present related approaches and compare them with our proposal in Section 5.5.

5.1 Approach Overview

Security breaches in STSs have repeatedly resulted in multi-million dollar losses per year to large organizations, and this cost is on the rise [Ponemon, 2015]. A primary reason for these breaches is the complexity and heterogeneity of STSs, consisting of people, processes, technology and infrastructures. All of these heterogeneous components raise a plethora of security concerns and present a larger attack surface compared to more homogeneous software systems. In addition, given the increased number of attacks, STSs have become the ideal target of multistage attacks, as attackers can compose atomic attack actions associated with different components to perform more dangerous attacks [Mitnick and Simon, 2011]. Failing to consider such diverse attacks while designing STSs can result in vulnerable systems.

Many approaches have been proposed to analyze attacks, but focusing on a specific layer of STSs. For example, Mitnick and Simon [2011] analyze social engineering attacks; Sindre and Opdahl [2005] propose Misuse Cases to capture attacker's malicious behaviors on the business process level; Weingart [2000] surveys and discusses known physical attacks and presents corresponding security methods. However, there is no approach that takes into account all layers of STSs and holistically identifies attacks. Moreover, a lack of knowledge about impending attacks introduces another challenge to attack analysis, as analysts cannot *realistically* identify how attackers might attack a system and thus have either false positives or false negatives during security analysis. Although the security community has summarized practical attack knowledge in terms of 504 CAPEC attack patterns[1], there are no methods which efficiently incorporate such knowledge into attack analysis. As such, analysts are reluctant to use such patterns in practice, as "the impressive size and scope of CAPEC may make it intimidating for people to jump in" [Shostack, 2014].

Regarding such challenges, we propose a holistic attack analysis framework, which can holistically analyze the security attacks on STSs. As shown in Fig. 5.1, three-layer requirements goal models (as introduced in Chapter 4) are taken as input of the attack analysis framework to provide domain information about the target system. On the other hand, the outcome of our holistic attack analysis is a set of alternative (multistage) attacks on the system, which are transferred back to the three-layer security requirements framework, helping us to analyze critical security requirements (Section 4.3.2). Note that such final outcomes (i.e., all the attack alternatives) are specified in text, while the established attack models (i.e., the left part of Fig. 5.1) are only middle products which are produced by our holistic attack analysis approach. In such a way, this holistic attack

---

[1] `https://capec.mitre.org`

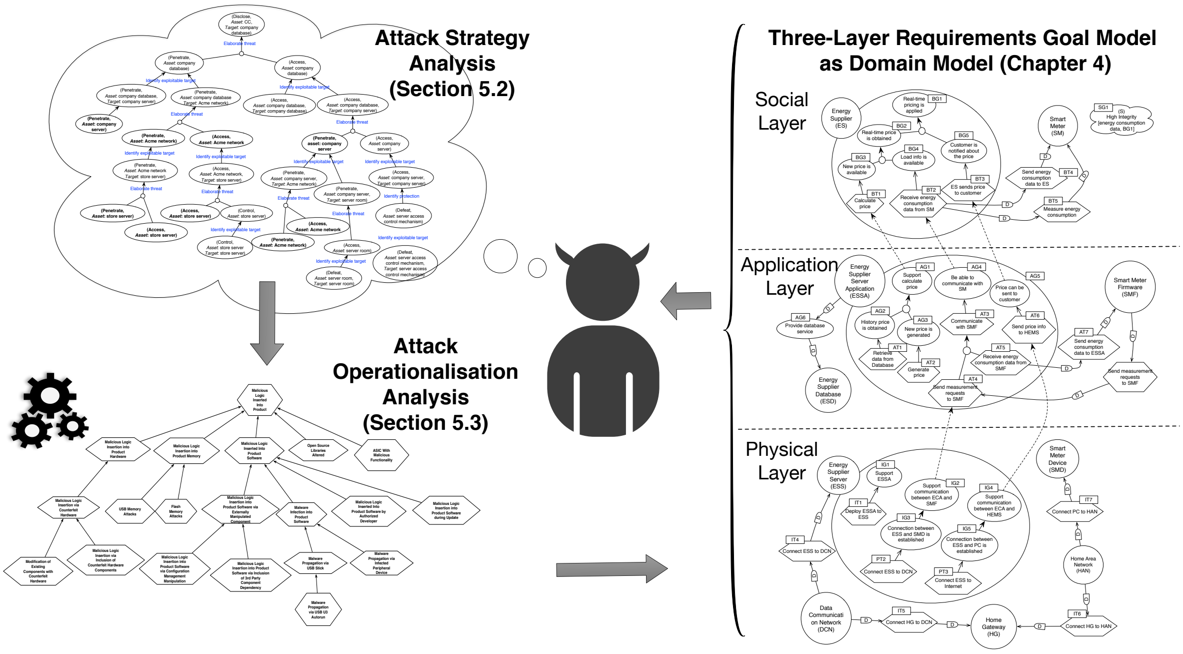

Attack Strategy Analysis (Section 5.2)
Attack Operationalisation Analysis (Section 5.3)
Three-Layer Requirements Goal Model as Domain Model (Chapter 4)
Social Layer
Application Layer
Physical Layer

Figure 5.1: An overview of the holistic attack analysis framework

analysis fills the vacancy of threat analysis for our holistic security requirements analysis framework (as discussed in Section 4.4).

Our proposal takes an attacker's perspective to analyze security breaches, which has been advocated as an effective approach [Schneier, 1999; Sindre and Opdahl, 2005]. In particular, our attack analysis consists of two parts, as shown in the left part of Fig. 5.1. Each of these two parts addresses a particular challenge of attack analysis, contributing to the state-of-the-art.

Firstly, we contend that it is important to explicitly analyze the rationale behind an attacker's actions, which we call attack strategies. Such strategies can include alternative attack plans, each of which may consist of multiple steps. For example, to disclose a data asset, one attack strategy can be finding out all software applications that process the data and then hacking the applications to disclose the data, or directly hacking the hardware that stores the data. Several existing approaches can model attack strategies based on external security knowledge sources, e.g., [Lamsweerde, 2004; Mouratidis et al., 2004; Elahi et al., 2010]. However, when analyzing *holistic and multistage* attacks on STSs, it is very difficult to find available knowledge sources that provide such attack strategies. Thus, as the first contribution of our holistic attack analysis framework, we propose to generate an attacker's strategies by systematically elaborating the attacker's malicious intentions (i.e., the top-left part of Fig. 5.1).

Secondly, after generating the attack strategies which describe what and when to attack, analysts need to know how an attacker behaves to achieve the attack. In order to bridge the knowledge gap between attackers and defenders, we leverage CAPEC attack patterns for realistic attack knowledge. In particular, beyond the existing CAPEC-related approaches[2], we propose a method to semi-automatically select the best attack pattern among the large number of candidates (CAPEC includes 504 patterns thus far). Therefore, the second contribution of our holistic attack analysis framework is operationalizing attack strategies in terms of realistic attack behaviors, based on which we can identify alternative (multistage) attacks on the target systems (i.e., the bottom-left part of Fig. 5.1).

5.2 Analyzing Attack Strategies via Anti-Goal Refinement

In this section, we present the first part of our holistic attack framework, which generates attack strategies by iteratively refining an attacker's high-level anti-goals. In particular, we focus on how to systematically refine anti-goals in order to obtain comprehensive attack strategies. Thus, our primary goal is to develop an approach, grounded in real evidence, to support systematic exploration of attack strategies, producing strategies which are more complete. To achieve this goal, we perform the following steps.

1. perform a grounded study on three real attack scenarios [Mitnick and Simon, 2011] in order to investigate how attackers elaborate their malicious intentions in reality, from which we identify five anti-goal refinement patterns.
2. propose an anti-goal refinement approach, which systematically refines anti-goals using the identified anti-goal refinement patterns, and eventually reveals attack scenarios.
3. evaluate the proposed refinement framework by applying it to a different credit card theft scenario [Skoudis and Liston, 2005], the result of which shows that our framework is able to generate a comprehensive attack strategy, which not only covers the reported attack scenarios, but also reveals new attack scenarios.

5.2.1 Real Attack Scenario Examination

In this section, we examine three real attack scenarios in order to understand attack strategies that have been applied in reality. In particular, we apply the anti-goal modeling to real attack scenarios, and then investigate the rationale behind each anti-goal refinement within the modeled scenarios, and finally extract five anti-goal refinement patterns. We

---

[2] `https://capec.mitre.org/community/citations.html`

first briefly introduce the real attack scenarios that we examine, and then present our examination on these scenarios in detail.

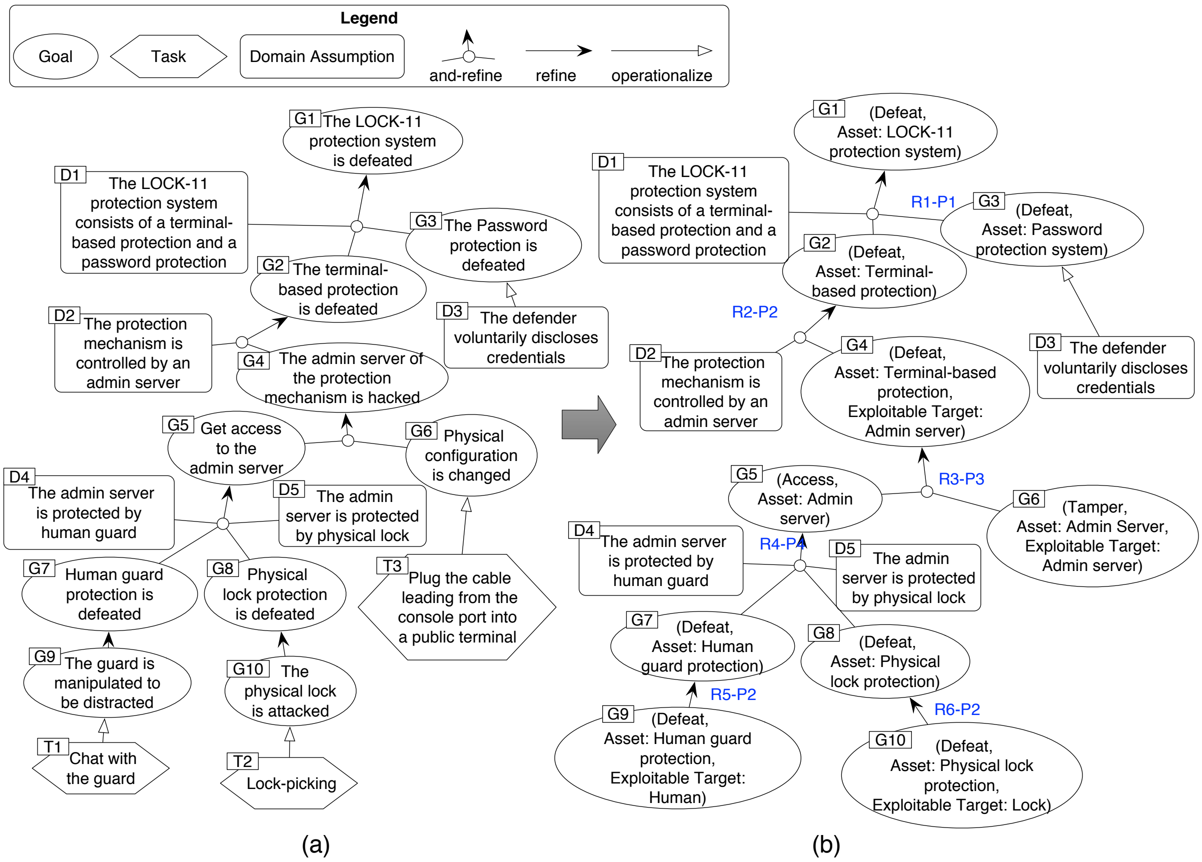
Sample attack scenarios. To reveal sophisticated attack strategies from the examination, we define three criteria for selecting the attack scenarios to be examined. Firstly, the attacks should cover a wide spectrum of attack techniques, from social engineering to software/hardware hacking. Secondly, we look for multistage attacks that consist of a sequence of steps, rather than an atomic attack that is launched with a single exploit. Thirdly, the description of the attacks should present not only attack actions performed by attackers, but also the intentions motivating the actions.

According to the above criteria, we select three attack scenarios that are documented in Mitnick's book [Mitnick and Simon, 2011, Ch. 11]. Each of these attack scenarios involves both social and technical issues, and consists of multiple attack steps. In this case, the author narrates the entire attack process in detail, shedding light on both the why and how for each attack step. The general problems and contexts of these attack scenarios are as follows:

- *Easy Money*: Two attackers aim to defeat a security product that is designed for access control in order to get prize money. The product applies terminal-based security technique, which identifies system users based in part on the particular computer terminal being used.
- *Dictionary as an Attack Tool*: An external attacker intends to steal the source code of a new electronic game, which is developed by a global company. The source code is stored on an unknown server of the company.
- *The Speedy Download*: An external attacker wants to obtain some confidential files of an accounting firm in order to affect the stock price of publicly traded companies. The confidential files are stored on the workstation, which can only be accessed from the company's local network.

In this section, we illustrate the examination process using the "Easy Money" scenario. The complete set of examination results can be found in Appendix A.

Construct initial anti-goal models. We first build initial anti-goal models according to the textual description of the attack scenarios. The construction of anti-goal models is carried out by combining top-down and bottom-up analysis. The content of each node is described in natural language, using a particular part of the scenario description. Fig. 5.2(a) presents the entire anti-goal model that is built from the "Easy Money" attack scenario. Note that we capture the attack actions as *tasks* so as to provide a full view of the scenario, but our analysis focuses on the anti-goal refinement rather than the anti-goal operationalization. To easily reference to the elements of the anti-goal model, we annotate each element with regard to the type of the element. In particular, *G* stands for *Goal*, *T* stands for *Task*, and *D* stands for *Domain Assumption*.


Legend
Goal
Task
Domain Assumption
and-refine
refine
operationalize
G1 The LOCK-11 protection system is defeated
D1 The LOCK-11 protection system consists of a terminal-based protection and a password protection
G3 The Password protection is defeated
G2 The terminal-based protection is defeated
D3 The defender voluntarily discloses credentials
D2 The protection mechanism is controlled by an admin server
G4 The admin server of the protection mechanism is hacked
G5 Get access to the admin server
G6 Physical configuration is changed
D4 The admin server is protected by human guard
D5 The admin server is protected by physical lock
T3 Plug the cable leading from the console port into a public terminal
G7 Human guard protection is defeated
G8 Physical lock protection is defeated
G9 The guard is manipulated to be distracted
G10 The physical lock is attacked
T1 Chat with the guard
T2 Lock-picking
(a)
G1 (Defeat, Asset: LOCK-11 protection system)
D1 The LOCK-11 protection system consists of a terminal-based protection and a password protection
R1-P1
G3 (Defeat, Asset: Password protection system)
G2 (Defeat, Asset: Terminal-based protection)
R2-P2
D3 The defender voluntarily discloses credentials
D2 The protection mechanism is controlled by an admin server
G4 (Defeat, Asset: Terminal-based protection, Exploitable Target: Admin server)
R3-P3
G5 (Access, Asset: Admin server)
G6 (Tamper, Asset: Admin Server, Exploitable Target: Admin server)
D4 The admin server is protected by human guard
R4-P4
D5 The admin server is protected by physical lock
G7 (Defeat, Asset: Human guard protection)
G8 (Defeat, Asset: Physical lock protection)
R5-P2
R6-P2
G9 (Defeat, Asset: Human guard protection, Exploitable Target: Human)
G10 (Defeat, Asset: Physical lock protection, Exploitable Target: Lock)
(b)

Remark: the label *Rx-Py* means the refinement *Rx* falls into the pattern *Py* (Table 5.2)

Figure 5.2: Anti-goal models that are built from the “Easy Money” attack scenario

Characterize anti-goals. It is our goal to capture anti-goals and their refinements, such as in Fig. 2(a), in a more structured and abstract way. Thus we characterize each anti-goal with a structured description language, which is specified in Table 5.1 by using EBNF syntax.

Each anti-goal is characterized by one threat and one or several attributes (*Rule_1*). A *threat* presents an undesired state that an attacker wants to impose on the targeting system. We classify threats using an existing, established threat categorization, STRIDE, provided by Microsoft [Shostack, 2014]. STRIDE is an acronym that stands for six threat

Table 5.1: The EBNF syntax of the structured description language

Rule_1: <anti-goal> ::= <threat>, <attribute-description>+

Rule_2: <threat> ::= 'tamper' | 'disclose' | 'spoof' | 'repudiate' | 'deny' | 'reach' |'access' |'control' | 'defeat'

Rule_3: <attribute-description> ::= <attribute>, <descriptor>

Rule_4: <attribute> ::= 'asset' | 'exploitable target' | 'interval'

categories: *Spoofing*, *Tampering*, *Repudiation*, *Information disclosure*, *Denial of service*, and *Elevation of privilege*. These threat categories provide comprehensive coverage of security threats and have been adopted and investigated in both academia and industry [Shostack, 2014; Scandariato et al.]. Note that we describe the threat categories in terms of their essential actions rather than the full description (*Rule_2*), as the threat actions are more succinct and intuitive when combined with other attributes. For example, instead of specifying *information disclosure*, we represent this type of threat using *disclose*. For the threat *Elevation of privilege*, we specifically consider three threat actions *reach*, *access*, and *control*, each of which implies a particular level of privilege. When comparing the available categories in STRIDE to the anti-goals collected from the real cases, we find the need to add an additional threat category, specifically, "defeated security mechanism" which captures the attacker intention to break system protections.

Moreover, we characterize anti-goals with three other attributes (*Rule_4*): an *asset* is anything of value to stakeholders, it is normally the object of a threat; an *exploitable target* is a component of a system, which involves assets and has vulnerabilities that are exploitable by attackers; an *interval* represents the time period, during which attackers carry out attacks. Note that values of these attributes are described in text (*Rule_3*). By using the structured description language, we characterize the anti-goals in the initial anti-goal model, resulting in a characterized model as shown in Fig. 5.2(b).

Identify refinement patterns. Once the characterized anti-goal model is obtained, we investigate each refinement relation in detail, on the basis of which we can identify refinement patterns.

We first investigate the influences of refinement relations on the refined anti-goals, i.e., what have been changed from the refined anti-goals to their sub-goals. For example, as shown in Fig. 5.2(b), the influence of refinement *R1* is that the asset of the anti-goals *G2* and *G3* have been modified from their parent goal *G1*. After performing such analysis on all 25 refinement relations in the three attack scenarios, we cluster refinement relations

Table 5.2: Summarized refinement patterns

No.	Pattern Name	Pattern Influences	Occurrence
P1	Asset-based refinement	Modify asset	2
P2	Target-based refinement	Add exploitable target	7
P3	Threat-based refinement	Modify asset; modify threat; remove exploitable target	10
P4	Protection-based refinement	Modify threat; modify asset; remove exploitable target	4
P5	Interval-based refinement	Modify interval	2

with similar influences, based on which we summarize five refinement patterns. Table 5.2 presents the identified refinement patterns, as well as their influences and number of occurrence in the three attack scenarios. Examples of the application of the refinement patterns can be found in Fig. 5.2(b), where each refinement relation is annotated with its corresponding refinement pattern.

5.2.2 An Anti-Goal Refinement Approach

The extracted five anti-goal refinement patterns shed light on different ways to refine an anti-goal, based on which we propose an anti-goal refinement approach. This approach efficiently leverages the proposed refinement patterns to refine an attacker's high-level anti-goals and to generate comprehensive attack strategies, the analysis process of which is shown in Fig. 5.3. Note that the output of this analysis process is a set of attack strategies which are specified in terms of anti-goal models. Such models are not the final output of the entire holistic attack analysis approach, but middle products, which are used by the attack operationalization analysis that will be described in the next section.

Each of these steps makes use of one particular refinement pattern, and the detailed guidelines for performing these steps are presented below. It is worth noting that we describe the anti-goal refinement framework from an attacker's perspective to clearly show the rationale of the strategy, but the corresponding analysis is actually performed by security analysts with a complete set of system information in order to discover all potential attack scenarios. In particular, the description of each analysis step focuses on addressing the following issues:

- *Rationale.* We first describe the rationale of each analysis step, which explains the design of the analysis process (Fig. 5.3). Note that the proposed anti-goal refinement framework is a specific way to analyze attack strategies, and does not exclude other possible ways.

- *Input.* We then specify the inputs that are required for performing the analysis step. It is worth noting that our proposal is a general framework, which is not associated with specific models. Thus, for inputs, we only describe the types of information that are required, and all models that capture the corresponding information can be used.
- *Sanity check.* Our framework is intended to cover various attacks and thus provides a comprehensive security analysis. As a result, a single anti-goal can lead to a very large model. To deal with this complexity, we propose to prune the model as part of its construction, i.e., performing sanity checks after each analysis step in order to reduce the refinement space.
- *Stop criteria.* Finally, we describe the stop criteria of each analysis step.

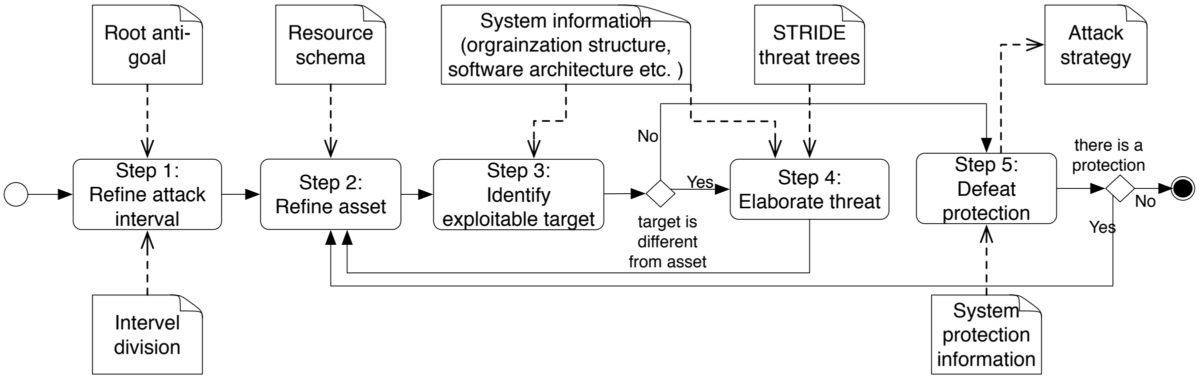

Root anti-goal
Resource schema
System information (orgrainzation structure, software architecture etc.)
STRIDE threat trees
Attack strategy
Step 1: Refine attack interval
Step 2: Refine asset
Step 3: Identify exploitable target
No
Yes
target is different from asset
Step 4: Elaborate threat
Step 5: Defeat protection
there is a protection
No
Yes
Intervel division
System protection information

Figure 5.3: An analysis process of anti-goal refinement

Step 1: Refine attack interval. System security settings can change over time, affecting an attacker's anti-goals. As the first step, an attacker applies the interval-based refinement pattern in order to concentrate on specific time intervals. Thus, this analysis step requires specific domain knowledge about the division of time intervals. In particular, for each interval-based refinement, the analyst should check whether system security settings have been changed from the original interval to its sub-intervals. If the security settings remain the same, this refinement will not contribute to disclosing new attack scenarios and should be pruned. The interval refinement analysis is completed once the finest-grained intervals have been reached via refinements.

Step 2: Refine asset. Given a composite asset, it is easier for an attacker to attack a fine-grained part of the asset rather than attacking the composite asset as a whole. An attacker can leverage the asset-based refinement pattern to generate sub-goals that focus on more specific sub-assets. The asset-based analysis takes the system resource schema as input, which documents "part-of" relations between system resources. To identify system assets among system resources, we refer to the asset identification process that is specified in ISO27005:2011 [ISO and Std, 2011, Annex B], which deals with both the primary assets and the supporting assets. In particular, the primary assets include *business processes and activities* and *information*; the supporting assets include *hardware*, *software*, *network*, *personnel*, *site*, and *organization's structure*. This analysis step is completed when all identified assets in the resource schema are analyzed.

Step 3: Identify exploitable target. Once an attacker has determined the assets he intends to impair, he needs to find out corresponding vulnerable system components (a.k.a. exploitable targets), by exploiting which the assets will be damaged. In particular, an asset can be involved in system components in different ways according to the type of the components, e.g., an information asset can be accessed by people, processed by software, or stored in hardware. Note that the asset and the exploitable target of an anti-goal can be the same, if the asset itself is a vulnerable system component.

We here consider the types of vulnerable system components in line with the list of supporting assets presented in ISO27005:2011 [ISO and Std, 2011, Annex B.1.2]. As such, corresponding system information is required, e.g., information of system infrastructure, software architecture, and organization structures. When identifying the exploitable target, analysts should check the risk of exploiting the target, e.g., using the CORAS approach [Lund et al., 2010]. If the risk is under certain threshold, determined by the analysts, the target is assumed to be secure and is excluded from this refinement step. After using the target-based refinement pattern to identify all potential exploitable targets, this analysis step is complete.

Step 4: Elaborate threat. If an attacker aims to impose a threat to an asset by exploiting a target, which is different from the asset, then the attacker should identify new threats that he wants to impose on the exploitable target in order to successfully impose the original threat to the asset. For example, if an anti-goal is intended to disclose (*threat*) confidential files (*asset*) that are stored in a database (*exploitable target*), then it can be refined to getting access to (*new threat*) the same database (*new asset*) by using the threat-based refinement pattern.

When applying the threat-based refinement pattern, the system information and related security knowledge are required to support the threat elaboration. Specifically, we refer to 19 STRIDE threat trees as the security knowledge sources, which describe alternative ways about how a threat category can be refined to other categories. As we specify the threats of anti-goals using the STRIDE threat categories, the application of the STRIDE threat trees can be seamlessly integrated into this analysis step. In order to discover all potential attack scenarios, once we identify the new threats to the exploitable target that can lead to the original threat to the asset, we iteratively analyze the new threats to the exploitable target through the analysis step 2 and 3, i.e., we treat the exploitable target as a new asset. Such as in the aforementioned example, the newly introduced sub-goal "getting access to (*new threat*) the database (*new asset*)" concerns the database as a new asset, which was the exploitable target in the parent anti-goal.

Step 5: Defeat protection. From an attacker's perspective, security protections are obstacles to his attacks. If the attacker targets a system component which is protected by some security mechanisms, such as encryption and firewalls, then he needs to first defeat the mechanisms in order to achieve their anti-goals. According to the knowledge about system security design, the attacker can use the protection-based refinement pattern to generate anti-goals against related security protection mechanisms.

Each of the newly generated anti-goals concerns a specific protection mechanism as its asset and is intended to defeat it. Similar to the last analysis step, as long as new assets have been identified in the new anti-goals, subsequent analysis will iteratively refine assets and identify targets for the new anti-goals, i.e., going back to the analysis step 2. It is worth noting that during the anti-goal refinement, we focus on identifying which protection mechanisms need to be defeated by exploiting which targets, not answering which specific attack techniques to be used to defeat the mechanisms. Once there are no further security protections to be defeated, i.e., there are no new assets have been found, the analysis reaches an end as all potential attack scenarios have been obtained.

5.2.3 Evaluation

In order to evaluate the proposed anti-goal refinement approach, we applied it to a realistic credit card theft scenario, which is documented in a textbook [Skoudis and Liston, 2005, Ch.12]. In this section, we first introduce the evaluation scenario, and then illustrate the application of this approach to the scenario. Finally, we evaluate the resulting anti-goal model. Fig. 5.4 presents part of the resulting model, which is used for illustrating the application of our approach. In particular, we focus on evaluating the usability of our anti-goal refinement approach and the quality of the analysis results, which are detailed in the following research questions:

- ***RQ1***: *Can our approach be easily used?*
  - ***RQ1.1***: *How long does it take to apply our approach?*
  - ***RQ1.2***: *Are there any difficulties in applying our approach?*
- ***RQ2***: *Can our approach effectively identify attack strategies?*
  - ***RQ2.1***: *Can our approach identify realistic attack strategies documented in the textbook?*
  - ***RQ2.2***: *Can our approach identify additional attack strategies beyond the strategies documented in the textbook?*

Credit card theft scenario. This scenario presents a complicated multistage attack in reality, which is documented in Skoudis's hacking book [Skoudis and Liston, 2005, Ch.12] and is different from the source of the previous three real attack scenarios. Specifically, in this scenario, there is a widgets corporation which operates more than 200 retail stores. Each retail store communicates with the central corporate network by using a VPN, and all credit card transactions are seamlessly moved from individual stores back to the central database. Each store has several Point-of-Sale (POS) terminals, which access the local store network using wireless access points. Each store also has a store server, which processes credit card transactions and forwards the transactions back to the company server.

Applying the anti-goal framework. As a pre-step, we first processed the scenario description to extract information that was required by the analysis. Specifically, we captured the attacker's high-level malicious intention, i.e., steal customer's credit card information, and modeled it as his root anti-goal (*G1* in Fig. 5.4). In addition, we captured related domain information, such as asset relations and system infrastructure, which was used to refine anti-goals. Having the root anti-goal and related domain information as input, we applied the proposed approach (Fig. 5.3) to refine the root anti-goal into operational anti-goals and thus generated a comprehensive attack strategy. We summarize this process as follows.

1. As the scenario only dealt with the credit card system in a general time span and did not describe any particular time interval, we opted not to apply the first step. In other time-sensitive cases, this step would be applied.
2. We refined the asset of the root anti-goal *G1* according to the composition relations among assets. As the entire set of credit card information was composed of information of credit cards that were processed in different retailer stores, the root-goal *G1*

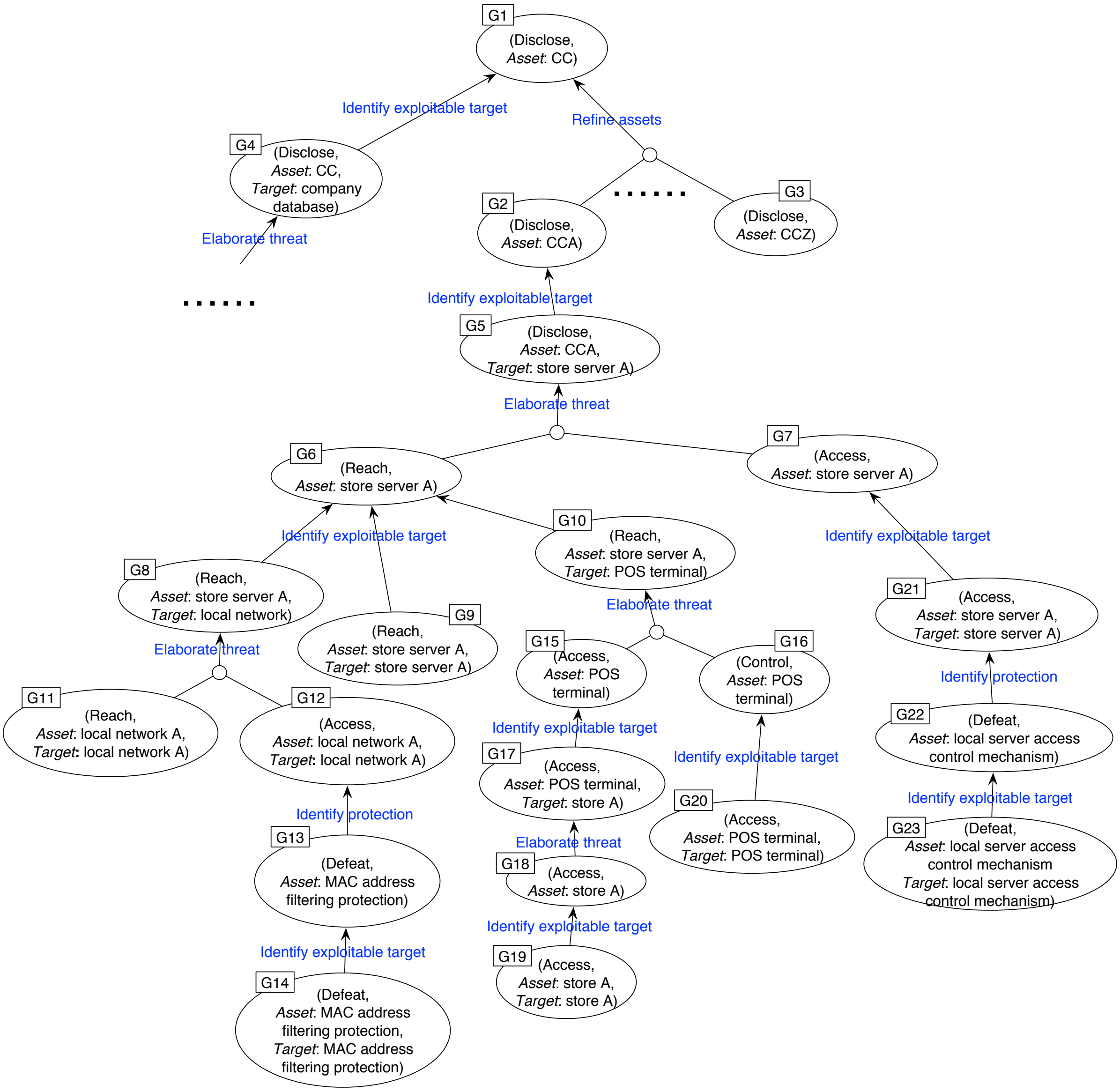

G1
(Disclose, *Asset*: CC)
Identify exploitable target
Refine assets
G4
(Disclose, *Asset*: CC, *Target*: company database)
Elaborate threat
G2
(Disclose, *Asset*: CCA)
G3
(Disclose, *Asset*: CCZ)
Identify exploitable target
G5
(Disclose, *Asset*: CCA, *Target*: store server A)
Elaborate threat
G6
(Reach, *Asset*: store server A)
G7
(Access, *Asset*: store server A)
Identify exploitable target
G10
(Reach, *Asset*: store server A, *Target*: POS terminal)
Identify exploitable target
G8
(Reach, *Asset*: store server A, *Target*: local network)
G9
(Reach, *Asset*: store server A, *Target*: store server A)
Elaborate threat
G21
(Access, *Asset*: store server A, *Target*: store server A)
Elaborate threat
G15
(Access, *Asset*: POS terminal)
G16
(Control, *Asset*: POS terminal)
Identify protection
G11
(Reach, *Asset*: local network A, *Target*: local network A)
G12
(Access, *Asset*: local network A, *Target*: local network A)
Identify exploitable target
G17
(Access, *Asset*: POS terminal, *Target*: store A)
Identify exploitable target
G20
(Access, *Asset*: POS terminal, *Target*: POS terminal)
G22
(Defeat, *Asset*: local server access control mechanism)
Identify exploitable target
Identify protection
G13
(Defeat, *Asset*: MAC address filtering protection)
Elaborate threat
G18
(Access, *Asset*: store A)
G23
(Defeat, *Asset*: local server access control mechanism *Target*: local server access control mechanism)
Identify exploitable target
Identify exploitable target
G19
(Access, *Asset*: store A, *Target*: store A)
G14
(Defeat, *Asset*: MAC address filtering protection, *Target*: MAC address filtering protection)

Figure 5.4: Applying anti-goal refinement analysis to the scenario of credit card theft (excerpt)

should be and-refined to more than 200 sub-goals, each of which was intended to disclose credit cards of one particular store. Since all the retailer stores had homogeneous design and configuration, the attack scenarios about these retailer stores were the same, i.e., the more than 200 anti-goals were refined in the same way. Thus, as shown in Fig. 5.4, we only focused on the first sub-goal *G2* in later analysis.

3. We identified exploitable threats to the assets. According to the domain knowledge that the information of credit cards which were used in store A (i.e., *CCA*) was kept

in the *store server A*, *G2* was refined to *G5*, targeting the *store server A*. In addition, we refined *G1* to *G4* because the entire set of credit card information as a whole was stored in the company database. Note that we will skip the illustration of this branch in the later analysis steps, which can be found in the full model (Fig. 5.5).

4. As the asset and the exploitable target of *G5* were not the same object, we elaborated *G5* to identify which threats should be imposed to the *store server A* in order to *disclose* the *CCA*. According to the threat knowledge, *G5* was and-refined to *G6* and *G7*, which were intended to reach the server and to access into the server, respectively.
5. As *G6* and *G7* introduced a new asset *store server A*, iterative analysis was performed to these two goals from the secondly step until the analysis no longer introduced new assets. As shown in Fig. 5.4, the longest refinement paths {*G6, G10, G15, G17, G18, G19*} iterated such analysis three times.
6. After identifying an exploitable target, we checked whether there were security mechanisms that were applied to protect the target. Take *G21* for example, it was refined to *G22* as an *access control mechanism* was applied to protect the *store server A*. Because *G22* promoted the *access control mechanism* as a new asset, another round of analysis started from the second step.
7. Performing the iterative analysis on *G13* and *G22* resulted in the anti-goals *G14* and *G23*, respectively. As the iterative analysis did not introduced new assets, the anti-goal refinement reached an end.

Results and analysis. The application of our approach finally resulted in an anti-goal model with 46 anti-goals and 48 refinements (Fig. 5.5), which took me 5 hours to build (***RQ1.1***). In particular, during the application of our approach, we noticed several issues. Firstly, the retrieval and application of domain information was time-consuming, which may raise the scalability problem when applying our approach to large-scale and complicated scenarios (***RQ1.2***). In our study, we extracted domain information from the scenario specification, which was then used by different anti-goal refinement methods. As we did not formalize such information, we had to retrieve and apply it manually, which was time-consuming. Therefore, in the future, we plan to capture (part of) the required domain information in our three-layer requirements goal model, allowing us to automatically retrieve corresponding information when refining anti-goals.

Secondly, we realized that the manual sanity check was important for pruning the model, relieving the scalability problem. As discovered in our examples, there were cases

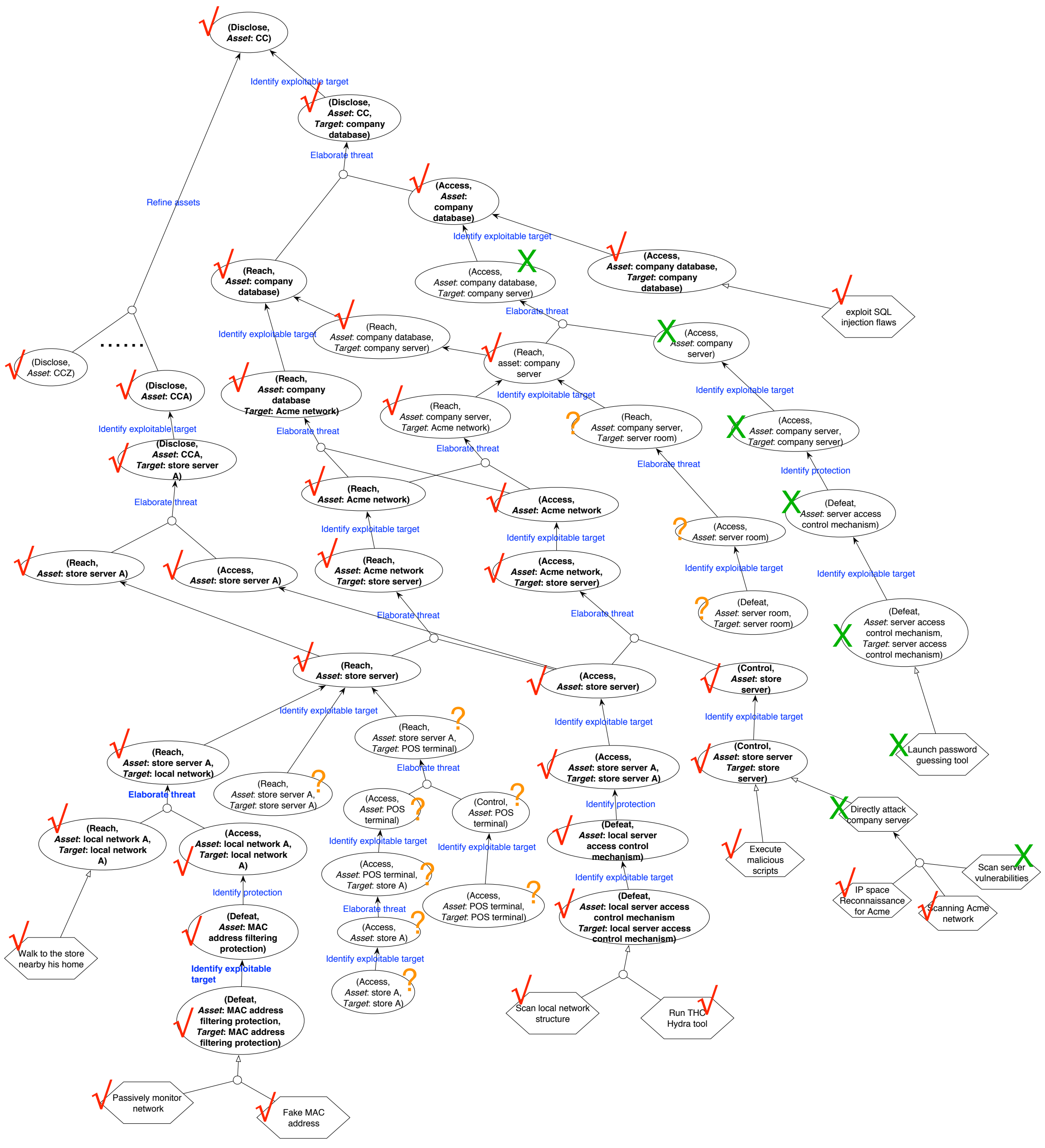

(Disclose, Asset: CC)
Identify exploitable target
(Disclose, Asset: CC, Target: company database)
Elaborate threat
Refine assets
(Access, Asset: company database)
Identify exploitable target
(Reach, Asset: company database)
(Access, Asset: company database, Target: company server)
(Access, Asset: company database, Target: company database)
exploit SQL injection flaws
Elaborate threat
(Disclose, Asset: CCZ)
......
(Disclose, Asset: CCA)
Identify exploitable target
(Reach, Asset: company database, Target: company server)
(Reach, asset: company server)
(Access, Asset: company server)
(Reach, Asset: company database Target: Acme network)
(Reach, Asset: company server, Target: Acme network)
Identify exploitable target
(Reach, Asset: company server, Target: server room)
(Access, Asset: company server, Target: company server)
Identify exploitable target
Identify exploitable target
(Disclose, Asset: CCA, Target: store server A)
Elaborate threat
Elaborate threat
Elaborate threat
Identify protection
(Reach, Asset: Acme network)
(Access, Asset: Acme network)
(Defeat, Asset: server access control mechanism)
(Access, Asset: server room)
Identify exploitable target
Identify exploitable target
Identify exploitable target
(Reach, Asset: store server A)
(Access, Asset: store server A)
(Reach, Asset: Acme network Target: store server)
(Access, Asset: Acme network, Target: store server)
Identify exploitable target
(Defeat, Asset: server room, Target: server room)
(Defeat, Asset: server access control mechanism, Target: server access control mechanism)
Elaborate threat
Elaborate threat
(Reach, Asset: store server)
(Access, Asset: store server)
(Control, Asset: store server)
Identify exploitable target
Identify exploitable target
Identify exploitable target
(Reach, Asset: store server A, Target: POS terminal)
(Reach, Asset: store server A, Target: local network)
(Reach, Asset: store server A, Target: store server A)
Elaborate threat
Elaborate threat
(Access, Asset: store server A, Target: store server A)
(Control, Asset: store server Target: store server)
Launch password guessing tool
(Access, Asset: POS terminal)
(Control, Asset: POS terminal)
Identify protection
Directly attack company server
(Reach, Asset: local network A, Target: local network A)
(Access, Asset: local network A, Target: local network A)
Identify exploitable target
Identify exploitable target
(Defeat, Asset: local server access control mechanism)
Execute malicious scripts
(Access, Asset: POS terminal, Target: store A)
Identify exploitable target
Scan server vulnerabilities
Identify protection
(Access, Asset: POS terminal, Target: POS terminal)
IP space Reconnaissance for Acme
Scanning Acme network
Elaborate threat
(Defeat, Asset: local server access control mechanism Target: local server access control mechanism)
Walk to the store nearby his home
(Defeat, Asset: MAC address filtering protection)
(Access, Asset: store A)
Identify exploitable target
Identify exploitable target
(Access, Asset: store A, Target: store A)
Scan local network structure
Run THC Hydra tool
(Defeat, Asset: MAC address filtering protection, Target: MAC address filtering protection)
Passively monitor network
Fake MAC address

Figure 5.5: Apply the attacker strategy to the credit card stolen scenario (full)

that one asset can be refined into different but homogeneous sub-parts (i.e., the entire set of credit card information consists of credit card information from different retailers, and all the retailers were assumed to have the same IT infrastructure in this scenario). Thus, each of these sub-parts was targeted by a particular anti-goal, and all such anti-goals can actually be attacked using the same attack strategies. In other words, the refinement analysis performed over such anti-goals resulted in the same refinement structure, and there was no need to perform repeated analysis over each of these anti-goals; As such, we only focused on one of those anti-goals, and reused the results for all other goals. We argue that such a decision has to be manually made during the sanity check, as it is very difficult to automatically determine whether the sub-parts of an assets are homogeneous.

Thirdly, we noticed that it was possible to obtain repeated anti-goals from different refinement branches, which needed to be merged in order to reduce the scalability problem (details about this situation will be further discussed in the next section). However, such checks were performed manually in our study, which was time-consuming and error-prone (***RQ1.2***). As for future work, we plan to automate such checks and thus further improve our approach.

By analyzing the and/or refinement operators, we identified that the final model contains a total of 11 alternative attack scenarios. To evaluate the effectiveness of the proposed approach, we carried out a bottom-up analysis to check whether the realistic credit card theft scenarios (documented in the book) can be covered by the attack scenarios that were identified by our approach. Specifically, we identified all specific attack actions that are performed by the attacker based the scenario description, including both successful and failed attack actions. Then, we checked whether the intention of these actions can be matched with the leaf goals in the anti-goal model. Our examination turned out that all the attack actions documented in the scenario can be associated with the leaf goals, i.e., the identified attack scenarios completely covered the real attack scenarios (***RQ2.1***). Specifically, among all the 11 attack scenarios discovered by our approach, 6 of them were reported in the scenario description (2 successful, 4 failed), while the other 5 potential attack scenarios were not mentioned in the scenario description, revealing previously unconsidered attacks (***RQ2.2***).

Overall, considering the time and encountered difficulties, we argue our approach can be easily applied to a middle-scale scenario (***RQ1***). In the meantime, we have identified a number of issues that need to be improved in our future work. In addition, based on the comparison between our analysis results and the realistic attack scenario reported in the textbook, we contend that our approach can effectively identify realistic attack strategies (***RQ2***).

Threats to validity of the evaluation. A major threat to the conclusion validity is that the study was only applied to one middle-scale scenario. In the future, we need to evaluate our approach with more real attack scenarios. In addition, the evaluation was performed by one of the method designers (i.e., myself), imposing a threat to the external validity. As such the conclusion we have made before (i.e., the approach is easy to apply) is based on a *strong* assumption that the practitioner has already gained a full understanding of our approach. As such, we need to further evaluate our approach with practitioners who were not involved in the method development. Finally, as the entire evaluation is performed by only one person (i.e., myself), there is threat to reliability. To tackle such threat, we need to involve more people in our future evaluation.

5.2.4 Discussion

Diversity of anti-goal refinement. As our proposal is based on the examination of three real attack scenarios that come from the same book [Mitnick and Simon, 2011], and we acknowledge that examining other scenarios from different sources may have different results. In addition, the examination process reflects our specific interpretation of attack strategies, and the outcome of the examination can vary from person to person. Consequently, we acknowledge that the proposed anti-goal refinement framework is a particular way of refining anti-goals, while there can be other ways for performing such analysis. Based on the study we have performed over a middle-size credit card theft scenario, we argue our approach is effective to analyze attacker's malicious intentions. Moreover, we will continue to evaluate this method as part of our entire security analysis framework.

Security knowledge sources. We analyze attack strategies by examining three real attack scenarios that are documented in a security textbook [Mitnick and Simon, 2011]. Apart from this book, we have found other potential security knowledge sources. Attack patterns were first proposed by Moore et al. [2001] to summarize reusable attack knowledge from repeated attacks in support of system security analysis. In particular, CAPEC (Common Attack Pattern Enumeration and Classification) is a comprehensive attack pattern repository, which was first released in 2007 and has accumulated 504 attack patterns [Barnum and Sethi, 2007]. However, these attack patterns indeed describe low-level attack knowledge about how to use specific attack techniques and tools to perform a particular attack, such as "exploit user-controllable input to perform a format string injection". Thus, CAPEC attack patterns do not fit our need of analyzing high-level attack strategies in this paper, but they can be used to operationalize anti-goals and support security analysis, which will be presented in Section 5.3. Another security threat knowledge source is the STRIDE threat trees [Shostack, 2014], which focuses on how one threat can be refined into other threats. However, these threat trees only capture a single step of threat elaboration and cannot account for multistage attacks. As a result, we do not examine these threat trees for analyzing attack strategies, but use them as an external security knowledge source to support the threat elaboration analysis in our anti-goal refinement framework.

Scalability. Our framework is designed to provide a comprehensive anti-goal refinement analysis, i.e., covering all potential attack scenarios. As such, the scalability issues are raised due to the large refinement space. To deal with this problem, we have proposed sanity checks for each analysis step in Section 5.2.2, in order to prune the model as part of its construction. In addition to the checks, we also observe a further phenomenon which helps to mitigate scalability. During the anti-goal refinement, it is possible to obtain repeated anti-goals, i.e., different anti-goals can be refined into the same anti-goals. This is because one anti-goal can have different influences, e.g., accessing to the server of a retailer store not only discloses credit card information stored in that server but also enables the attacker to penetrate the company internal network. As such, it is important to detect and merge the repeated anti-goals during the anti-goal analysis as new anti-goals are generated. Otherwise, the repeated anti-goals will be further refined separately and the size of the model can grow exponentially. Note that merging repeated anti-goals is performed by adding all refinement links of these anti-goals to one anti-goal and removing other anti-goals. As such, the derived model is not a tree but a directed acyclic graph (DAG).

Thus far our attack strategy analysis is performed manually, and we are planning to extend our prototype tool in order to semi-automate such analysis, especially reducing the efforts of graphical modeling. We intend to define formal inference rules based on the five anti-goal refinement patterns that are proposed in Section 5.2.1. By executing those rules, the tool will be able to automatically perform anti-goal refinement in a step-by-step manner. To guarantee the correctness of the analysis and to reduce model complexity, the tool should interact with analysts in order to support manual revision after each analysis step, allowing the analyst to, for example, perform sanity checks over the refinements.

5.3 Operationalizing Anti-Goals with Attack Patterns

In this section, we present a systematic approach which can efficiently operationalize the identified attack strategies using CAPEC attack patterns. Based on the operationalization results, we can automatically identify alternative (multistage) attacks on the target system. To this end, we first describe a systematic method to model CAPEC attack patterns as contextual goal models. The benefits of constructing contextual goal models are twofold: firstly, we are able to semi-automatically check attack context and select suitable attack patterns to operationalize attack strategies with the support of our prototype tool (which will be detailed in Chapter 8). Secondly, the contextual goal models can be seamlessly integrated with the goal-based attack model, as they share the same core constructs (e.g., goals, tasks, and domain assumptions). We have pragmatically applied this method

Table 5.3: Pattern-related conceptual mappings

Primary Pattern Concept	**Attack Pattern Attribute**	**Goal Model Element**
Context	Attack Prerequisites; Technical Context	Context; Domain Assumption
Problem	Attack Motivation-Consequences; Domains of Attack	Goals
Solution	Attack Execution Flow	Tasks

to model 102 CAPEC attack patterns, which can be reused for operationalizing attack strategies. Built on such patterns, we then define a systematic process and a collection of formal inference rules to efficiently select attack patterns for operationalizing attack strategies in terms of practical attacks.

5.3.1 Model Attack Patterns as Contextual Goal Model

A pattern consists of three primary pattern concepts: *Context*, *Problem*, and *Solution* [Alexander et al., 1977], which specify a proven solution can be applied to solve a problem in certain context. Attack patterns, as a specific type of pattern, are specified in the same spirit, but from an attacker's viewpoint, i.e., what an attacker wants to attack (problem), how does the attacker perform the attack (solution). Based on the semantics of attack pattern attributes[3], we have identified relevant attack pattern attributes that specify *problem, context, solution*, and indirectly map them to the contextual goal model elements, as shown in Table 5.3.

On the basis of these mappings, we propose a systematic method to model attack patterns as contextual goal models. Specifically, we take the attack pattern *CAPEC-66: SQL Injection* as an example to illustrate each modeling step in detail (shown in Fig. 5.6). The complete specification of this pattern can be found online[4].

It is worth noting that the method we proposed here is in line with the method for modeling security patterns (Section 6.2), but has been adjusted to accommodate specific concepts in attack patterns.

---

[3] A full CAPEC schema, `https://capec.mitre.org/data/xsd/ap_schema_v2.7.xsd`

[4] `https://capec.mitre.org/data/definitions/66.html`

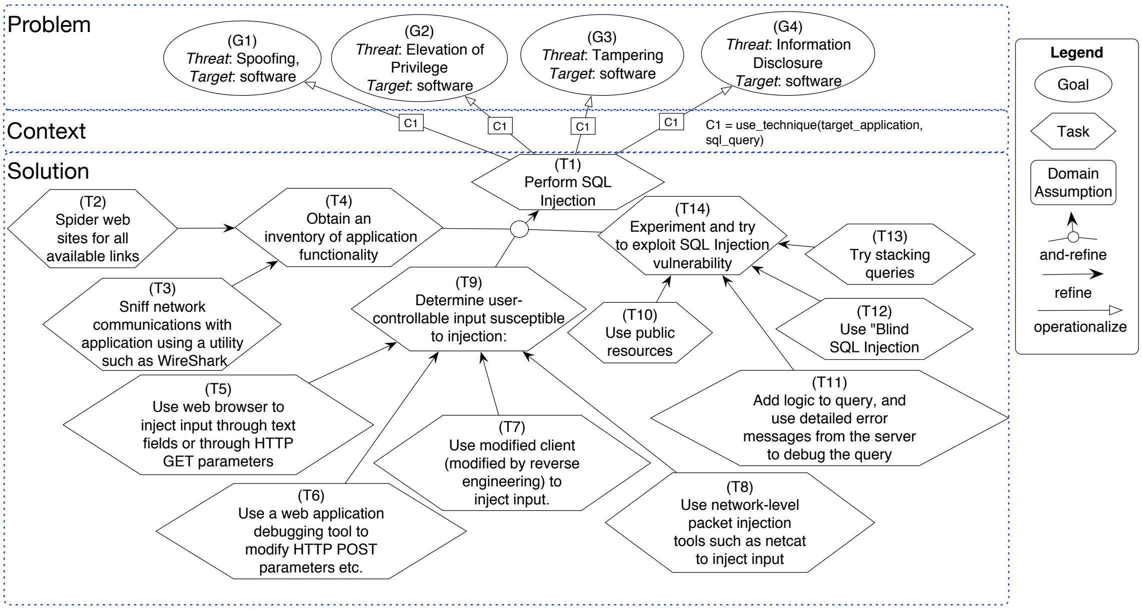

Problem
(G1) Threat: Spoofing, Target: software
(G2) Threat: Elevation of Privilege Target: software
(G3) Threat: Tampering Target: software
(G4) Threat: Information Disclosure Target: software
Context
C1
C1
C1
C1
C1 = use_technique(target_application, sql_query)
Solution
(T1) Perform SQL Injection
(T2) Spider web sites for all available links
(T4) Obtain an inventory of application functionality
(T14) Experiment and try to exploit SQL Injection vulnerability
(T13) Try stacking queries
(T3) Sniff network communications with application using a utility such as WireShark
(T9) Determine user-controllable input susceptible to injection:
(T10) Use public resources
(T12) Use "Blind SQL Injection
(T5) Use web browser to inject input through text fields or through HTTP GET parameters
(T11) Add logic to query, and use detailed error messages from the server to debug the query
(T7) Use modified client (modified by reverse engineering) to inject input.
(T6) Use a web application debugging tool to modify HTTP POST parameters etc.
(T8) Use network-level packet injection tools such as netcat to inject input
Legend
Goal
Task
Domain Assumption
and-refine
refine
operationalize

Figure 5.6: An example attack pattern model

Modeling Attack Pattern Problems

A *Problem* solved by an attack pattern is actually the malicious intention that an attacker wants to achieve, which will be modeled as a *Goal* in the goal model. Specifically, we identify such malicious intentions from the attack pattern attributes *Attack Motivation-Consequences* and *Domains of Attack*.

To seamlessly integrate the attack pattern analysis with previous attack strategy analysis, we specify the malicious intention of each attack pattern using structured anti-goals (which was introduced in Section 5.2.1). In particular, we focus on the *Threat* that is imposed by an attack pattern and the *Target* that is exploited by an attack pattern, which are important for automatic pattern matching (which will be described in the next subsection). The threat information is elicited based on the attack impact specified in *Attack Motivation-Consequences*. As the CAPEC schema enumerates a total of 18 types of attack impacts, we map these impacts to STRIDE threat categories (Table 5.4), using these categories as the *Threat* attribute in the resulting anti-goal. We elicit the target information with respect to *Domains of Attack* of an attack pattern. In particular, the CAPEC schema includes six specific domains, and we identify typical attack targets for each of these domains, as shown in Table 5.5. In the attack pattern example (Fig. 5.6), the pattern has an impact "*Read application data*", which is mapped into the threat *Information Disclosure* as modeled in the anti-goal *G4*. In addition, as this pattern is under the software domain, we specify the *Target* attribute of *G4* as *software*.

Table 5.4: Mappings between attack pattern impact and STRIDE threats categories

STRIDE	**Attack Impact**
Information Disclosure	Read application data Read memory Read files or directories
Tampering	Modify application data Modify application data memory Modify application data files or directories Unexpected states Alter execution logic
Denial of Service	DoS: instability DoS: resource consumption (CPU) DoS: resource consumption (memory) DoS: crash / exit / restart DoS: amplification DoS: resource consumption (other)
Elevation of Privilege	Gain privileges / assume identity Execute unauthorized code or commands Bypass protection mechanism
Spoofing	Gain privileges / assume identity
Repudiation	Hide Activities

Table 5.5: Typical targets of attack domains

Attack Domain	**Target**
Social Engineering	People
Supply Chain	Software; Hardware
Communication	People; Software; Hardware
Software	Software
Physical Security	Hardware
Hardware	Hardware

Modeling Attack Pattern Context

The context of an attack pattern specifies under which situation the attack can be applied to achieve an attacker's malicious intention. Thus, we model such context and associate it with the operationalization link between goals and tasks in the goal model, where the goals can only be operationalized into the tasks if the context holds (e.g., context *C1* shown in Fig. 5.6).

We obtain the context information of an attack pattern by looking at the attack pattern attributes *Attack Prerequisites* and *Technical Context*. As the context information is specified in natural language, analysts have to manually check such context during the application of attack patterns. After reviewing the context information of 102 attack patterns, we propose a set of formal predicates for specifying context, which can be automatically checked against the three-layer requirements goal model (Section 4.2). Such predicates include *protected_by*, *communicate*, *use_technique*, *use_data_from*, and *accept _user_input*, detailed reasoning with these predicates will be presented in Section 5.3.2. It is worth noting that the proposed predicates do not express contexts that cannot be captured and checked in the three-layer requirements goal model. Such contexts are normally too detailed to capture in the domain model, for example, "*The targeted application runs with elevated OS privileges*". These cases will require manual analyst intervention via an interactive pop-up in the tool.

In our example, the *SQL Injection* pattern has an attack prerequisite "*SQL queries used by the application to store, retrieve or modify data*", which is then formalized using the predicate *use_technique* as shown in Fig. 5.6. Note that if there are several pieces of context information, they are specified in a conjunctive way, i.e., all of them have to hold in order to apply the corresponding attack pattern.

Modeling Attack Pattern Solutions

Solutions of attack patterns are specific attack actions that are performed by attackers using concrete attack techniques. We elicit such information from the attack pattern attribute *Attack Execution Flow*, and model each attack action as a task in the goal model. In particular, we focus on capturing the alternative attack actions for implementing the attack.

When modeling the solution section of the pattern, we first create a general task to summarize the overall attack that achieves the previously modeled anti-goals, such as *T1: Perform SQL Injection* shown in Fig. 5.6. The *Attack Execution Flow* is specified in terms of a sequence of attack steps that are required to fulfill the attack, thus, we capture this information as sub-tasks, and-refining the general task. In our example, the tasks *T4*, *T9*, and *T14* are individual attack steps specified in the *Attack Execution Flow*. Moreover, within each attack step, the attack pattern also describes alternative attack techniques that can be used for performing the attack step. Thus, we model each of these techniques as a refinement to the corresponding attack step. For example, in Fig. 5.6, the tasks *T2* and *T3* present two alternative attack techniques that can be applied to perform task *T4*. Note that when specifying the tasks, we reuse the original description provided by attack patterns, maintaining their security expertise in the model.

Applying the Modeling Method

I have spent three person-days pragmatically applying the proposed method to model 102 (out of 504) attack patterns. In particular, during the modeling practice, I noticed that this modeling task requires modelers to first thoroughly understand the rationale of the pattern to be modeled. On average, each pattern costs me 10-20 minutes to model, depending on the complexity of the pattern. The obtained models can be (re-)used to operationalize attack strategies in a semi-automatic way, using our prototype tool (Chapter 8). A full list of modeled attack patterns can be found in Appendix C.

These attack patterns are selected under the following criteria: first, the select patterns should cover all attack pattern domains (as shown in Table 5.5) in order to assist our attack analysis with comprehensive attack knowledge. Secondly, the pattern specifications need to be complete, i.e., all the attack pattern attributes that are required to build the contextual goal model should be well documented. In particular, each CAPEC pattern has been specified with an attribute *Completeness*, valuing from *Hook*, *Stub*, to *Complete*, and thus we focus on patterns that have complete specification.

It is worth noting that most attack patterns under the *Social Engineering* and *Physical* domains have only incomplete specifications. In order to preserve the comprehensive coverage of the selected patterns, instead of dropping all such incomplete patterns, we identify the required pattern attributes in accordance with the overall description of those patterns. As such, the constructed models for these patterns are comparatively simple. In some extreme cases, an attack pattern model may only consist of one goal with task operationalizes the goal. Such phenomena, on one hand, disclose the needs for the pattern community to further develop the incomplete patterns. On the other hand, as the analysis results of our approach are affected by such attack patterns, we should keep revising the established models in accordance with the recent advances in the field of attack patterns.

In the CAPEC repository, each attack pattern has been documented with related patterns that are more abstract or more detailed to it, using $ChildOf$ relations. We also capture such relations among the modeled 102 patterns, establishing pattern hierarchies, which can help us to reduce the complexity of the attack operationalization analysis. Such attack pattern hierarchies are presented in Appendix B.

5.3.2 Attack Pattern Selection and Application

In this section, we present a tool-supported systematic process for operationalizing attack strategies and eventually generating a collection of realistic attack alternatives that can satisfy an attacker's root anti-goal. The overall attack operationalization process is shown in Fig. 5.7, each analysis step is detailed below. In particular, this analysis process takes the attack strategy model (i.e., the anti-goal model generated in Section 5.2) as input, while eventually produces a set of alternative attacks that are specified in text. It is worth noting that all inference rules defined for the the attack pattern analysis (e.g., Table 5.6) have been implemented by our prototype tool (Chapter 8), thus can be automatically inferred.

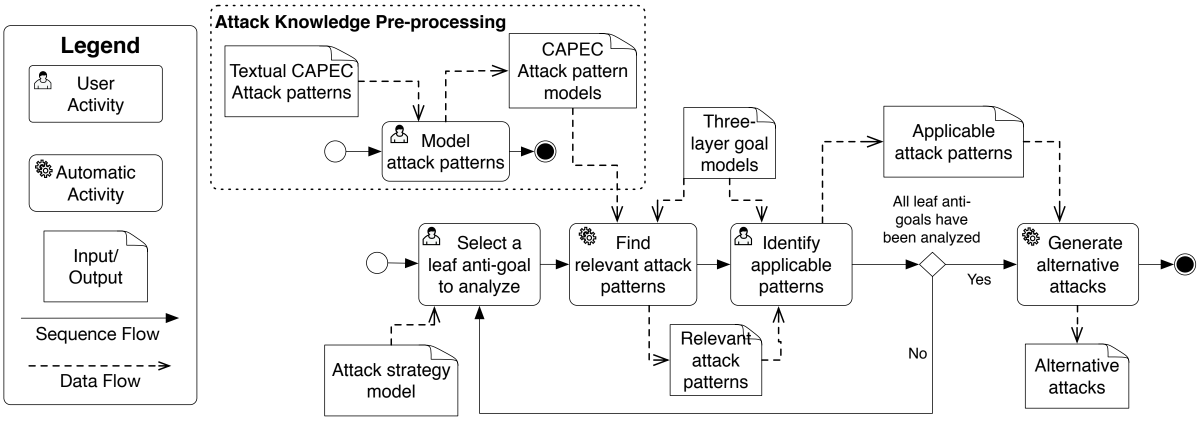

Legend
User Activity
Automatic Activity
Input/ Output
Sequence Flow
Data Flow
Attack Knowledge Pre-processing
Textual CAPEC Attack patterns
CAPEC Attack pattern models
Model attack patterns
Three-layer goal models
Applicable attack patterns
All leaf anti-goals have been analyzed
Select a leaf anti-goal to analyze
Find relevant attack patterns
Identify applicable patterns
Yes
No
Generate alternative attacks
Attack strategy model
Relevant attack patterns
Alternative attacks

Figure 5.7: A systematic process for attack strategy operationalization

Select A Leaf Anti-Goal

The operationalization analysis takes the attack strategy model as an input, which is obtained from our attack strategy analysis (as presented in Section 5.2). An example of the attack strategy model is shown at the top of Fig. 5.8, which specifies what an attacker intends to attack and why. To operationalize different attack strategies, we need to iteratively perform operationalization analysis for each leaf anti-goal in the attack strategy model (i.e., *G8-G11*). As highlighted in Fig. 5.8, we select anti-goal *G10* for illustration.

Find Relevant Attack Patterns

To select appropriate attack patterns that operationalize the given anti-goal, we first identify all relevant attack patterns according to the *problem* we have modeled for each attack pattern. Since the problem of an attack pattern has also been modeled as structured anti-goals, we can identify relevant patterns by matching the threat and target specified in the structured anti-goals. Such match is automated using the inference rule *REV* (Table 5.6): given an anti-goal *G1* which imposes a threat *TH* to a target *TA1*, if an attack pattern *AP* has an anti-goal *G2* (as its problem), where *G2* imposes the same threat *TH* to a category of target *TA2* that *TA1* belongs to, then the attack pattern *AP* is relevant to the anti-goal. For example, as shown in Fig. 5.8, we identify CAPEC pattern *SQL Injection* is relevant to *G10*, as *G10* can be matched with the problem *G2* of *SQL Injection* according to rule *REV*. Similarly, the other two patterns *CAPEC-186 Malicious Software Update* and *CAPEC-100 Overflow Buffers* are also identified as relevant to the anti-goal *G10*. Note that, in Fig. 5.8, we use pentagons to represent collapsed attack pattern models in order to provide a better overview of the operationalization analysis (an excerpt of an uncollapsed pattern model is indicated in the bottom left corner).

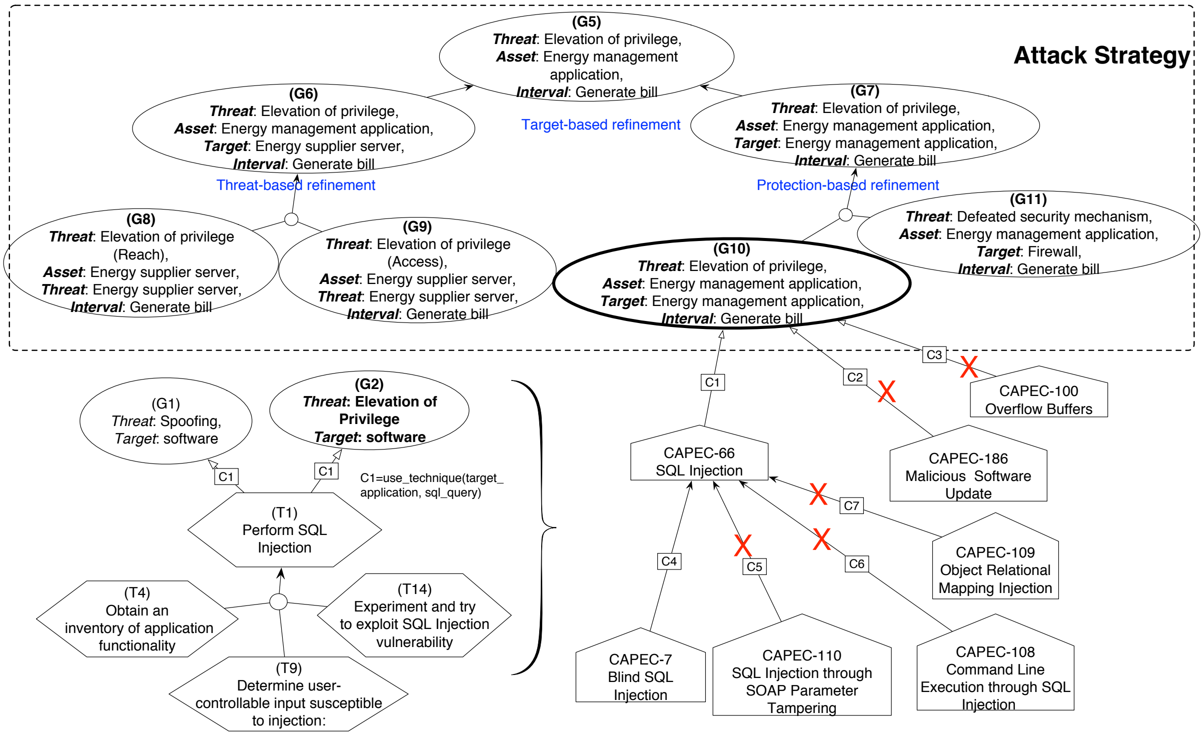

Attack Strategy
(G5) Threat: Elevation of privilege, Asset: Energy management application, Interval: Generate bill
(G6) Threat: Elevation of privilege, Asset: Energy management application, Target: Energy supplier server, Interval: Generate bill
Target-based refinement
(G7) Threat: Elevation of privilege, Asset: Energy management application, Target: Energy management application, Interval: Generate bill
Threat-based refinement
Protection-based refinement
(G8) Threat: Elevation of privilege (Reach), Asset: Energy supplier server, Threat: Energy supplier server, Interval: Generate bill
(G9) Threat: Elevation of privilege (Access), Asset: Energy supplier server, Threat: Energy supplier server, Interval: Generate bill
(G10) Threat: Elevation of privilege, Asset: Energy management application, Target: Energy management application, Interval: Generate bill
(G11) Threat: Defeated security mechanism, Asset: Energy management application, Target: Firewall, Interval: Generate bill
(G1) Threat: Spoofing, Target: software
(G2) Threat: Elevation of Privilege Target: software
C1
C1
C1=use_technique(target_application, sql_query)
(T1) Perform SQL Injection
(T4) Obtain an inventory of application functionality
(T14) Experiment and try to exploit SQL Injection vulnerability
(T9) Determine user-controllable input susceptible to injection:
C1
C2
C3
CAPEC-100 Overflow Buffers
CAPEC-66 SQL Injection
CAPEC-186 Malicious Software Update
C7
CAPEC-109 Object Relational Mapping Injection
C4
C5
C6
CAPEC-7 Blind SQL Injection
CAPEC-110 SQL Injection through SOAP Parameter Tampering
CAPEC-108 Command Line Execution through SQL Injection

Figure 5.8: Operationalize a leaf anti-goal using attack patterns

When performing the attack pattern selection analysis, we take into account the pattern hierarchies in order to select the most appropriate patterns. For example, as the *SQL Injection* pattern has four children patterns which are also relevant to *G10*, we model them as refinements of *SQL Injection*, as shown in Fig. 5.8. According to such a hierarchy, a pattern and its ascendants represent only one operationalization alternative rather than

Table 5.6: Inference rules for attack operationalization

REV	$relevant\_to(AP, G1) \leftarrow has\_threat(G1, TH) \wedge has\_target(G1, TA1) \wedge has(AP, G2) \wedge has\_threat(G2, TH) \wedge has\_target(G2, TA2) \wedge isa(TA1, TA2)$
$CR1$	$communicate(A, B) \leftarrow depend(A, \_, B)$
$CR2$	$communicate(A, B) \leftarrow depend(B, \_, A)$
$CR3$	$use\_technique(A, R) \leftarrow has(A, R) \wedge resource(R)$
$CR4$	$use\_data\_from(A, B) \leftarrow depend(A, R, B) \wedge data(R)$
$CR5$	$accept\_user\_input(A) \leftarrow depend(A, R, B) \wedge data(R) \wedge human(B)$
$CR6$	$protected\_by(A, SM) \leftarrow sec\_goal(SG) \wedge has\_asset(SG, A) \wedge operationalize(SM, SG)$
APP	$applicable\_to(sql\_injection, G) \leftarrow relevant\_to(sql\_injection, G) \wedge has\_target(G, TA) \wedge use\_technique(TA, sql\_query)$
$ALT1$	$achieved(G1) \vee ... \vee achieved(Gn) \leftarrow refine(\{G1...Gn\}, G0) \wedge achieved(G0)$
$ALT2$	$achieved(G1) \leftarrow and\_refine(G1, G0) \wedge achieved(G0)$

multiple alternatives.

Identify Applicable Attack Patterns

After finding attack patterns that are relevant to an anti-goal, we further check their context to determine whether these patterns are applicable in current system context. We import the three-layer requirements goal model as the domain model that captures system context, against which we can automatically check the attack pattern context. To this end, we have defined a number of inference rules that specify the implication relation between the formal context predicates and the goal model predicates, which are shown as $CR1$-6 in Table 5.6. In particular, the formal context predicates are put in the left hand side (i.e., the head of the rule), while the right hand side (i.e., the body of the rule) presents the corresponding facts in the three-layer requirements goal model. For instance, rule $CR1$ and $CR2$ express that if there is a dependency relation between two actors, it implies the context that the two actors are communicating with each other. Detailed information about the formal predicates of the three-layer goal model can be found in Section 4.2. Overall, given a three-layer goal model, we can automatically apply the above context rules to infer system context.

On top of these context check rules (i.e., $CR1$-6), we have defined specific applicability rules for each attack pattern, as different patterns require different contexts. For example, the rule APP shown in Table 5.6 is specifically defined for the pattern *SQL Injection*.

This rule says if the *sql_injection* pattern has been identified as relevant to an anti-goal G, and the target of G uses the technique *sql_query*, then the *sql_injection* pattern is applicable to operationalize G. The entire set of pattern-specific applicability rules, which we have defined for all the 102 attack patterns, can be found in Appendix D. All these inference rules can be automatically inferred to assess the applicability of attack patterns.

It is worth noting that the context check rules (i.e., CR1-6) only apply to context that can be captured in by our three-layer requirements goal model (i.e., the domain model). For contexts that cannot be captured in the domain model, we define specific predicates for each of them. In particular, such predicates are manually checked using *question* rules, which let our prototype tool interact with analysts for checking the corresponding context. For example attack pattern *CAPEC-10: Buffer Overflow via Environment Variables* has a context "*The application uses environment variables*", which cannot be captured in our three-layer requirements goal model. Thus, we describe such context the predicate *use_detailed_technique*, and define a corresponding *question* rule as below:

$$\begin{aligned}
&question(use\_detailed\_technique, TA, environment\_variables) \leftarrow \\
&\qquad relevant\_to(capec\_10, AG), has\_target(AG, TA), \\
&\qquad not\ use\_detailed\_technique(TA, environment\_variables), \\
&\qquad not\ no\_use\_detailed\_technique(TA, environment\_variables)
\end{aligned}$$

The above rule means that when the attack pattern *capec*_10 (i.e., Buffer overflow via environment variables) is relevant to an anti-goal AG which targets TA, if there is neither positive nor negative evidence about TA uses environment variables, then a question will be presented to analysts for checking that context. Based on the answers from analysts, our prototype tool will automatically update context specification using corresponding predicates (i.e., *use_detailed_technique* and *no_use_detailed_technique* in this case). As such, the same context does not need further check, i.e., one such context only needs to be manually check once. Overall, with the support of our prototype tool, analysts are able to semi-automatically identify applicable attack patterns.

When identifying the applicable pattern, we also consider the hierarchy among patterns. As "parent" patterns focus on a more abstract problem, concerning a more general context, we first check the parent patterns. If a parent pattern is identified as inapplicable, then all its children patterns will be identified as inapplicable without additional checking; if a parent pattern is applicable, then we will further check each of its children patterns. For example, as shown in Fig. 5.8, we first check the context of *SQL Injection, Malicious Software Update, Overflow Buffers*, (i.e., *C1, C2, C3*), which turns out that only *SQL Injection* is applicable (i.e., *C1* holds). Then, we further check the context of children patterns of *SQL Injection*, and identify that only pattern *Blind SQL Injection* is applicable.

Generate Alternative Attacks

Once identifying all applicable attack patterns to an anti-goal, we unfold the collapsed applicable attack patterns (i.e., the pentagon notations in Fig. 5.8) and show detailed attack behaviors (i.e., the solution part of an attack pattern model in Fig. 5.6). As such, we complete the entire attack model, including both the attack strategies and attack behaviors.

Once the entire attack model is obtained, we want to answer the question "*Is the root anti-goal achievable?*", "*If so, how many different combinations of attack behaviors can be performed to achieve the goal?*". To this end, we define inference rules to exhaustively explore the space of alternatives, which has been implemented in Disjunctive Datalog and can be inferred by our prototype tool using the DLV inference engine[5]. As shown in Table 5.6, the rule $ALT1$ means if a goal $G0$ is alternatively refined by sub-goals $G1...Gn$, then the achievement of **each** sub-goal serves as an alternative to achieve $G0$. On the other hand, if a goal $G0$ is and-refined by sub-goals $G1...Gn$, then the achievement of **all** the sub-goals is required to achieve $G0$, i.e., no more alternatives are introduced. This rationale is implemented as the rule $ALT2$. By applying these two rules to the root anti-goal of the attack model, we are able to obtain all the alternative attacks.

All the identified alternative attacks will be specified in text, which will be used to assess the criticality of security goals as specified in our holistic security requirements analysis (Section 4.3.2). It is worth noting that before using the identified attacks for criticality analysis, risk assessments over such attacks are required in order to prioritize them. Specifically, only attacks that are at *high* risk to damage systems will be used to in the criticality analysis. As a result, the holistic attack analysis currently has not been fully integrated with the three-layer security requirements analysis, which will be further discussed in Chapter 10.

5.4 Validation

In this section, we focus on validating the entire holistic attack analysis approach, in particular, whether the approach can help analysts to effectively identify attacks. To this end, we applied it to a smart grid scenario, which was first described in Section 4.2.1 and will be presented in more details in Section 9.1. Since we have evaluated the attack

---

[5] `www.dlvsystem.com`

strategy analysis in Section 5.2.3, in this case study, we place an emphasis on the operationalization of attack strategies. In particular, we exclusively evaluate the effectiveness of the operationalization results, i.e., whether our approach is able to identify realistic attack actions. To this end, we reference to a comprehensive security analysis performed on the same smart grid case [Suleiman and Svetinovic, 2013], this study took a total of 16 person-months to identify threats, vulnerabilities and security requirements, the results of which have been evaluated as realistic and effective.

We took the three-layer requirements goal model of the smart grid scenario as the domain model, and applied our holistic attack analysis approach to identify alternative attacks on this scenario. Note that all models that have been built and analyzed during this study can be found in Appendix E. As some of these figures are too large (containing more than 1000 elements), we make the vector file of these models online[6].

Generate attack strategy. As the first part of our holistic attack framework, we identified alternative attack strategies following our systematic approach (presented in Section 5.2). In particular, we started from determining an anti-goal to analyze, which attacked the integrity of energy consumption data, i.e., {*Threat: Tampering, Asset: Energy consumption data, Interval: Real-time pricing is applied*}. Having this anti-goal as a root goal, we manually applied anti-goal refinement patterns based on the domain model to systematically refine the anti-goal from an attacker's viewpoint and generated alternative attack strategies. The resulting attack strategy model included 53 anti-goals, 39 refinement links, and 20 and-refinement link, implying 12 alternative attack strategies.

Operationalize attack strategy. Once alternative attack strategies were obtained, we operationalized them in terms of realistic attacks, using the attack pattern approach presented in Section 5.3. Using the 102 attack patterns we have modeled, we first automatically identified relevant attack patterns for all the 14 leaf anti-goals in the attack strategy model, resulting in 368 attack patterns in total. Each of the leaf anti-goals, on average, was matched with 26 relevant patterns. The full model is presented in Appendix E. All the identified relevant patterns were automatically organized in a hierarchical manner to facilitate the applicability analysis. After semi-automatically checking the context of all the relevant attack patterns, we derived 28 applicable attack patterns for all of the 14 anti-goals, covering social, software, and physical attacks.

Take Fig. 5.9 as an example, which shows an excerpt of the final attack model. A wide spectrum of attack patterns were identified, varying from social attacks (e.g., rogue integration procedures), to software attacks (e.g., rainbow table password cracking), to physical attack (e.g., lock picking). In particular, according to the attack model, we can identify alternative (multistage) attacks that achieve the root anti-goal, i.e., imposing

---

[6] `http://disi.unitn.it/~li/ap/validation.zip`

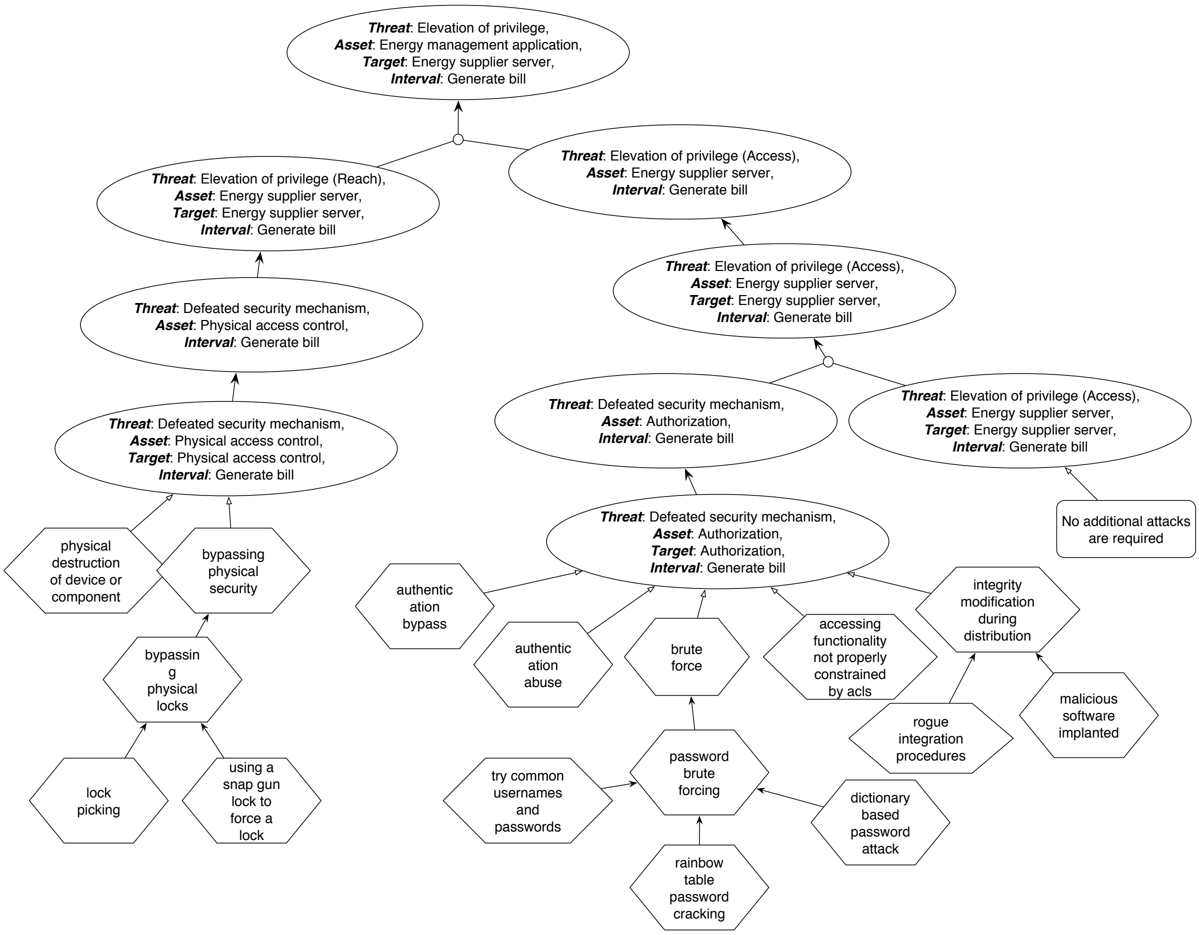

Threat: Elevation of privilege, Asset: Energy management application, Target: Energy supplier server, Interval: Generate bill
Threat: Elevation of privilege (Reach), Asset: Energy supplier server, Target: Energy supplier server, Interval: Generate bill
Threat: Elevation of privilege (Access), Asset: Energy supplier server, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Physical access control, Interval: Generate bill
Threat: Elevation of privilege (Access), Asset: Energy supplier server, Target: Energy supplier server, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Physical access control, Target: Physical access control, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Authorization, Interval: Generate bill
Threat: Elevation of privilege (Access), Asset: Energy supplier server, Target: Energy supplier server, Interval: Generate bill
No additional attacks are required
Threat: Defeated security mechanism, Asset: Authorization, Target: Authorization, Interval: Generate bill
physical destruction of device or component
bypassing physical security
authentication bypass
integrity modification during distribution
authentication abuse
brute force
accessing functionality not properly constrained by acls
bypassing physical locks
malicious software implanted
rogue integration procedures
try common usernames and passwords
password brute forcing
lock picking
using a snap gun lock to force a lock
dictionary based password attack
rainbow table password cracking

Figure 5.9: An excerpt of the final attack model

the *elevation of privilege* threat to the *energy management application.* For instance, an attacker can first perform the *lock picking* attack to reach the *energy supplier server* and then perform the *rainbow table password cracking* attack to gain access into the *energy supplier server.*

Based on the complete attack model, including both high-level attack strategies and realistic attacks behaviors, we automatically generate 108 realistic attack alternatives, each of which consists of one or multiple attacks. To validate the analysis results, we compared them with a comprehensive security analysis performed on the same smart grid case [Suleiman and Svetinovic, 2013], the results of which showed that our approach was able to effectively identify realistic. In particular, we found out that our identified attack alternatives can cover all the threats to the integrity of energy consumption data that are reported in [Suleiman and Svetinovic, 2013]. In addition, our results discovered detailed attack behaviors that can be performed by attackers, and showed how such attack actions were composed to form multistage attacks. For example, the comparison study discovered a high-level threat "*Tampering with SM's firmware*" (Table 4, T.5 in [Suleiman and Svetinovic, 2013]), while our results yielded a corresponding multistage attack which first performed *CAPEC-16: Dictionary-based Password Attack* (consisting of four detailed attack steps) to defeat the password-based authorization and gained access to the smart meter firmware, and then performed *CAPEC-186: Malicious Software Update* to tamper with the smart meter firmware.

Overall, we contend that our approach can be applied to effectively identify realistic attacks. In the meantime, we acknowledge that additional empirical studies should be performed to evaluate different aspects of this approach, such as usability and scalability.

5.5 Discussion and Related Work

Attack modeling and analysis techniques. Attack trees are a typical way of representing attack scenarios. Although there is no unique way of creating attack trees, different researchers have proposed their own ways to build attack trees, which are related to our anti-goal refinement framework. Morais et al. [2013] advocate to first build the overall attack, and then identify the violated security properties and the security mechanisms to be exploited, respectively, and finally model the concrete attack actions. Paul [2014] proposes a layer-per-layer approach to generate skeletons of attack trees using information comes from system architecture, risk assessment study, and related security knowledge base. However, these approaches do not capture the attacker's malicious intentions and cannot analyze attack strategies as we define them.

Apart from the attack trees, attack graphs are another way for representing attack scenarios. An attack graph shows all paths through a system that end in a state where an attacker achieves his malicious intentions. Phillips and Swiler [1998] first use attack graphs to analyze network security. Due to the homogeneous settings of machines in the network, the states of machines (i.e., nodes in the attack graph) and the atomic attacks on machines (i.e., transitions in the attack graph) can be enumerated. As such, it is possible to fully automate the generation of attack graphs using a comparatively simple attack strategy. Take the approach of Sheyner et al. [2002], for example: an attacker starts from a machine with the root permission, he then iteratively detects a new machine in the network, logs into that machine, and gets the root permission of that machine until reaching his target machine. In a recent study, Beckers et al. [2015] propose to apply the attack graph approach to analyze social engineering attacks, where the states of people are modeled as nodes and social engineering attacks are captured as transitions between nodes. However, the attack graph approach only applies to systems that have simple and homogeneous components, and is therefore inappropriate for security analysis of complex socio-technical systems that have heterogeneous components, such as people, software, and hardware.

Attacker-oriented analysis. Inspired by the *Art of War* philosophy, i.e.,"*Know your enemies and know yourself, you will not be imperiled in a hundred battles.*" [Tzu, 2011], several approaches have been proposed to model and analyze security attacks from an attacker's viewpoint. Lin et al. [2003b] capture the requirements of a malicious user that subverts an existing requirement as anti-requirements, which are incorporated into abuse frames to represent threats and analyze security requirements. Lamsweerde [2004] proposes to use anti-goals to model attacker's malicious intention, and then exploit alternative attacks by systematically refining such anti-goals. Sindre and Opdahl [2005] extend traditional use cases to cover misuse cases, which describe harmful behaviors to a system performed by adversaries . Building on the misuse cases, they propose a systematic process for eliciting security requirements. Elahi et al. [2010] extend goal models to model attacker templates which consist of malicious goals and tasks, based on which they assess system risks and identify countermeasures. All of theses approaches require attacker knowledge as input, based on which they can analyze the influences of attacks on systems and identify corresponding countermeasures. In particular, these approaches make a strong assumption about the availability of relevant knowledge, e.g., "*The proposed framework assumes that analysts have knowledge about vulnerabilities, potential attacks, and proper countermeasure or can obtain such information*" [Elahi et al., 2010].

We argue that performing the attack analysis from an attacker's viewpoint is a knowledge-intensive task, where the body of attack knowledge plays an important role. However, as pointed out by Souag et al. [2015], security knowledge is hard to acquire for software designers in reality. Without bridging the knowledge gap, the assumptions made in the above approaches become unrealistic, preventing the real adaption of those attack analysis approaches. Our approach tackles this challenge by building on realistic and reusable knowledge from existing attack knowledge repository, and can complement the above attacker-oriented analysis approaches. In particular, our approach identifies alternative attacks based on realistic attack knowledge, which can be used by those approaches to perform particular analysis, e.g., analyzing the impact of the attacks.

Attack pattern-based knowledge reuse. Moore et al. [2001] first emphasize the importance of reusing known attack knowledge, which significantly affects the practicality of attack analysis methods. Therefore, they define attack patterns, which encapsulate attack knowledge, in order to facilitate knowledge-intensive attack analysis. In particular, each pattern consists of four sections: goal, precondition, attack steps, and post-condition.

Other researchers have been inspired by Moore's work, and have defined various types of attack patterns. Gegick and Williams [2005] define software attack patterns in term of a sequence of events, using regular expressions. Specifically, each event is expressed by its associated component, such as user, server, hard disk, etc. By automatically matching such patterns with system design, the approach can assist analysts in identifying system vulnerabilities. Fernandez et al. [2009] specify attack patterns (i.e., misuse patterns) using POSA template [Buschmann et al., 2007]. The POSA template includes much more sections than the initial one defined by Moore et al. [2001], such as context, known uses, countermeasures, etc., which contribute to the practicality of attack pattern-based analysis. Although the above approaches contribute to the theoretical foundation of attack patterns, they have not been pragmatically applied to develop attack patterns. For example, Moore et al. [2001] illustrate their approach with four patterns, and we are unaware of subsequent work to develop further patterns; Fernandez has only developed three misuse patterns, as presented in his recent book[Fernandez-Buglioni, 2013].

Compared to the above theoretical approaches which focus on defining attack patterns, CAPEC emphasizes the pragmatic development of security patterns, which is initiated as a baseline catalog of attack patterns along with a comprehensive schema and classification taxonomy and has accumulated 504 attack patterns thus far. Since CAPEC provides a significant amount of practical security knowledge, it is receiving an increase in attention from both academia and industry. Thus, we choose CAPEC as the realistic attack knowledge source used in our approach.

One of the challenges of using the CAPEC repository is dealing with it's considerable size. Kaiya et al. [2014] define term-maps, which link terms in requirements specifications to specific security terms used in CAPEC, so as to automatically associate attack patterns to requirements specifications and further derive security requirements. Engebretson and Pauli [2009] enrich the CAPEC attack patterns with the concepts parent threat and parent mitigation in order to facilitate the navigation among the large number of attack patterns. Yuan et al. [2014] map CAPEC patterns to the STRIDE threat categories and develop a tool to facilitate the retrieval of CAPEC patterns based on such categories. However, all the above approaches do not check the applicability of attack patterns in terms of context required by the patterns. Thus, the patterns retrieved by these approaches may still include many non-applicable patterns, which need to be further checked by analysts to determine their applicability.

Our approach contributes to the context-based pattern selection by clearly modeling the *context*, *problems*, and *solutions* of each attack pattern in terms of contextual goal models, which can be semi-automatically analyzed based on domain models. As such, our proposal can help analysts to identify applicable attack patterns in a more effective manner. Apart from the retrieval and selection issues of CAPEC patterns, existing approaches only focus on reusing the knowledge of attack behaviors from the CAPEC patterns, without mining the intention of attacks. For example, Kim and Kim [2014] extract possible attack behaviors from CAPEC, and specify such behaviors using formal language in order to simulate attacks within specific system settings. However, their approach cannot analyze combined behaviors from different attack patterns, and thus can only detect a limited number of attacks. Our proposal not only analyzes the realistic attack behaviors but also concerns attacker's intention as described in Section 5.2. As such, by associating the CAPEC patterns with an attacker's high-level intention, we are able to capture multistage attacks which consist of several attack patterns, revealing a larger space of attacks.

Practical challenges in reusing attack patterns. Encapsulating knowledge as structural patterns is an effective way of reusing knowledge. Various patterns have been proposed to relieve knowledge-intensive analysis in different domains, such as requirements patterns, design patterns, security patterns, attack patterns, etc. Souag et al. [2015] survey reusable knowledge-based security requirements engineering approaches over the last 20 years, which shows that 9 out of 95 surveyed papers represent reusable knowledge in the form of patterns (other forms include catalogs, taxonomies, etc.). Although patterns can be reused in a comparatively easy manner, Araujo and Weiss [2002] have pointed out that analysts need first to have a thorough understanding of available patterns in order correctly select and apply them. This issue has been confirmed by us when applying our approach to the CAPEC patterns. When dealing with a small number of patterns, this issue will not be a challenge, as the analysts can afford the learning costs. However, we argue that such issue can impose practical challenges when analyzing a large number of patterns, e.g., the 504 CAPEC attack patterns. Facing this challenge, our approach formalizes the context of attack patterns and semi-automates the context-based attack pattern selection analysis, relieving analysts from scrutinizing the detailed context of all patterns. We have applied the approach to pragmatically process 102 (out of 504) attack patterns. Such processing must be performed only once, and the resulting models can be directly used by our prototype tool.

Security risk assessment. Our attack analysis approach exclusively addresses the challenge of holistically identifying (multistage) attacks in STSs. All the attacks that are identified by our approach can possibly happen to a system, however, we do not mean all of them have to be treated by certain security mechanism. Instead, according to the security requirements engineering process we have surveyed in Section 2.1.1, all the identified realistic attacks should be further assessed for risks. In particular, only the attacks that raise unacceptable risks will eventually be treated by corresponding security mechanisms. Currently, our holistic attack analysis cannot assess risks for the identified attacks. As a result, when practitioners leverage our approach to analyze security attacks, they are always encouraged to perform a risk assessment step afterwards. In addition, as a potential extension of our approach, we also intend to incorporate existing risk assessment approaches (e.g., CORAS [Lund et al., 2010] and OCTIVE [Alberts and Dorofee, 2002]) to accommodate such analysis.

5.6 Chapter Summary

In this chapter, we present a holistic attack analysis approach, which complements our holistic security requirements framework. In Section 5.1, we provide the overview of this approach, which takes security requirements and domain models (i.e., three-layer requirements goal models) as input and eventually produces a set of realistic (multistage) attacks for STSs, assisting analysts in determining critical security goals (as described in Section 4.3.2). In particular, we take an attacker's perspective to discover possible attacks, and divide our approach into two parts. Firstly, we aim to capture attacker's high-level attack strategies, which shed light on what and when an attacker wants to attack. To this end, we examine three real attack scenarios, based on which we propose an approach to systematically generate attack strategies (Section 5.2). Secondly, we investigate how to efficiently operationalize the identified attack strategies in terms of realistic attack actions, which is reported in Section 5.3. Specifically, we propose a systematic method to construct contextual goal models from CAPEC attack patterns, and have practically applied this method to model 102 patterns. Based on these models, we further propose a systematic analysis process and a collection of formal inference rules so as to semi-automatically leverage the attack pattern models to operationalize attack strategies. In Section 5.4, we validate the utility of the holistic attack analysis approach by applying it to a smart grid scenario, the result of which shows our approach is able to not only identify realistic threats but also to disclose detailed multistage attacks. Finally, we discuss our proposal from several aspects, comparing it with corresponding state-of-the-art.

Chapter 6

Integrating Security Patterns with Security Requirements Analysis

Good programmers know what to write. Great ones know what to rewrite (and reuse).

Eric S. Raymond

In Chapter 4 we have proposed to leverage existing security patterns to operationalize security requirements in terms of security mechanisms. However, several challenges were revealed during that work, hindering the application of security patterns. Firstly, there are normally more than one security pattern candidates that can potentially operationalize one security requirement, and analysts have to manually choose the best pattern to apply, which a non-trivial task as analysts need to manually compare all candidates. Specifically, the complexity of such task grows with the number of security requirements. Secondly, applying security patterns in terms of goal models is laborious and time-consuming, requiring analysts to have a full understanding of a security pattern they are about to use.

In this chapter, we propose a systematic and tool-supported approach to integrate existing security patterns into our three-layer framework, facilitating the application of security patterns. Note that the approach is an extension of the security goal operationalization analysis we have introduced in Section 4.3.3, which can be applied in an easier and more efficient way. As a result, the final output of this approach is an operationalized security goal model, which is specific to our holistic security requirements analysis approach.

Specifically, we define a collection of concept mappings between the constituent concepts of security patterns and contextual goal models, and provide a detailed process for practically constructing contextual goal models from existing security pattern specifications. Following this modeling process, we have pragmatically built contextual goal models for 20 security patterns documented in [Fernandez-Buglioni, 2013]. On the basis of such contextual goal models, a systematic process has been proposed for selecting the most appropriate security pattern and applying it to our three-layer security goal models (introduced in Chapter 4). Both the construction of security pattern models and the selection and application analysis are supported by our prototype tool, which will be presented in Chapter 8.

In the remainder of this chapter, we first describe the rationale of integrating security patterns with security requirements analysis in detail (Section 6.1). Then, we present how to establish contextual goal models based on textual security patterns in Section 6.2. In Section 6.3, we introduce a systematic process for selecting and applying security patterns. Finally, in Section 6.4, we compare our proposal with related work.

6.1 Security Patterns Complement Security Requirements Operationalization

Requirements Engineering (RE) has been increasingly focusing on security-specific issues, arguing for an upfront treatment of security in software system design. Goal-oriented modeling techniques constitute an effective way to capture and analyze stakeholder intentions. Proposals such as Secure Tropos [Mouratidis and Giorgini, 2007a], Secure-i* [Liu et al., 2009], and STS analysis [Paja et al., 2013], have been used by many researchers to analyze security requirements. In particular, our three-layer security requirements analysis framework, presented in Chapter 4, aims to holistically analyze security requirements for STSs. However, dealing with security concerns for complex software systems is a laborious and knowledge-intensive process, especially during the operationalization of security requirements.

Security patterns encapsulate reusable security knowledge that can support analysts with little security knowledge. Much work has been done to collect and document such patterns, resulting in several security pattern repositories, such as [Hafiz et al., 2007; Scandariato et al., 2008; Fernandez-Buglioni, 2013]. According to a recent mapping study [Ito et al., 2015], currently there are more than 200 security patterns have been proposed. However, among 30 recent security pattern approaches surveyed by Ito et al. [2015], only 13% of these approaches (i.e., four approaches) involve tooling. More specifically, these four tool-supported approaches deal with either the selection of security patterns or the application of security patterns, but none of them can support both topics. In addition, these tool-supported approaches only involve a limited number of security patterns (no more than six patterns). As summarized by Ito et al. [2015], "*security pattern research* *is still in its infancy in terms of automation*". As such, security patterns are handled largely by human, hindering the practical adoption of security patterns in large-scale systems (e.g., STSs). Especially when selecting security patterns from the large number of candidates, manual analysis becomes almost impossible.

Regarding the above challenges, we argue that integrating goal-oriented security requirements analysis with security pattern analysis can benefit both types of analysis. On one hand, goal models capture the rationale for applying security patterns and facilitate selection among alternatives. On the other hand, since security patterns encapsulate a large body of security knowledge, they can help to efficiently operationalize security requirements into specific security solutions.

6.2 Model Security Patterns as Contextual Goal Models

In this section, we first present a contextual goal modeling language, used to define contextual goal models for security patterns. In addition, we define mappings between constitutes of security pattern and the contextual goal modeling language, and present a detailed process for creating a contextual goal model based on security pattern specifications. Finally, we summarize some empirical observations derived from practically modeling 20 security patterns.

6.2.1 A Contextual Goal Modeling Language

We extend our three-layer goal modeling language described in Section 4.2 with *domain properties* in order to model and analyze context within a goal model. A *domain property* is a fact related to a particular domain, while a *design-time domain property* is a domain property that can be verified at design time by related analysts. For example, "*Computer systems on a local network connected to the Internet*" is a design-time domain property, and analysts can verify this fact during design time according to the designed system infrastructure. For another example, "*The number of users increases significantly*" is not a design-time domain property, as it can only be verified at run-time. Since security pattern analysis is carried out at design-time, we only capture *design-time domain properties* to analyze design-time contexts. A particular context can be arbitrarily complex, consisting of either a single domain property or could be an aggregation of domain properties of any complexity, typically, via *and/or* operators.

It is worth noting that the concept *design-time domain property* should be distinguished from the concept *domain assumption.* A *domain assumption* is always assumed to be true during system designs and does not need to be checked. For instance, in

Fig. 6.2, *"other security measures do not cover all possible attacks"* is a domain assumption. As introduced before in Section 3.2, we follow the method defined in [Ali et al., 2010] to model context within goal models. In particular, each context that is attached to a goal model element serves as a variation point, with the semantics that the goal model element is required if and only if the context holds.

Apart from the *design-time domain property*, we also include two unary relations *mandatory* and *preferred (nice-to-have)* in the contextual goal modeling language. Such relations were defined by Jureta et al. [2010], where *mandatory* indicates that a requirement must be satisfied and *preferred* indicates that a requirement is nice-to-have. In particular, we define the context that is attached to mandatory requirement is a *primary context*, while define the context that is attached to preferred requirements is a *secondary context*. Such differentiation helps us to select applicable security patterns, which will be introduced in Section 6.3.2.

6.2.2 A Process for Creating Contextual Goal Models from Security Patterns

To build contextual goal models that capture contents of security patterns, we focus on analyzing the five essential predefined sections of security patterns, as exemplified in Table. 6.1. For each of the five sections of the security patterns, we map the content of the section to concepts of the contextual goal modeling language by considering the definitions of those concepts. Our analysis results in a concept mapping, shown in Fig. 6.1. Apart from the concept mapping, we further provide detailed guidelines that constitute a systematic process for creating a contextual goal model for a given security pattern. The *Intrusion Detection System* pattern (Table. 6.1) is used throughout this section to illustrate the mappings and guidelines, with the corresponding contextual goal model shown in Fig. 6.2.

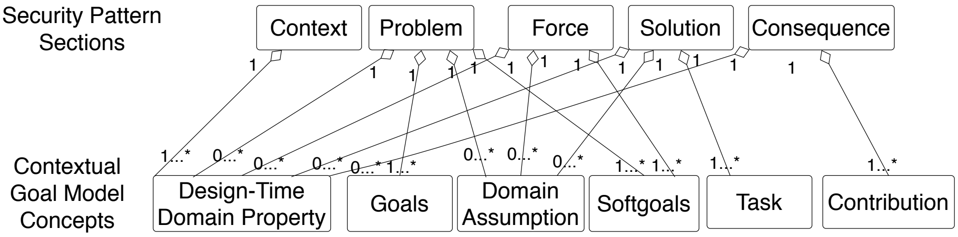


Figure 6.1: Concept mappings between contextual goal models and security patterns

Context section analysis. This section describes the initial context of the security pattern, in which the security problem occurs and is being solved. We model the context with one or multiple *design-time domain properties*. For example, the context *C1* in

Table 6.1: The specification of the Intrusion Detection System pattern [Fernandez-Buglioni, 2013]

Name: Abstract Intrusion Detection System
Context: Nodes for local systems that need to communicate with each other using the Internet.
Problem: An attacker may try to infiltrate our system through the Internet. We need to know when an attack is happening and take appropriate response.
Force: • *Incomplete security.* Security measures such as encryption, authentication and so on may not protect all our systems, because they do not cover all possible attacks. • *Non-suspicious users.* Request coming from a non-suspicious address (permitted by a firewall) could still be harmful and should be monitored further. • *Flexibility.* Hard-coding the type of attack can be done easily. But it will be hard and time-consuming to adapt to attack patterns that change constantly.
Solution: Each request to access the network is analyzed to check whether it conforms to the definition of an attack. If we detect an attack, an alert is raised and some countermeasures may be taken.
Consequence: • *Non-suspicious users.* A request coming from a non-suspicious address (permitted by a firewall) is further inspected and analyzed. • *Flexibility.* The detection information can be modified to include new attacks. • There is some overhead in the addition of IDSs to a system.

Fig. 6.2 is the initial context, which is represented by one design-time domain property *DTDP1* extracted from the context section. Note that the context *C1* is attached to the root goal of the security pattern model, which is extracted from the problem section (introduced below).

Problem section analysis. A *problem* is a description of a situation, for which stakeholders do not have a solution. We use one or several *goals* or *softgoals* to capture stakeholder needs concerning such a problem. As the problem is essential to a security pattern, we model the goals/softgoals that capture the problem as *mandatory* requirements, which have to be satisfied by security patterns.

We analyze this section sentence by sentence, each of which usually leads to the inclusion of a goal/softgoal. If there are several goals/softgoals, we need to consider the relations between sentences and determine the refinement structure of these goals/softgoals. It is worth noting that the description of the problem section may also involve domain

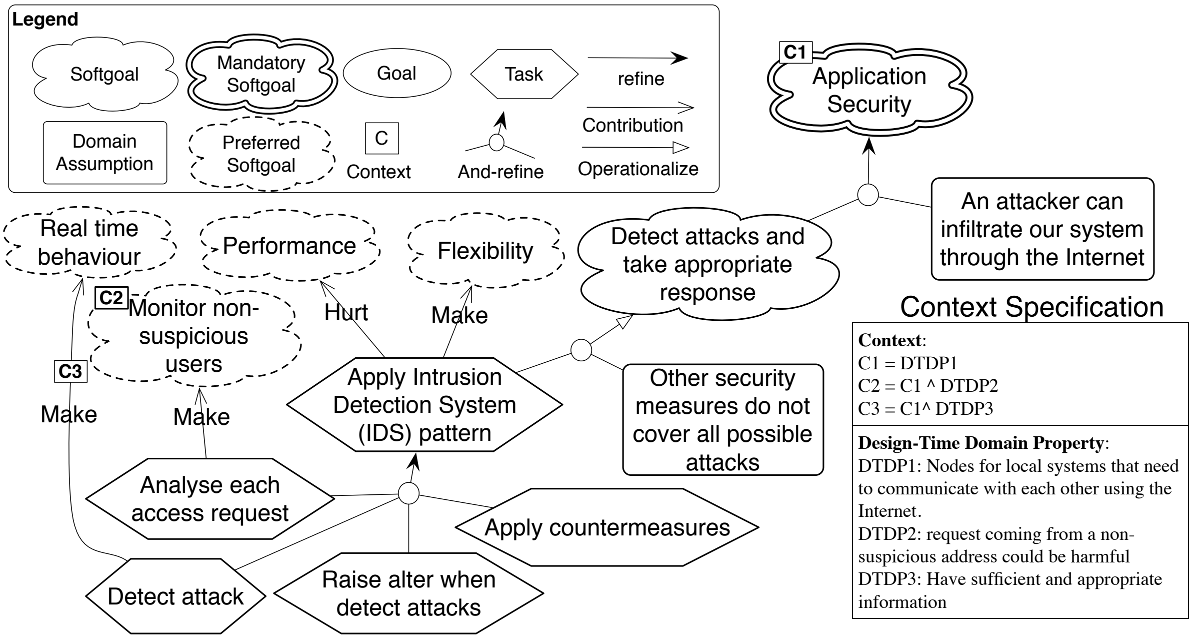

Legend
Softgoal
Mandatory Softgoal
Goal
Task
refine
Contribution
Operationalize
Domain Assumption
Preferred Softgoal
C
Context
And-refine
C1
Application Security
An attacker can infiltrate our system through the Internet
Detect attacks and take appropriate response
Real time behaviour
Performance
Flexibility
C2
Monitor non-suspicious users
C3
Hurt
Make
Make
Make
Apply Intrusion Detection System (IDS) pattern
Other security measures do not cover all possible attacks
Analyse each access request
Apply countermeasures
Detect attack
Raise alter when detect attacks
Context Specification
Context:
C1 = DTDP1
C2 = C1 ^ DTDP2
C3 = C1^ DTDP3
Design-Time Domain Property:
DTDP1: Nodes for local systems that need to communicate with each other using the Internet.
DTDP2: request coming from a non-suspicious address could be harmful
DTDP3: Have sufficient and appropriate information

Figure 6.2: The contextual goal model of the IDS pattern

assumptions, which should be identified and modeled within the refinement structure, such as *"An attacker may infiltrate a system through the internet"* shown in Fig. 6.2. The root element of a goal model constructed through this process must be mandatory.

To integrate the security patterns with security requirements analysis, we extract not only the specific problems that are solved by the security pattern, but also the high-level security requirements that lead to those detailed problems. If the high-level security requirements are not explicitly specified in this section, we need to do further analysis. In the *IDS* example, the first sentence presents a domain assumption "*Attacker may infiltrate a system through internet*", which also implicitly presents a high-level security requirement, i.e., the security of the software application should be protected. The next sentence "*We need to know when an attack is happening*" specifies a detailed problem, which is a refinement of the high-level security requirement. Thus, we obtain a model fragment as shown in the upper-right corner of Fig. 6.2.

Force section analysis. Forces are considerations, often contradictory, which have to be taken into account to determine the applicability of a pattern. These considerations are often related to non-functional requirements (NFRs), such as performance and cost. We model such forces as *preferred softgoals*, where stakeholders want to satisfy as many of such goals as possible. Other forces may belong to domain assumptions, which are always assumed to be true during the security analysis. For instance, the force *Existing security measures cannot cover all attacks*, shown in Fig. 6.2, is a domain assumption, under which the *IDS* security pattern operationalizes the security goals.

This section is specified in an itemized manner. Each item starts with a key word, based on which we decide whether the force is a preferred softgoal (e.g., *Flexibility*) or a domain assumption (e.g., *Incomplete Security*). It is worth noting that some preferred softgoals are context-dependent, only needing to be considered in particular context. For example, the softgoal "*Monitor non-suspicious users*" only holds under the context "*requests from non-suspicious address could be harmful*". If this context does not hold, the force does not need to consider. Therefore, we identify another context *C2* and add it to this softgoal.

Solution section analysis. This section describes actions that are carried out by a security pattern. We model them as *tasks*, which specify how the "system-to-be" implements a security pattern. Similar to the analysis in the *Problem* section, the relations between tasks should also be identified and modeled in an appropriate structure. As shown in Fig. 6.2, we identify four sub-tasks, which are siblings, for applying the IDS pattern. Note that, the granularity of solutions varies from pattern to pattern. If the information provided in this section is too general, we can optionally extract additional information from other non-essential sections, such as the *Structure*, *Dynamic*, and *Implementation* sections.

Consequence section analysis. This section describes the consequences of a security pattern, which indicates both benefits and liabilities of the pattern. We capture these influences using *contribution* links. This section is also documented in an itemized way, and each item should correspond to one force, documented in the *Force* section. However, the correspondence between the *Force* section and the *Consequence* section may not be strict. The *Consequence* section may introduce NFRs in addition to those described in the *Force* section. These NFRs should also be taken into account, via inclusion as preferred softgoals, when choosing a security pattern. For example, the consequence description "*There is some overhead in the addition of IDSs to a system*" (Table. 6.1) indicates the IDS pattern *hurt*s the performance of a system. The NFR (performance), which is not initially specified in the *Force* section, should be added into our model. In other cases, the preferred softgoals from the *Force* section may not be mentioned in the *Consequence* section. Thus, we need to infer the pattern's influences on those softgoals based on our understanding of the pattern, or search for related knowledge from other reliable knowledge bases.

Some influences on preferred softgoals are also context-dependent. As shown in Fig. 6.2, the task "*Detect attack*" can only *make* the softgoal "*Real time behaviour*" under the context that there are sufficient and appropriate information about attacks. It is worth noting that the influences of a security pattern may also depend on its detailed implementations.

For example, as described in the *Authenticator* pattern, the consequence "*The overhead depends on the protocol used*" cannot be directly modeled. We need to first model two alternative tasks "*Apply a simple protocol*" and "*Apply a complex protocol*", which refines the task "*Apply an appropriate protocol*" and then examine their influences respectively.

6.2.3 Empirical Observations

Thus far we have constructed contextual goal models for 20 security patterns described in [Fernandez-Buglioni, 2013]. All the pattern models are listed in Appendix F. During this exercise, we observed several issues, which may affect the quality of resulting models.

1. The specifications of some security patterns are incomplete, such as missing a section.
2. Not all security patterns are specified in a consistent way. For example, some patterns are specified in a threat-oriented manner, while others are in a function-oriented manner.
3. The granularity of descriptions may vary greatly among patterns. For instance, the solution section of some security pattern only describes general idea of the pattern in one sentence, while some other pattern uses several paragraphs to explain related security mechanisms.

These observations disclose that processing and modeling textual security patterns are time-consuming, and additional knowledge related to security patterns is usually required during this process. This fact further explains why security patterns are not widely applied. In the meanwhile, it justifies the value of our work, i.e. constructing *reusable* contextual goal models for 20 security patterns. In addition, the above observations also expose the shortcomings in existing security pattern specifications, which should be tackled by the security pattern community.

6.3 Selecting and Applying Security Patterns to Operationalize Security Requirements

Take the modeled security patterns as input, we propose a systematic process to select and apply them to operationalize critical security goals. It is worth noting that the proposed analysis process can be seen as an extended version of the *security goal operationalization* analysis (as presented in Section 4.3.3), which systematically leverages security patterns that have been modeled as contextual goal models to operationalize security goals. In this sense, we seamlessly integrate security patterns with security requirements analysis.

Note that we have implemented a prototype tool, which supports the application of our approach and will be described in detail in Chapter 8.

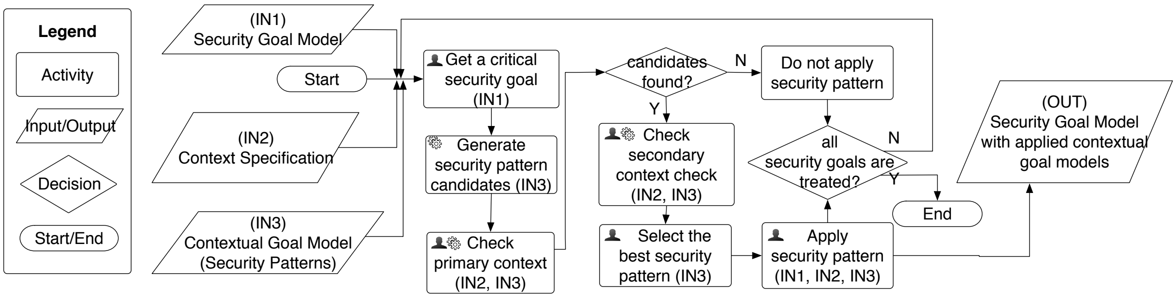

Legend
Activity
Input/Output
Decision
Start/End
(IN1)
Security Goal Model
Start
(IN2)
Context Specification
(IN3)
Contextual Goal Model
(Security Patterns)
Get a critical security goal (IN1)
Generate security pattern candidates (IN3)
Check primary context (IN2, IN3)
candidates found?
N
Y
Check secondary context check (IN2, IN3)
Select the best security pattern (IN3)
Do not apply security pattern
all security goals are treated?
N
Y
Apply security pattern (IN1, IN2, IN3)
End
(OUT)
Security Goal Model with applied contextual goal models

Figure 6.3: Security pattern analysis process

The entire security pattern analysis process is shown in Fig. 6.3, which requires three types of information as input:

1. The *security goal model*, which captures functional requirements and security requirements of the target domain. In particular, we use the three-layer requirements goal models (introduced in Chapter 4) for this purpose. The security requirements captured in the model are the objects that to be operationalized, while the functional requirements describe the domain.

2. The *context specification* describes the environments of the domain, which is composed of a list of design-time domain properties. This specification does not need to be complete at the beginning of the analysis, and it can be incrementally enriched during the application of security patterns. In particular, the initial context specification is derived from the three-layer requirements goal models. Then, during the context analysis, our prototype tool will interact with users for checking undecidable context, based on which the tool will update the context specification.

3. A collection of *security patterns*, which have been modeled in terms of contextual goal models (follow the method described in the last section). In particular, we here leverage the 20 security patterns we have pragmatically modeled (Section 6.2.3).

Note that, in Fig. 6.3, each input is assigned a tag, such as *IN1*. If the input is required by one activity, the tag of that input will be specified at the end of the description of that activity. The overall analysis process selects the best security pattern for each critical security goal, and applies the selected pattern to the security goal model, as well as updating the context specification. In particular, the final output of this analysis process is a security goal model with applied security patterns, i.e., solutions modeled in contextual goal models. Such a security goal model will be further used and analyzed in our holistic security requirements analysis, specifically, taking as the input of cross-layer analysis (Section 4.3.4). In the rest of this section, we describe each step of the process in detail, and finally illustrate the entire process with the smart grid scenario.

We here take part of a smart grid real-time pricing scenario as an example to illustrate the security pattern analysis process. This scenario has been briefly described before (Section 4.2.1), the full description of which will be presented in Section 9.1. Fig. 6.4 shows the exact part of the scenario, which we use for illustration.

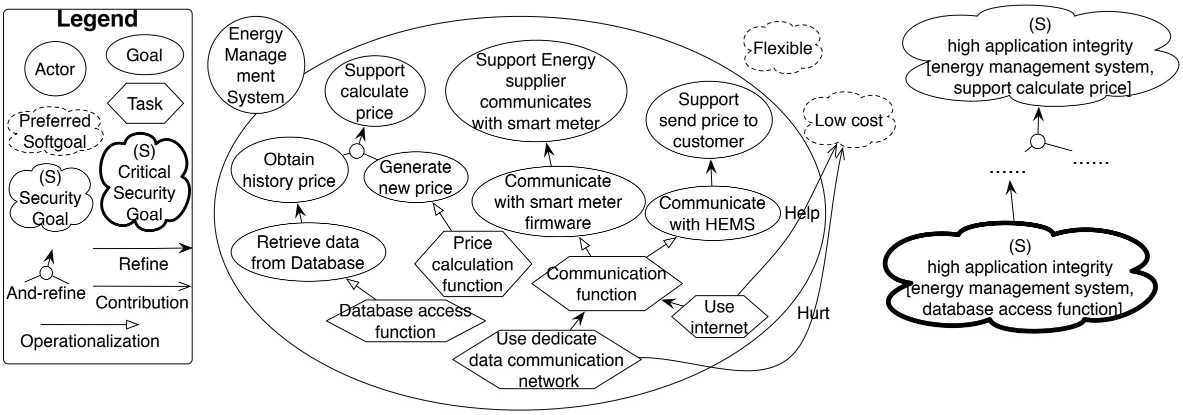

Legend
Actor
Goal
Task
Preferred Softgoal
(S) Critical Security Goal
(S) Security Goal
Refine
And-refine
Contribution
Operationalization
Energy Management System
Support calculate price
Support Energy supplier communicates with smart meter
Flexible
Low cost
Support send price to customer
Obtain history price
Generate new price
Communicate with smart meter firmware
Communicate with HEMS
Help
Retrieve data from Database
Price calculation function
Communication function
Database access function
Use dedicate data communication network
Use internet
Hurt
(S) high application integrity [energy management system, support calculate price]
......
......
(S) high application integrity [energy management system, database access function]

Figure 6.4: A part of the security goal model of the smart grid scenario

6.3.1 Generate Security Pattern Candidates

The security goal model contains a number of critical security goals, which are analyzed one by one in our analysis process. To operationalize a critical security goal, we first identify security pattern candidates, which can potentially treat the security goal. In particular, we match the security property of the security goal with the root goal of the contextual goal model of each security pattern (e.g. *Application Security* in Fig. 6.2) to determine whether a security pattern can be a candidate solution to the critical security goal. The results of this analysis reveal an initial set of security pattern candidates. Such analysis can be automated by our prototype tool.

It is worth noting that the match process takes the hierarchy of security property into account, such as described in [Firesmith, 2004; Scandariato et al., 2008]. For example, the security property *Application Integrity* is a specialization of *Application Security*. Thus, a security pattern which can tackle *Application Security* can also be applied to tackle the *Application Integrity* problem, such as the *IDS* pattern.

6.3.2 Security Pattern Selection

Once we have an initial set of the security pattern candidates, which typically contains more than one pattern, we carry out context-based selection to choose the most appropriate security pattern. To this end, we need to check each context to determine whether it holds within a particular domain.

The primary context, which is attached to mandatory requirements, is essential to the applicability of a security pattern. Such as shown in the IDS pattern (Fig. 6.2), the root goal is a mandatory requirement, which is valid when the primary context *C1* holds. If the primary context *C1* does not hold, the root goal will be deactivated, i.e., the pattern becomes inapplicable. In contrast to the primary context, the secondary context mainly affects the quality of the security pattern in terms of its contributions to the preferred softgoals. For instance, in the IDS pattern example, if the context *C2* does not hold, its corresponding preferred softgoal will be deactivated, as well as the contribution links connected to the softgoal.

Having the two types of contexts, we propose two steps for selecting security patterns. As shown in Fig. 6.3, we first check the primary contexts of security pattern candidates to filter inapplicable security patterns. After that, if there is more than one applicable security pattern left, we check the secondary contexts to determine the quality of each security pattern, based on which we select the best pattern. In particular, we quantify contribution links {*make, help, hurt, break*} as {*2, 1, -1, -2*} respectively to evaluate the influences a pattern has on the satisfaction of preferred softgoals, aiding in selection of patterns. Note that other more complicated goal satisfaction analysis techniques can also be used for this selection, such as those compared and evaluated in Horkoff and Yu [2013]

As manual checking whether a context holds is a non-trivial task, especially for complex and large models, we propose an interactive process that semi-automates this task. We propose to formalize the context of security pattern in Disjunctive Datalog rules, allowing us to automatically check them against the context specification using our prototype tool. For example, the context *C1* (in Fig. 6.2) can be formalized as below:

$R1\ \ hold(c1) \leftarrow node(N1) \wedge node(N2) \wedge communicate(N1, N2, internet)$

$R2\ \ not\_hold(c1) \leftarrow node(N1) \wedge node(N2) \wedge no\_communicate(N1, N2, internet)$

$R3\ \ undecidable(c1) \leftarrow not\ hold(c1) \wedge not\ not\_hold(c1)$

If neither *hold* nor *not_hold* can be inferred for a context, then the context is undecidable, and our prototype tool turns to users for manual check. On the basis of users' answers to a list of yes/no questions, our prototype tool can automatically update the context specification.

For each of the 20 security patterns we have modeled (shown in Appendix F), we have formalized its context expressions. In particular, during our formalization practice, we noticed that the contexts of security patterns are typically described in a very detailed manner, e.g., "*Enterprise applications in an organization's internal network are accessed by a broad spectrum of users that may attempt to abuse its resources*". Such detailed contexts cannot be captured in our three-layer requirements goal model, and thus most of them need to be checked by analysts. Although manual checks are still required, our prototype tool can facilitate such checks. In particular, if a context cannot be automatically checked, the tool can actively pop up dialog to analysts with corresponding questions. Based on the analysts' answer, the tool can automatically update the context specification. In such a way, one context only needs to be manually check at most once, as the check results have been updated in the the context specification. Another observation we have obtained is that contexts of different security patterns have little overlap. As such, we have almost defined particular predicates for each security pattern.

6.3.3 Security Pattern Application

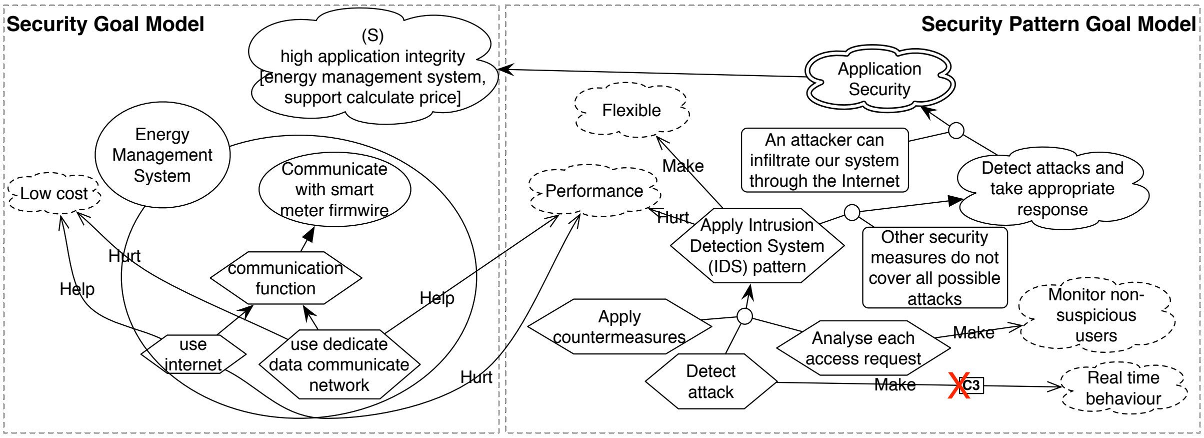

Security Goal Model
Security Pattern Goal Model
(S) high application integrity [energy management system, support calculate price]
Energy Management System
Low cost
Communicate with smart meter firmwire
communication function
use internet
use dedicate data communicate network
Help
Hurt
Help
Hurt
Application Security
Flexible
Performance
Make
Hurt
An attacker can infiltrate our system through the Internet
Detect attacks and take appropriate response
Apply Intrusion Detection System (IDS) pattern
Other security measures do not cover all possible attacks
Apply countermeasures
Analyse each access request
Make
Monitor non-suspicious users
Detect attack
Make
C3
Real time behaviour

Figure 6.5: Applying IDS pattern to the goal model of the smart grid scenario

Thanks to the reusable goal models we have constructed for security patterns, analysts do not need to manually construct the goal model of a security pattern each time they want to apply it. Thus, after selecting the best security pattern, the analyst can directly insert the goal model of the security pattern to the security requirements goal model, as illustrated in Fig. 6.5. Note that the red cross indicates the context *C3* does not hold, and the corresponding contribution link is deactivated.

To correctly integrate the goal model of a security pattern into the security goal model, firstly, the analyst needs to merge the softgoals newly introduced by the security pattern with the original softgoals by following the techniques proposed by Niu and Easterbrook [2007]. For example, in Fig. 6.5, the new softgoal *Flexibility* has been merged with the original softgoal *Flexible*. Secondly, the analyst should do a pairwise comparison of all the old elements with all the new elements to find what new contributions should be present. As shown in Fig. 6.5, two new contributions links are identified with regard to the new softgoal *Performance*. The model shown in Fig. 6.5 is the final output of our security pattern analysis, which is then used and analyzed in the cross-layer analysis of our holistic framework (Section 4.3.4).

Impact of security solutions. Once the above steps have been performed, we realized that the solutions offered by security patterns (i.e., *tasks* in Fig. 6.5) do not function by themselves but interact with existing system functions. For example, the task "*Analyze each access request*" requires each function that sends access requests to be recorded. Such impact of security solutions should be captured and reflected in the functional requirements, otherwise the system specification is incomplete. To this end, we propose a systematic approach to identify and enforce the impact imposed by security solutions in Chapter 7.

6.4 Related Work

There are several approaches that model security patterns as goal models. Mouratidis et al. [2006] extend their security analysis approach Secure Tropos by integrating four security patterns. Yu et al. [2008] propose to formally specify role-based access control as a security pattern in terms of i* models, and implement a tool to automatically detect contexts and apply security patterns. These approaches do not address the pattern selection issues, while our proposal has a main focus on the context-based pattern selection analysis. In addition, only a limited number of security patterns are presented in their work, and no details are provided on how to model security patterns as goal models. In contrast, we pragmatically target security patterns in existing pattern repositories. We not only provide a detailed method for modeling security patterns, but also practically follow the method to model 20 security patterns in [Fernandez-Buglioni, 2013].

Araujo and Weiss [2002] apply the Non-Functional Requirement (NFR) framework as a complementary representation for security patterns, which helps to analyze the tradeoffs between forces. In particular, they define the *force hierarchy* to represent interactions between forces, and model such hierarchy for 14 security patterns. Our approach is in line with this work, while further incorporates context-based analysis to facilitate the selection of security patterns. Asnar et al. [2011a] propose a method to design organizational patterns from SI* models, which deals with system security and dependability. Their approach proceeds in the opposite way of ours, they aim to extract security patterns from goal models, while we aim to apply security patterns in the context of goal model analysis.

Security patterns have been applied downwards. Shiroma et al. [2010] focus on applying security patterns in the context of UML diagrams. In particular, they have defined transformation rules to automate this application, rewriting UML diagrams based on the specification of security patterns. Sánchez-Cid and Maña [2008] argue that security patterns should not only be used in the early software engineering stage to represent abstract security solutions, but also need to be applied throughout the entire lifecycle of software development. As such, they provide a language which describe security solutions to assist software engineers in implementing security patterns into software applications. These approaches focus on the development stage of SDLC (Software Development Lifecycle), which contributes to the implementation of security solutions (included in security patterns). Different from them, our approach applies security patterns in the requirements stage of SDLC, focusing on identifying and applying the best security solutions that operationalize security requirements.

Hafiz et al. [2007] have presented and discussed several dimensions for classifying security patterns, such as security properties, logic tiers, security concepts, system viewpoints and so on. Such dimensions play an important role in navigating through security patterns, indirectly aiding the selection of security patterns. Our approach uses security properties for pattern classification, which serves as the first step for identifying relevant security patterns. After that, our approach further analyze the context of each relevant pattern in order to determine their applicability.

Other work has also been done on systematically analyzing textual patterns. Gross and Yu [2001] specify a systematic way to represent, analyze and apply design patterns by using NFR framework. They illustrate their method based on the study on several design patterns, and report their experiences in applying their method. However, their approach does not analyze the context of a pattern and provide no tool support for pattern selections. Supaporn et al. [2007] focus on generating security grammars, which are specified in extended-BNF formats, by analyzing descriptions of security patterns. In particular, they propose to build grammar trees to represent the semantics of security patterns. We argue such grammar trees can complement our approach, helping us to better understand and process the textual security patterns.

6.5 Chapter Summary

In this chapter, we propose to integrate security patterns with security requirements analysis by modeling security patterns as contextual goal models. We argue that our approach contributes to both the operationalization of security requirements and the adoption of security patterns (Section 6.1). In particular, we define concept mappings between security patterns and contextual goal models, and provide a detailed process for establishing contextual goal models based on security pattern specifications (Section 6.2). We have followed such method to pragmatically model 20 security patterns from a security pattern textbook [Fernandez-Buglioni, 2013]. On the basis of such security patterns, we propose a systematic process to select and apply security patterns and illustrate the process with a smart grid scenario (Section 6.3). The proposed analysis process can be seamlessly integrated into our holistic security requirements analysis framework (as presented in Chapter 4), enhancing the security goal operationalization analysis. Finally, we compare our proposal with related work (Section 6.4).

Chapter 7

Analyzing the Impact of Security Mechanisms

As part of our holistic security requirements analysis framework, we propose to operationalize security requirements in terms of security mechanisms by using security patterns (Section 4.3.3), which has been further expanded and detailed in Chapter 6. During such operationalization analysis, we have identified that the derived security mechanisms do not function on their own, but impose impact on existing functional requirements. Although related work has acknowledged that the application of security mechanisms can affect system requirements specifications, there is no systematic approach to describe and analyze this impact.

In this chapter, we propose to capture and enforce the impact that security mechanisms impose over system requirements in order to completely and correctly account for their integration. Specifically, we investigate in depth a collection of security mechanisms that are well documented in security pattern repositories [Scandariato et al., 2008; Fernandez-Buglioni, 2013] in order to better understand what they are and how they function. Based on this study, we propose a conceptual model for security mechanisms, apply and evaluate this model against 20 security mechanisms. On the basis of the conceptual model, we propose a systematic process for analyzing and enforcing security mechanisms on system requirements. In particular, given system requirements specifications and the security mechanisms to be applied, our approach can systematically and semi-automatically rewrite the requirements specifications, producing security-enhanced requirements specifications.

In Section 7.1, we detail the impact of security mechanisms and explain why it is important to capture and enforce such impact. Then, in Section 7.2, we present a Healthcare Collaborative Network (HCN) scenario, part of which is used for illustrating our approach. To analyze such impact, we first present an enriched requirements specification in Section 7.3, and then define a conceptual model which characterizes security mechanisms from a requirements viewpoint in Section 7.4. On the basis of such specifications, we propose a systematic way to analyze and enforce the impact of a security mechanism imposed on system requirements in Section 7.5. In particular, a set of inference rules have been defined to describe the impact of security mechanisms, semi-automating the analysis process. In Section 7.6, we evaluate the expressiveness of the proposed conceptual model by applying it to 20 security mechanisms (selected from [Scandariato et al., 2008; Fernandez-Buglioni, 2013]). Moreover, using these modeled security mechanisms, we further evaluate the proposed impact analysis by applying it to the full HCN scenario, which is presented in Section 7.2. Lastly, we compare our proposal with related work in Section 7.7.

7.1 Impact of Security Mechanisms

Dealing with security requirements in the early stages of the system development has become an important topic in Requirements Engineering (RE) and Security research, as software companies have grown tired of spending millions to fix system flaws downstream. Security requirements analysis techniques, such as misuse cases [Sindre and Opdahl, 2005], obstacle analysis [Van Lamsweerde and Letier, 2000], Secure Tropos [Mouratidis and Giorgini, 2007a], involve eliciting security requirements and identifying security mechanisms to fulfill those requirements. For example, *confidentiality* requirements can be operationalized by the security mechanism *encryption*. However, security mechanisms do not function independently but interact with and constrain parts of the target system in specific ways. As such, when applying a security mechanism, it requires to not only introduce new functional requirements, but also to modify existing system requirements.

Some approaches have claimed that the application of security mechanisms can influence system requirements specifications [Heyman et al., 2011; Haley et al., 2008]. However, these proposals only focus on new functional requirements that are introduced by a security mechanism and omit their impact on existing functional and non-functional requirements. In other words, their approaches operationalize security requirements into only functional requirements. In addition, there are neither systematic methods nor supporting tools available for analyzing and enforcing the impact of security mechanisms on system requirements.

We argue that system requirements specifications are not be complete unless they precisely capture such impact. For example, when applying an access control mechanism to protect a data asset stored in a server, this mechanism imposes global constraints on all functional requirements that access a server, which should be reflected in the requirements specification in order to correctly develop a secure system. Moreover, the quality of the system functions is affected by the application of security mechanisms, which should be captured and taken into account in order to select the best functional alternatives. For instance, applying the access control mechanism to a specific system function will impair the usability and performance of all related functions provided by the system. Thus, we believe that a security mechanism is not a localized solution that can be independently decided upon over other elements of a requirements specification.

In Chapter 6, we have proposed to seamlessly integrate security patterns into security requirements analysis by modeling security patterns as contextual goal models, which facilitates the context-based selection among alternative security patterns. After choosing the best security pattern, we apply its corresponding security mechanism (i.e. the *solution* of the security pattern), which is modeled using *tasks*, *domain assumptions*, and *softgoals*. Specifically, the application of a security mechanism amounts to directly attaching the security mechanism model into the requirements model via refinement and contribution links. However, that approach does not consider the impact of the mechanism on *existing* functional and non-functional requirements, capturing and analyzing which is a non-trivial task. Take the security mechanism VPN (Virtual Private Network) as an example, which requires endpoints to communicate via a cryptographic tunnel. To correctly apply the mechanism, all the functional requirements that communicate confidential information should be constrained by this mechanism, and these requirements are not easy to identify. Moreover, as the VPN mechanism impairs system performance, all the functional requirements that are constrained by VPN will have a negative influence on system performance. Such influences have to be taken into account when selecting the best requirements specification among alternatives.

As a result, we propose to capture and enforce the impact that security mechanisms impose over system requirements in order to completely and correctly account for their integration.

7.2 Scenario: The Healthcare Collaborative Network (HCN)

The HCN is a system that enables the exchange of healthcare messages and documents between and within organizations. The essential parts of the HCN include an admin server and a message flow server, which communicate with gateways deployed at both the publisher side and the subscriber side. A full description of the HCN can be found online[1]. It is worth noting that this scenario is independently used in this chapter for

---

[1] http://www.redbooks.ibm.com/redbooks/SG246779

illustrating and evaluating this impact analysis approach.

Fig. 7.1 shows part of the requirements goal model of the HCN, which captures the publisher gateway application, modeled using our requirements modeling language (Section 4.2). We assign unique identifiers to each node in the figure in order to facilitate the references in the remaining part of this chapter.

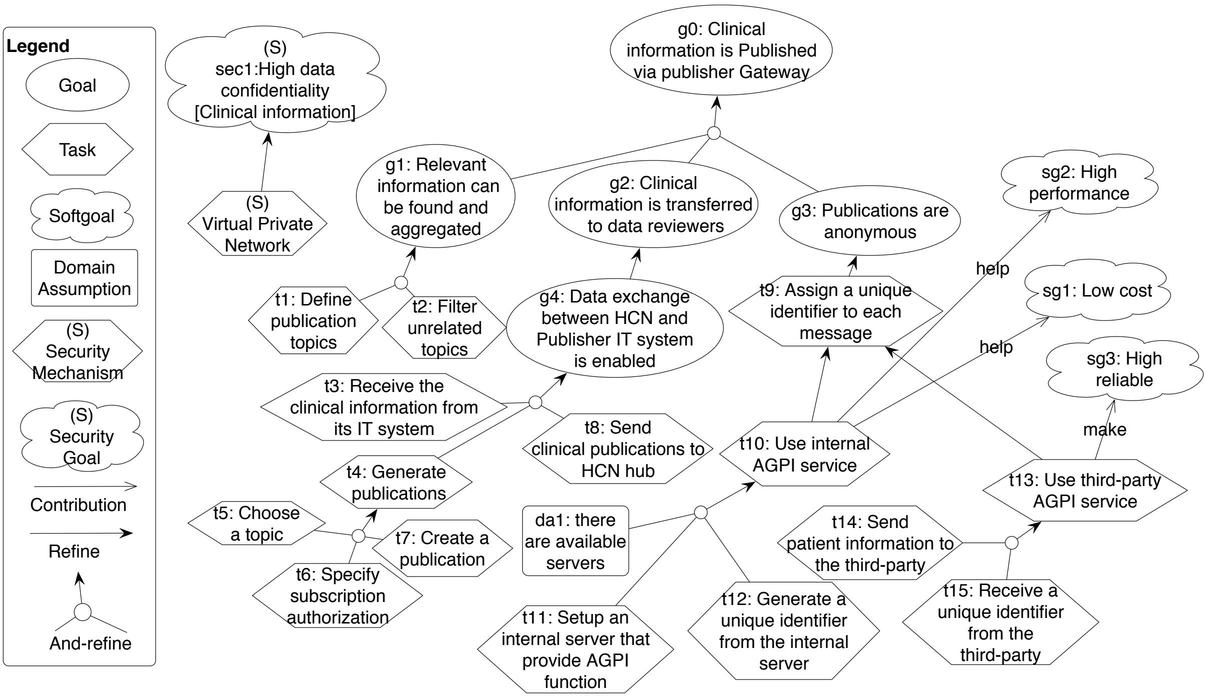

Legend
Goal
Task
Softgoal
Domain Assumption
(S) Security Mechanism
(S) Security Goal
Contribution
Refine
And-refine
(S) sec1:High data confidentiality [Clinical information]
(S) Virtual Private Network
g0: Clinical information is Published via publisher Gateway
g1: Relevant information can be found and aggregated
g2: Clinical information is transferred to data reviewers
g3: Publications are anonymous
g4: Data exchange between HCN and Publisher IT system is enabled
sg2: High performance
sg1: Low cost
sg3: High reliable
t1: Define publication topics
t2: Filter unrelated topics
t3: Receive the clinical information from its IT system
t4: Generate publications
t5: Choose a topic
t6: Specify subscription authorization
t7: Create a publication
t8: Send clinical publications to HCN hub
t9: Assign a unique identifier to each message
t10: Use internal AGPI service
t11: Setup an internal server that provide AGPI function
t12: Generate a unique identifier from the internal server
t13: Use third-party AGPI service
t14: Send patient information to the third-party
t15: Receive a unique identifier from the third-party
da1: there are available servers
help
help
make

Figure 7.1: A snippet of requirements goal model of HCN

7.3 An Enriched Requirements Specification

To analyze the impact of security mechanisms imposed on system requirements, we first define an enriched requirements specification. Such specifications consist of not only *goals (G)*, *softgoal (SG)*, *task (T)* (i.e., function), *domain assumption (DA) refinement (REF)* and *contribution (CON)* (as we have defined in Section 4.2.1), but also a new concept *task constraint (TC)* which reflects the impact of security mechanisms on tasks. In particular, a *task constraint* is specified in terms of task *invariants* and *pre/post-conditions.* The *invariants* describe properties that have to be true during the entire execution of the task. The *pre/post-conditions* describe properties that have to hold before/after the execution of the task. The value of a task constraint can be either a constant (e.g., *user_data*) or a predicate (e.g., *encrypted(user_data)*). Thus, an enriched requirements specification is defined as a 7-tuple, i.e.,

$$\mathcal{R} = \{G, SG, T, DA, REF, CON, TC\}$$

Fig. 7.1 presents an example of a requirements specification, including all these concepts except for task constraints. Note that the notation of the security mechanism shown in this figure (i.e., tasks with (S) annotation) is only used as a placeholder, which indicates a security mechanism is applied to operationalize a security goal. In Section 7.4, we will describe concepts of a security mechanism in detail, replacing such placeholders.

Apart from the enriched requirements specification, we also detail a task with three additional attributes (i.e., *subject*, *object*, and *operation*) in order to better analyze the semantics of the task. For example, as shown in Fig. 7.2, we detail the selected task with a subject *publisher_gateway_application*, an object *clinical_publications*, and an operation *send*. During the requirements elicitation phase, there are two ways to obtain such expanded attributes: firstly, interactively asking users when needed; secondly, automatically extracting the information from textual descriptions of tasks which have been elicited from stakeholders (with manual verification). For the second means, we propose to leverage Nature Language Processing (NLP) techniques, such as proposed in [Li et al., 2011], to identify the roles of phrases in a sentence and thus to automatically extract the *subjects*, *operations* and *objects* of the sentence.

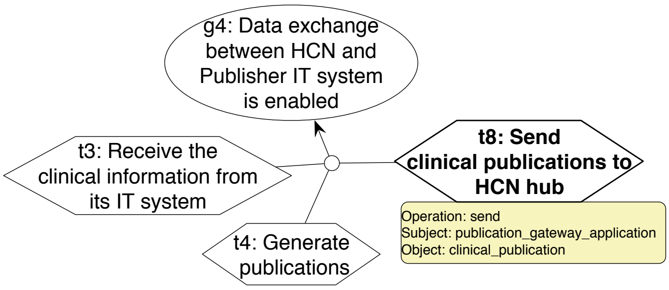

g4: Data exchange between HCN and Publisher IT system is enabled
t3: Receive the clinical information from its IT system
t8: Send clinical publications to HCN hub
Operation: send
Subject: publication_gateway_application
Object: clinical_publication
t4: Generate publications

Figure 7.2: An example of the enriched requirements elements

7.4 An Enriched Security Mechanism Specification

In this section, we propose a conceptual model to characterize security mechanisms from a requirements perspective. In particular, a security mechanism is specified in terms of *security tasks*, *assumptions*, *security constraints*, and *quality influences*. Since we exclusively focus on analyzing the impact of security mechanisms imposed on the requirements specification (as introduced in Section 7.3), we intend to map the concepts of the security mechanism to the requirements specification concepts as much as possible. In the remainder of this section, we describe each of the concepts that we use to model a security mechanism. An example of the VPN security mechanism is used for illustration, which

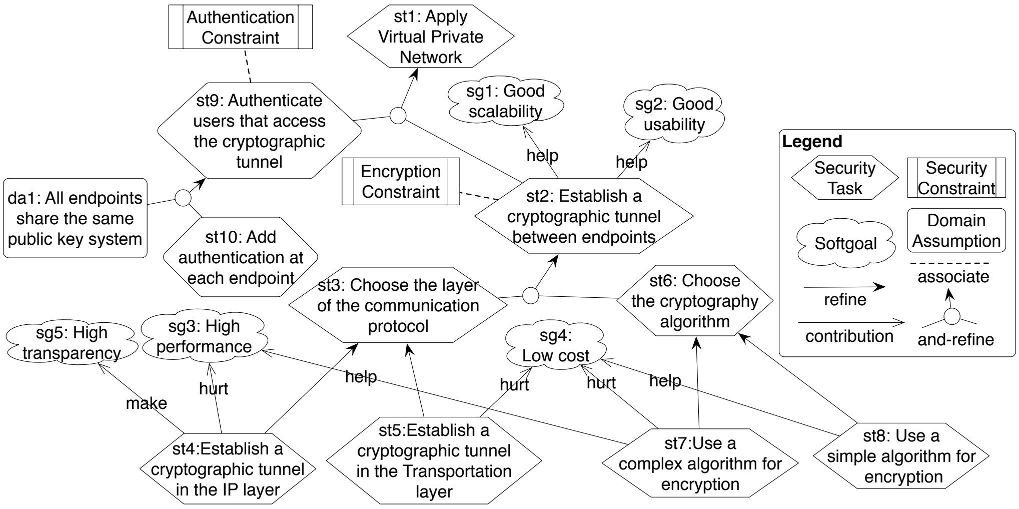

Authentication Constraint
st1: Apply Virtual Private Network
sg1: Good scalability
sg2: Good usability
st9: Authenticate users that access the cryptographic tunnel
Encryption Constraint
help
help
st2: Establish a cryptographic tunnel between endpoints
da1: All endpoints share the same public key system
st10: Add authentication at each endpoint
st3: Choose the layer of the communication protocol
st6: Choose the cryptography algorithm
sg5: High transparency
sg3: High performance
sg4: Low cost
make
hurt
help
hurt
hurt
help
st4:Establish a cryptographic tunnel in the IP layer
st5:Establish a cryptographic tunnel in the Transportation layer
st7:Use a complex algorithm for encryption
st8: Use a simple algorithm for encryption
Legend
Security Task
Security Constraint
Softgoal
Domain Assumption
refine
associate
contribution
and-refine

Figure 7.3: Modeling security mechanism — virtual private network (VPN)

is shown in Fig. 7.3. Note that the textual specification of this mechanism can be found in [Fernandez-Buglioni, 2013].

Security tasks. A *security task* is a detailed action performed by a system to achieve certain security goals. We define the security task as a specialization of *task*, and use T_S to represent the set of security tasks of a security mechanism. Each security task has an additional attribute *asset*, beyond the 3 attributes of regular tasks we have described before (Section 7.3). This attribute specifies the asset that is protected by a security task, from which we can infer the impact of the security task. As the target asset of a security task depends on the application scenario of the security task, the acquisition of this attribute is specified during the analysis process, described in Section 7.5.

As with all tasks, a composite security task can be decomposed into detailed security tasks, and we define the set of refinement relations between security tasks as REF_S. Note that we use the *root* security task to indicate the overall security mechanism, which can be repeatedly refined till reaching *leaf* security tasks, as shown in Fig. 7.3.

Assumptions. An *assumption* specifies an expected state of affairs, under which the security mechanism can be applied correctly. Normally, these assumptions are captured during the refinements of security tasks, such as the assumption "*All endpoints share the same public key system*", presented in Fig. 7.3. We map this concept to *domain assumption*, and use DA_S to represent the set of assumptions made in a security mechanism.

Security constraints. A security mechanism does not exist independently, but interacts and constrains existing system tasks in order to ensure that security requirements are satisfied. Thus, we explicitly capture such interactions between security tasks and tasks in the requirements model by using security constraints. We use SC to present the

Table 7.1: Security constraint rules

Global impact of security constraints

$$\begin{aligned}
&\textit{Rule\_1:}\ \mathit{constrain}(ST,T) \leftarrow \mathit{has\_operation}(T,F) \wedge \mathit{transfer\_operation}(F)\\
&\qquad \wedge \mathit{has\_object}(T,O) \wedge \mathit{protect}(ST,O)\\
&\qquad \wedge \mathit{has\_constraint}(ST,\mathit{encryption\_constraint})\\
&\textit{Rule\_2:}\ \mathit{constrain}(ST,T) \leftarrow \mathit{has\_operation}(T,F) \wedge (\mathit{protect}(ST,F)\\
&\qquad \vee(\mathit{access\_operation}(F) \wedge \mathit{has\_object}(T,O) \wedge \mathit{protect}(ST,O)))\\
&\qquad \wedge \mathit{has\_constraint}(ST,\mathit{authentication\_constraint})\\
&\textit{Rule\_3:}\ \mathit{constrain}(ST,T) \leftarrow \mathit{has\_operation}(T,F) \wedge (\mathit{protect}(ST,F)\\
&\qquad \vee(\mathit{access\_operation}(F) \wedge \mathit{has\_object}(T,O) \wedge \mathit{protect}(ST,O)))\\
&\qquad \wedge \mathit{has\_constraint}(ST,\mathit{authorization\_constraint})\\
&\textit{Rule\_4:}\ \mathit{constrain}(ST,T) \leftarrow \mathit{has\_operation}(T,F) \wedge \mathit{protect}(ST,F)\\
&\qquad \wedge \mathit{has\_constraint}(ST,\mathit{centralization\_constraint})\\
&\textit{Rule\_5:}\ \mathit{constrain}(ST,T) \leftarrow \mathit{has\_operation}(T,F) \wedge \mathit{access\_operation}(F)\\
&\qquad \wedge \mathit{has\_object}(T,O) \wedge \mathit{protect}(ST,O)\\
&\qquad \wedge \mathit{has\_constraint}(ST,\mathit{protection\_constraint})\\
&\textit{Rule\_6:}\ \mathit{constrain}(ST,T) \leftarrow ((\mathit{has\_function}(T,F) \wedge \mathit{protect}(ST,F))\\
&\qquad \vee(\mathit{has\_object}(T,O) \wedge \mathit{protect}(ST,O)))\\
&\qquad \wedge \mathit{has\_constraint}(ST,\mathit{auditing\_constraint})
\end{aligned}$$

set of security constraints imposed by a security mechanism.

As an initial effort towards such impact analysis, we summarize six security constraints after investigating more than 40 reusable security mechanisms that are documented in a security pattern textbook [Fernandez-Buglioni, 2013] and a security pattern repository [Yskout et al., 2006]. The six security constraints include *Encryption Constraint*, *Authentication Constraint*, *Permission Constraint*, *Centralization Constraint*, *Protection Constraint*, and *Auditing Constraint*. Each of these security constraints implies that a security task constrains specific tasks which have certain properties. Thus, according to the meaning of each security constraint, we define security constraint rules for each particular security constraint to identify tasks that are constrained by a security task. The full list of security constraint rules are shown in Table 7.1. Take the *Rule_1* as an example: if a security task ST has an *encryption_constraint*, which targets the asset O, and there is a task T that has an operation F, which $transfers$ the asset O, then the task T is constrained by the security task ST. Once having a list of security constraints, we need to go through each security task modeled before to identify whether it imposes certain security constraint. For example, as shown in Fig. 7.3, we identify that the security tasks *st2* and *st9* impose the *Encryption Constraint* and *Authentication Constraint*, respectively.

The proposed security constraints are not intended to be complete, but provide good coverage when considering the content of the 40 investigated security patterns. Additional constraints, together with their corresponding constraint rules (e.g., Table 7.1), can be incrementally integrated into our work.

Quality Influences. Each security task not only changes functions of a system, but may also influence the qualities of the system, either positively or negatively. We use a set of *contribution* links to capture such quality influences, which are represented as CON_S. A contribution link is a triple, which specifies the influence imposed by a security task over system related quality (captured as a softgoal). We define the set of softgoals affected by a security mechanism as SG_S. Thus, the quality influences are defined as:

$$CON_S \subseteq T_S \times \{make, help, hurt, break\} \times SG_S$$

For example, in Fig. 7.3, security task *st4:Establish a cryptographic tunnel in the IP layer makes* softgoal *sg5:High transparency*, while *hurts* another softgoal *sg3:High performance.*

7.5 A Systematic Process for Analyzing the Impact of Security Mechanisms

In this section, we propose a systematic process to analyze and enforce the impact security mechanisms impose on the existing system requirements specification, which is shown in Fig. 7.4. We take the enriched requirements specification $\mathcal{R}$ and the to-be-applied security mechanism specification $\mathcal{M}$ as the input of our analysis, i.e.,

Input: $\mathcal{R} = \{G, SG, T, DA, REF, CON, TC\}$, $\mathcal{M} = \{T_S, REF_S, DA_S, SC, SG_S, CON_S\}$

By systematically analyzing the impact of the security mechanism, our approach will generate an updated requirements specification, $\mathcal{R}'$, which reflects all the impacts of the security mechanisms imposed on the requirements specification, i.e.,

$$\text{Output: } \mathcal{R}' = \{G', SG', T', DA', REF', CON', TC'\}$$

We illustrate the analysis process by analyzing the impact of the VPN mechanism (Fig. 7.3) imposed on the piece of requirements specification of the HCN scenario (Fig. 7.1). It is worth noting that if there are multiple security mechanisms need to be applied, all of them will be analyzed iteratively using the same approach.

Step 1: Integrate Security Tasks. All security tasks, as a specialization of tasks, are directly incorporated into the initial requirements specification, as well as the refine-

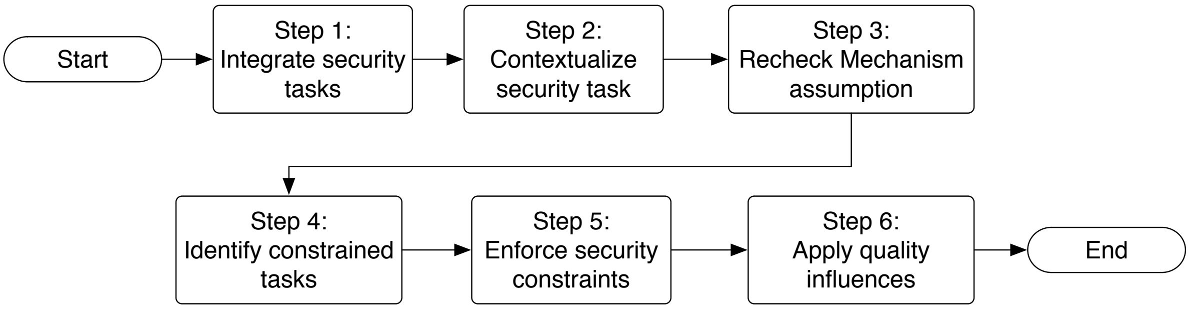

Start
Step 1: Integrate security tasks
Step 2: Contextualize security task
Step 3: Recheck Mechanism assumption
Step 4: Identify constrained tasks
Step 5: Enforce security constraints
Step 6: Apply quality influences
End

Figure 7.4: The process for analyzing impact of security mechanisms

ments relations among them (if they exist). As such, the integration is defined as follows:

$$T = T \cup T_S \ , \ REF = REF \cup REF_S$$

As a security mechanism is applied to operationalize a security goal, the *root* security task of the security mechanism will replace the placeholder described in Fig. 7.1, and is directly linked to the security goal. In the illustrating example, the result of integrating security tasks of the VPN mechanism to the requirements specification is shown in the right part of Fig. 7.5 (*st1-st10*).

Step 2: Contextualize Security Tasks. Once security tasks are integrated into the requirements and linked to a particular security goal, the target assets of security tasks should be determined in order to support the identification of constrained tasks in a later step. Each security goal in the requirements specification has already been specified an asset, such as the security goal *sec1* is specified with an asset *clinical_information* (Fig. 7.5). Thus, the security tasks that are applied to satisfy a security goal will inherit the asset from that security goal. In the illustrating example (Fig. 7.5), all the applied security tasks have the asset *clinical_information*, automatically derived from security goal *sec1*.

Step 3: Recheck Assumptions. When applying a security mechanism to a system within a particular domain, assumptions made in the mechanism should be further checked about whether or not it is still an assumption in the domain. Thus, a heuristic question can be asked, "Is the assumed phenomenon inside the boundary of system design now?" If so, we need to replace this assumption with a security task which *realizes* the assumption, and then add this security task to the set of tasks, i.e.,

$$T = T \cup \{a | \forall a \in DA_S, inside\_design\_boundary(a)\}$$

In this case, the newly added security tasks should be appropriately performed to ensure that the security mechanism is executed correctly. If the answer to the question is *No*, the properties in the assumption keep being assumed to be held, and we add the assumption to the set of domain assumptions, i.e.,

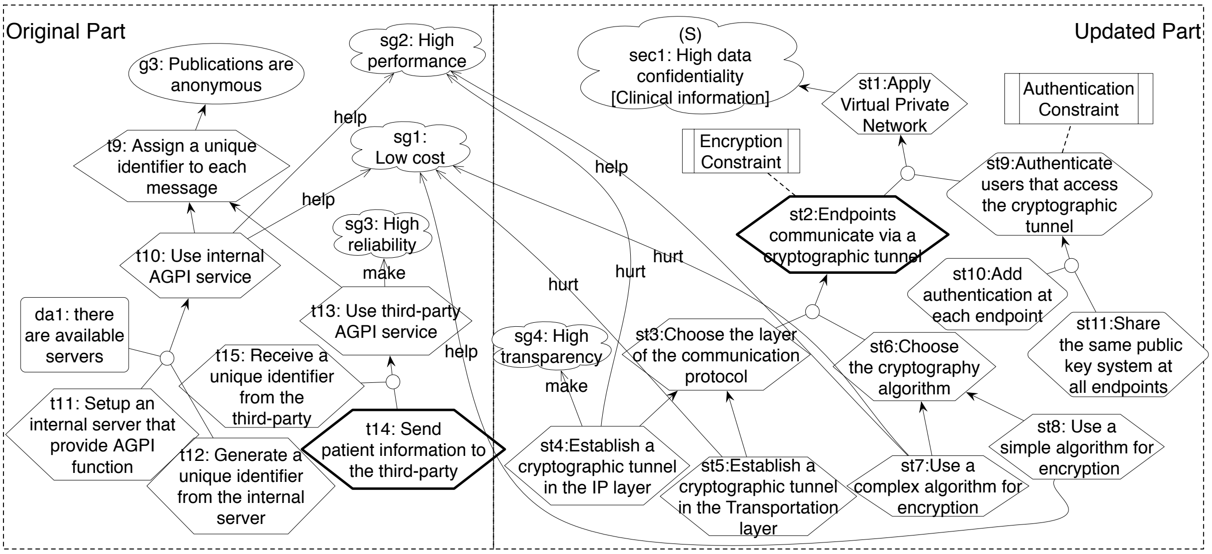

Original Part
Updated Part
g3: Publications are anonymous
sg2: High performance
sg1: Low cost
sg3: High reliability
t9: Assign a unique identifier to each message
t10: Use internal AGPI service
t13: Use third-party AGPI service
da1: there are available servers
t11: Setup an internal server that provide AGPI function
t12: Generate a unique identifier from the internal server
t15: Receive a unique identifier from the third-party
t14: Send patient information to the third-party
help
help
help
help
make
(S) sec1: High data confidentiality [Clinical information]
st1:Apply Virtual Private Network
Authentication Constraint
Encryption Constraint
st9:Authenticate users that access the cryptographic tunnel
st2:Endpoints communicate via a cryptographic tunnel
st10:Add authentication at each endpoint
st11:Share the same public key system at all endpoints
sg4: High transparency
make
hurt
hurt
hurt
st3:Choose the layer of the communication protocol
st6:Choose the cryptography algorithm
st8: Use a simple algorithm for encryption
st4:Establish a cryptographic tunnel in the IP layer
st5:Establish a cryptographic tunnel in the Transportation layer
st7:Use a complex algorithm for encryption

Figure 7.5: Impact of the application of VPN (part)

$$DA = DA \cup \{a|\forall a \in DA_S, outside\_design\_boundary(a)\}$$

In our example, the assumption of the VPN mechanism "*All endpoints share the same public key system*" is determined to be inside the system design boundary. So we create a security task based on this assumption (i.e., the *st11* in Fig. 7.5), and add this security task to the set of tasks.

Step 4: Identify Constrained Tasks. After security tasks have been contextualized with the asset information, we now apply the security constraint rules (Table 7.1) to automatically identify interactions between security tasks in the security mechanism specification and tasks in the requirements specification, i.e., identifying which tasks are constrained by a security task.

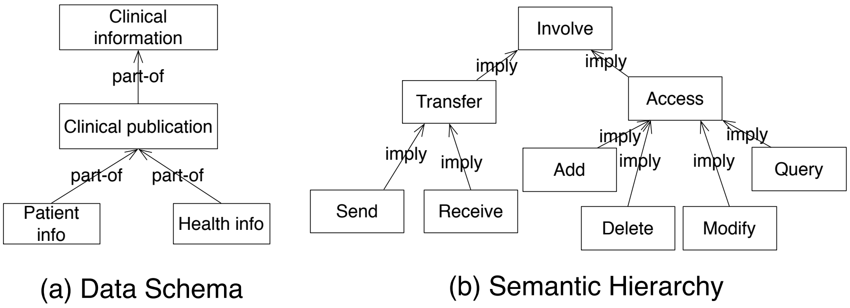
During the above impact identification, we are concerned about not only the information derived from the two specifications (i.e., $\mathcal{R}$ and $\mathcal{M}$), but also additional domain knowledge models, such as data schemes (Fig. 7.6 (a)) and semantic hierarchies of words (Fig. 7.6 (b)). These models provide auxiliary rules to facilitate the analysis, e.g., the following rules:

Rule_7: $protect(ST, A2) \leftarrow protect(ST, A1) \wedge part\_of(A1, A2)$

Rule_8: $transfer\_opertiona(O) \leftarrow send\_operation(O)$

Rule_7 indicates that if an asset needs to be protected, all the parts of this asset also should be protected. *Rule_8* indicates that if an operation is of the type of *send*, then it is also of the type of *transfer*.

In our example, we apply Rule 1 and identify three tasks {*t3, t8, t14*} (Fig. 7.1), which are constrained by security task *st2*. Note that, for illustration purposes, Fig. 7.5 only represents part of the original requirements model that is related to *t14*.



(a) Data Schema (b) Semantic Hierarchy

Figure 7.6: Examples of knowledge models

Step 5: Enforce Security Constraints. After identifying all tasks that are constrained, we further enforce security constraints on those tasks. In particular, we propose specific enforcement measures for each of the six security constraints in accordance with their meanings, which are detailed in Table 7.2. In this table, we first present the impact introduced by each security constraint. After that we describe the concrete enforcement measures, which are either adding task constraints or replacing tasks. For example, the *Encryption Constraint* adds a new pre-condition to the constrained task, the *Protection Constraint* adds a new invariant to the constrained task, and the Auditing *Constraint* adds a new post-condition to the constrained task. Apart from imposing task constraints, the *Centralization Constraint* replaces the constrained task with the corresponding security task. In this case, all the refinement relations that were linked to the constrained task are now redirected to the security task, and then the constrained task is removed.

According to the proposed enforcement measures, in our example, we enforce the encryption constraint on the constrained task $t14$ (Fig. 7.5), i.e., adding a new pre-condition $performed(st2)$ to this task.

Step 6: Apply Quality Influences. Many requirements analysis techniques rely on qualities, which are normally captured as non-functional requirements (NFRs), to select alternative requirements [Horkoff and Yu, 2013]. Due to the interactions between security tasks and tasks, the quality influences introduced by security tasks may affect system requirements decisions, which need to be re-evaluated.

As the first step of applying quality influences, we correlate the softgoals in SG_S with the softgoals in SG, i.e. checking whether they are the same softgoals. As the same concept may be presented by different terms in different ways, this correlation analysis may require additional techniques, such as the Repertory Grid Technique (RGT) [Niu and Easterbrook, 2007]. In the illustrating example (Fig. 7.3), SG_S of the VPN mechanism involves several softgoals among which *sg4: Low cost* and *sg3: High performance* have

Table 7.2: Enforcement measures for the six security constraints

Security Constraints	**Impact**	**Enforcement**
Encryption Constraint	The encryption security task should be done before the constrained task	*add(performed(st), t.precondition)*
Authentication Constraint	The authentication security task should be done before the constrained task	*add(performed(st), t.precondition)*
Permission Constraint	The authorization security task should be done before the constrained task	*add(performed(st), t.precondition)*
Centralization Constraint	The constrained task is replaced by the centralized security task	*replace(t, st)*
Protection Constraint	The protection security task should be enforced to cover the whole execution period of the constrained task	*add(cover_by(st), t.invariant)*
Auditing Constraint	The auditing security function should be done after the execution of the constrained task	*add(need_to_perform(st), t.postcondition)*

Remark: the *st* indicates the corresponding security task of a security constraint, while the *t* stands for the constrained task.

been correlated with softgoals in SG (in this particular case, the correlated softgoals have the same contents). For the softgoals in SG_S that are not correlated, the analyst needs to re-evaluate stakeholders' non-functional requirements to decide whether to include these softgoals. In our example, after evaluating the uncorrelated softgoal *High traceability*, we decided to consider this software for the entire system, i.e., adding it to the SG (shown in Fig. 7.5). This integration is defined below,

$$SG = SG \cup \{sg | \forall sg \in SG_S, uncorrelated(sg) \wedge decide\_include(sg)\}$$

However, the other uncorrelated softgoals, such as *sg2: Good usability*, are evaluated and are determined to not fit in with the current scenario. Once the above correlated softgoals and newly added softgoals are determined, all their corresponding contribution links in CON_S will be integrated into the requirements specification, i.e.,

$$CON = CON \cup \{contribute(st, inf, sg) | \forall contribute(st, inf, sg) \in CON_S, \exists sg \in SG\}$$

After correlating softgoals, we analyze the quality influences of a security task to its constrained tasks. Specifically, if a security task constrains a task, then all the quality influences introduced by this security task should be taken into account when evaluating the constrained task, especially if the constrained task is part of a requirements alternative. In the example (Fig. 7.5), since *t14* is constrained by *st2*, the correct execution of *t14* requires the appropriate interactions with *st2*. Thus, when evaluating the requirements alternatives that involve *t14*, such as the alternative tasks {*t11*,*t12*} vs. {*t14*,*t15*},

Table 7.3: Statistics of applying the conceptual model to 20 security mechanisms

	Security Task	**Assumption**	**Security Constraint**	**Quality Influence**
Total	89	15	27	148
Average	4.45	0.75	1.35	7.4

the influences *st2* imposed on the qualities (i.e., *sg1, sg2, sg4*) have to be taken into consideration.

7.6 Evaluation

In this section, we focus on answering two questions related to the proposed impact analysis approach. Firstly, we intend to evaluate the expressiveness of the proposed conceptual model of security mechanisms by applying it to model 20 existing security mechanisms. In particular, we concern whether the conceptual model can capture specific semantics of such security mechanisms, and how long does it take for such practical conceptualization. Secondly, we evaluate the effectiveness of our impact analysis approach by applying it to a medium-scale HCN scenario, where we report our experiences in the practical application of our proposal.

Evaluating the Conceptual Model of Security Mechanisms. We applied the proposed conceptual model to 20 security mechanisms, which are specified as reusable security solutions in the security pattern textbook [Fernandez-Buglioni, 2013]. These 20 security mechanisms are taken from the *solution* part of the 20 security patterns we have modeled in Chapter 6. In particular, we focus on capturing and modeling the security constraints imposed by such security mechanisms. On average, each mechanism cost me 20-30 minutes to model, and the statistics of the overall results is summarized in Table 7.3.

During this practical modeling session, we found that the six types of security constraints (proposed in Section 7.4) were able to capture specific semantics of the 20 security mechanisms. We captured 27 security constraints in total for the 20 security mechanisms, on average 1.35 constraints per mechanism. In particular, we contend that each security mechanism must impose at least one security constraint, indicating the major feature of the security mechanism. On the other hand, a complex security mechanism can impose multiple security constraints that function together to deliver particular security features. Such as the VPN example shown before (in Fig. 7.3), which imposes not only *authentication constraints* to the communication tunnel, but also *encryption constraints.*

In our modeling practice, once we modeled all security tasks, we followed a bottom-up process to identify the security constraints imposed by each security tasks. Specifically, we first checked each leaf security task whether it imposed a security constraint, and then we moved forwards to check the parents of such security tasks. If we had identified that one security task imposed a security constraint, we would not model such a constraint for the parent of that security task. In this way, we kept the security constraints at the detailed level of granularity as much as possible.

From the statistics of other elements of security mechanisms, we observed several issues. Firstly, on average each mechanism had more than four security tasks, which implies that security mechanisms are normally described at high abstraction level and can be further refined into detailed security tasks. Also, this further explains why one security mechanism may impose multiple security constraints. Secondly, the number of quality influences we have modeled for the 20 security mechanisms was significantly larger than any other concepts (e.g., assumptions). Such phenomenon strongly justifies that security mechanisms can heavily affect the quality of systems, emphasizing the need of capturing and analyzing the quality influences of security mechanisms.

On the whole, by applying the conceptual model, a single security mechanism had around 14 nodes on average. Thus, we argue that the conceptual model is scalable to model a larger number of security mechanisms. In particular, such modeling work is performed once for all, i.e., the modeled security mechanisms can be reused in our impact analysis. In this sense, the 20 modeled security mechanisms can serve as the initial mechanism repository that is used by our impact analysis approach, while additional mechanisms can be incrementally added to this repository either by ourselves or by other researchers who have followed our modeling approach. It is worth noting that in this study we exclusively focus on evaluating the expressiveness of the proposed conceptual model. In particular, the study was performed by the conceptual model designer and we did not yet evaluate whether other people are able to model security mechanisms using our conceptual model. Such study is left for future work.

Evaluating the Impact Analysis Approach. We applied the proposed analysis approach to the full requirements model that we have built for the HCN scenario, as introduced in Section 7.2. In particular, we adopted our three-layer requirements modeling language to establish a comprehensive requirements model for the entire scenario, which contained 66 goals, 7 softgoals, 163 tasks, and 198 refinement links. The complete model is shown in Fig. 7.7, while the vector file of this model is available online[2]. In this study, we exclusively analyzed the impact of a VPN mechanism (Fig. 7.3), which had been de-

---

[2] `http://disi.unitn.it/~li/SoSyM/model_hcn.pdf`

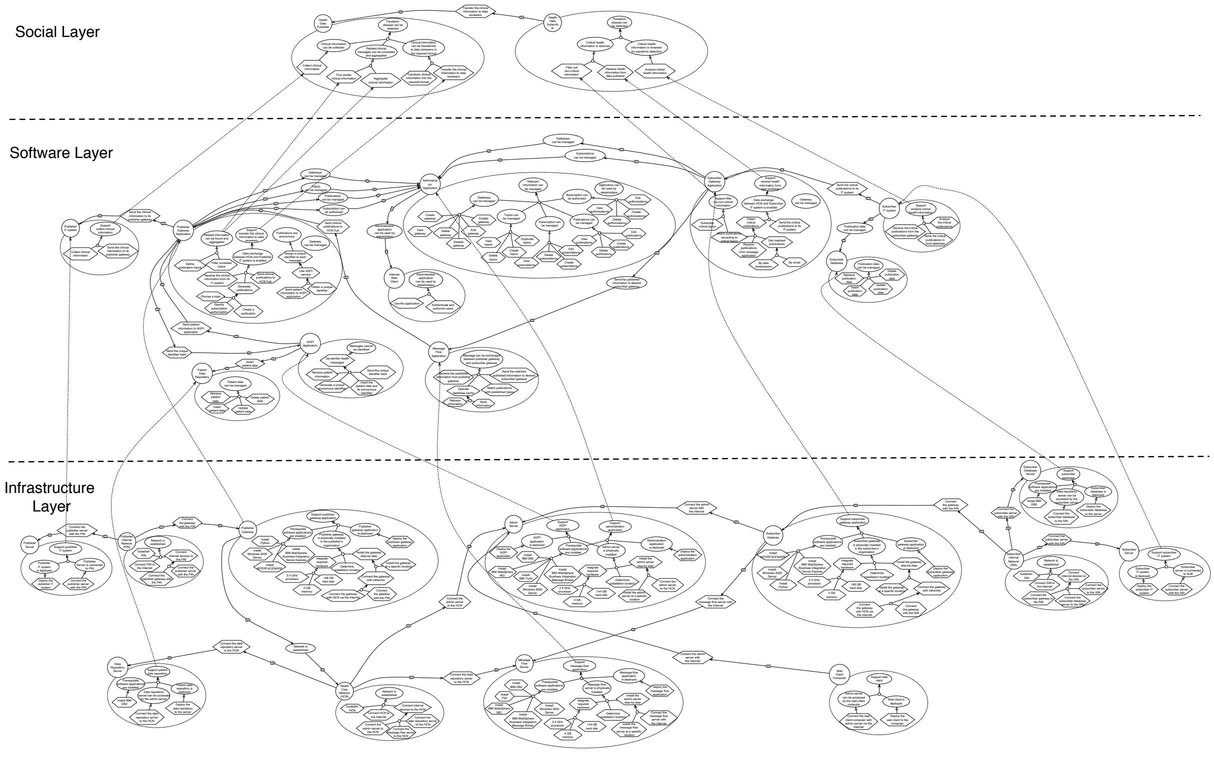


Figure 7.7: The complete three-layer requirements goal model of the HCN scenario

termined to operationalize a security goal *high data confidentiality [Clinical information, Data exchange between HCN and Publisher IT system is enabled]*. Since the VPN mechanism is a software security mechanism, the impact analysis was performed in the software layer of the three-layer requirements model, which contains 23 goals, 7 softgoals, 67 tasks, and 75 refinement links (as shown in Fig. 7.8). To be noted that we here exclusively report our experience in applying the impact analysis approach, and do not involve issues such as how to select a security mechanism to operationalize security goals (which is part of Chapter 6).

With the support of our prototype tool (Chapter 8), we were able to not only graphically model the requirements model and the security mechanism model, but also to automatically infer the tasks that were constrained by specific security tasks based on the proposed rules (Table 7.1) and thus to enforce such constraints. In addition, we leveraged another NLP tool to facilitate generating enriched requirements specification, which can automatically extract the *subject*, *object*, and *operation* from the description of a task by using NLP techniques [Li et al., 2011]. However, we encountered practical challenges when using the tool to automatically extract task attributes, and had to manually specify many task attributes. This was because the NLP tool extracts the task attributes by processing the content of each task, requiring the task description to be complete and

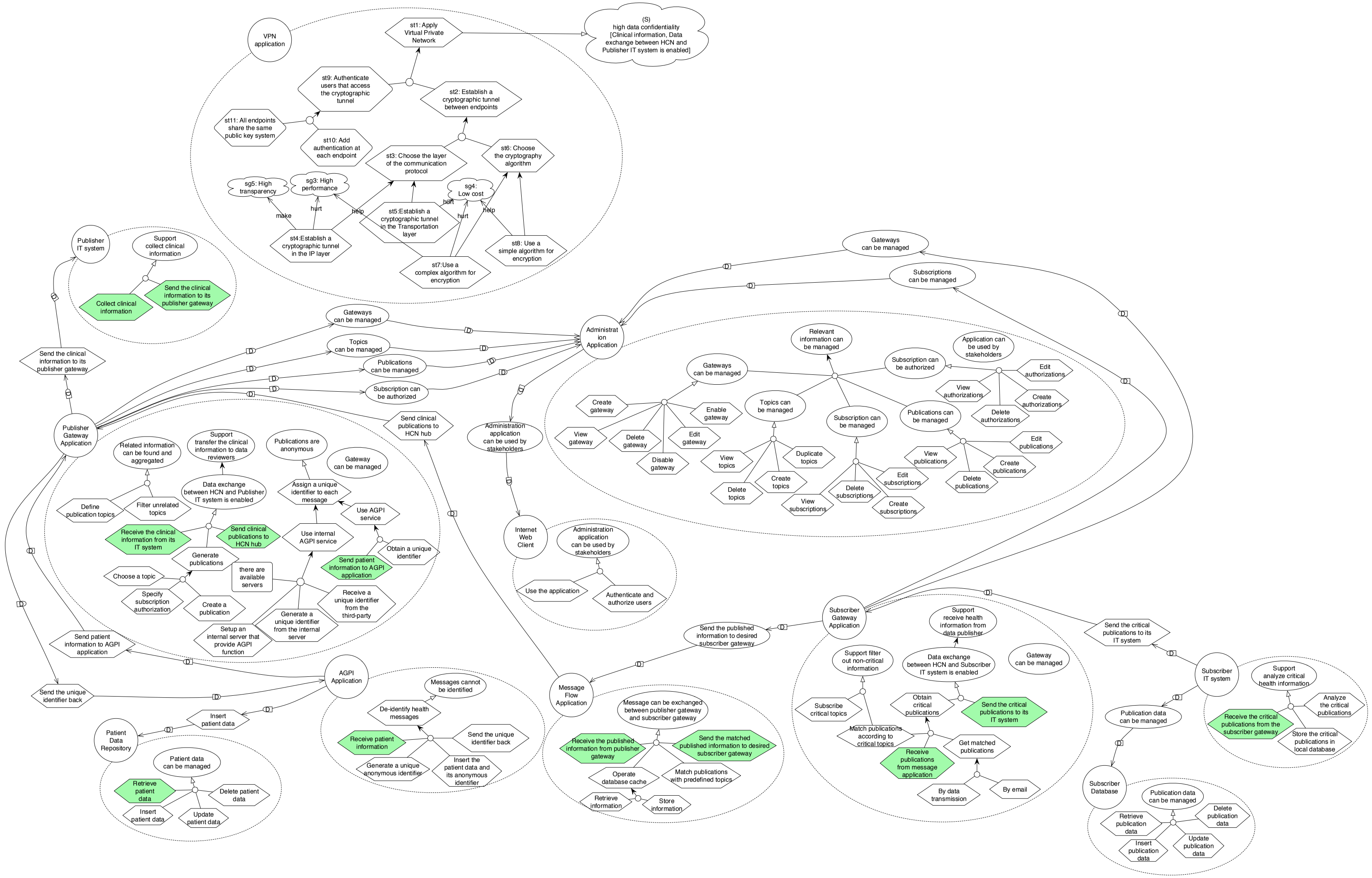

VPN application
st1: Apply Virtual Private Network
(S) high data confidentiality [Clinical information, Data exchange between HCN and Publisher IT system is enabled]
st9: Authenticate users that access the cryptographic tunnel
st2: Establish a cryptographic tunnel between endpoints
st11: All endpoints share the same public key system
st10: Add authentication at each endpoint
st3: Choose the layer of the communication protocol
st6: Choose the cryptography algorithm
sg5: High transparency
sg3: High performance
sg4: Low cost
make
hurt
help
st5:Establish a cryptographic tunnel in the Transportation layer
hurt
hurt
help
st4:Establish a cryptographic tunnel in the IP layer
st7:Use a complex algorithm for encryption
st8: Use a simple algorithm for encryption
Publisher IT system
Support collect clinical information
Collect clinical information
Send the clinical information to its publisher gateway
Send the clinical information to its publisher gateway
Gateways can be managed
Topics can be managed
Publications can be managed
Subscription can be authorized
Send clinical publications to HCN hub
Administration Application
Administration application can be used by stakeholders
Gateways can be managed
Subscriptions can be managed
Relevant information can be managed
Application can be used by stakeholders
Gateways can be managed
Subscription can be authorized
Create gateway
Enable gateway
Topics can be managed
Subscription can be managed
Publications can be managed
View authorizations
Edit authorizations
Create authorizations
Delete authorizations
View gateway
Delete gateway
Edit gateway
Disable gateway
View topics
Duplicate topics
Create topics
Delete topics
View subscriptions
Delete subscriptions
Edit subscriptions
Create subscriptions
View publications
Edit publications
Create publications
Delete publications
Internet Web Client
Administration application can be used by stakeholders
Use the application
Authenticate and authorize users
Publisher Gateway Application
Related information can be found and aggregated
Support transfer the clinical information to data reviewers
Publications are anonymous
Gateway can be managed
Define publication topics
Filter unrelated topics
Data exchange between HCN and Publisher IT system is enabled
Assign a unique identifier to each message
Use AGPI service
Receive the clinical information from its IT system
Send clinical publications to HCN hub
Use internal AGPI service
Obtain a unique identifier
Send patient information to AGPI application
Generate publications
Choose a topic
there are available servers
Specify subscription authorization
Create a publication
Receive a unique identifier from the third-party
Generate a unique identifier from the internal server
Setup an internal server that provide AGPI function
Send patient information to AGPI application
Send the unique identifier back
AGPI Application
Insert patient data
Patient Data Repository
Patient data can be managed
Retrieve patient data
Delete patient data
Insert patient data
Update patient data
Messages cannot be identified
De-identify health messages
Receive patient information
Send the unique identifier back
Generate a unique anonymous identifier
Insert the patient data and its anonymous identifier
Message Flow Application
Message can be exchanged between publisher gateway and subscriber gateway
Receive the published information from publisher gateway
Send the matched published information to desired subscriber gateway
Operate database cache
Match publications with predefined topics
Retrieve information
Store information
Send the published information to desired subscriber gateway
Subscriber Gateway Application
Support receive health information from data publisher
Support filter out non-critical information
Data exchange between HCN and Subscriber IT system is enabled
Gateway can be managed
Subscribe critical topics
Obtain critical publications
Send the critical publications to its IT system
Match publications according to critical topics
Get matched publications
Receive publications from message application
By data transmission
By email
Send the critical publications to its IT system
Subscriber IT system
Support analyze critical health information
Publication data can be managed
Receive the critical publications from the subscriber gateway
Analyze the critical publications
Store the critical publications in local database
Subscriber Database
Publication data can be managed
Retrieve publication data
Delete publication data
Insert publication data
Update publication data
D

Figure 7.8: Impact of the application of VPN over HCN scenario

grammatically correct. However, when building the requirements goal model, we did not specify the full description of a task but the essential part which made sense to most readers. For example, within the actor boundary of *Publisher Gateway Application*, we typically specify a task without claiming its subject as this is implied by the actor model. As such, the NLP cannot fully automate the extraction of task attributes and required additional manual effort for specifying such information. Similar challenges have also been reported by Casagrande et al. [2014], where the authors applied NLP techniques to process goal models. To tackle this challenge, we plan to improve the NLP tool by taking into account more semantics of the requirements goal model. Alternatively, we can choose to always specify the full details of each task.

After specifying the three attributes for each task in the scenario, I spent one hour to perform the impact analysis for the VPN mechanism. We report and evaluate such application of our approach in detail below:

- **Step 1**: We integrated all the security tasks of the VPN mechanism into the requirements model in order to operationalize the security goal *high data confidentiality [Clinical information, Data exchange between HCN and Publisher IT system is enabled]*. In particular, our prototype tool has good usability which allows us to easily *copy-paste* the established security mechanism models into the requirements model.

- **Step 2**: After inserting the security tasks to operationalize the security goal, the asset concerned by the security goal were then propagated to the security tasks we just inserted. This step was automatically executed by our prototype tool.

- **Step 3**: We then checked the assumption made by the VPN security mechanism. As we determined that the assumption was inside the system design boundary, it should be achieved by the system. Thus, we reformed it as a security task, replacing the previous assumption. This step was performed manually, requiring a through understanding of the target system, especially on the system boundary.

- **Step 4**: Since the VPN mechanism contains two security constraints, i.e., the authentication constraint and the encryption constraint associated with two different security tasks (as previously shown in Fig. 7.3), we then identified constrained requirements tasks by applying corresponding inference rules (*Rule_*1 and *Rule_*2 in Table 7.1). In particular, with the tool support we instantly identified 12 tasks that were constrained by the two security constraints, which have been highlighted in Fig. 7.8.

- **Step 5**: Once contained tasks had been identified, we then followed the enforcement measures (Table 7.2) to rewrite the specification of those tasks. In particular, we added two preconditions to each of the 12 constrained tasks, indicating that before performing these tasks a cryptographic tunnel must be first established and users who access the tunnel mush be authenticated.

- **Step 6**: As the last step, we processed the quality influences of the security mechanism. In particular, we manually checked all the qualities (i.e., softgoals) influenced by the VPN mechanism against seven qualities required by stakeholders. Eventually, two qualities were matched with existing qualities, while other three were not. Among the three unmatched qualities, one was determined to be required by the stakeholder and thus added to the requirements model, while the other two were decided to be excluded.

After going through the impact analysis process, the final security enhanced requirements specification is shown in Fig. 7.8. In particular, compared to the initial requirements specification, this rewritten specification comprehensively captured all the functional tasks that were introduced by the security mechanisms and their impact over initial functional and non-functional requirements specifications. Overall, we conclude that the proposed impact analysis approach can be applied to a medium-scale scenario to identify and enforce impact of security mechanisms within a reasonable amount of time. Beyond this study, we intend to apply our approach to larger scenarios in order to further assess the scalability of our approach. In addition, apart from the effectiveness, we want also to further evaluate other aspects of our impact analysis approach, especially, whether it can be adopted in reality by other practitioners (i.e., usability).

7.7 Related Work

The interaction between requirements and architecture was first emphasized by Nuseibeh [2001], where he proposes a twin peaks model to show these interactions at an abstract level. Heyman et al. [2011] and Okubo et al. [2012] specialize the twin peaks model in the security area, respectively. They all outline a constructive process for co-developing secure software architectures and security requirements, but do not consider the impact imposed by security architecture on other non-security requirements. In addition, none of these approaches has formalized the interactions between the twin peaks, and there is no tool has been developed to support the analysis process.

In Goal-Oriented Requirements Engineering (GORE), stakeholder's requirements, i.e., goals and softgoals should be operationalized into specific functions. As summarized by Dalpiaz et al. [2014], there are several types of operationalization among existing GORE approaches, namely: functional requirements operationalization, qualitative operationalization, adaptation requirements operationalization, and behavior operationalization. Most of the existing work about security requirements operationalization falls into the first category, i.e., operationalizing security requirements into particular functions [Mouratidis and Giorgini, 2002; Haley et al., 2008; Li and Horkoff, 2014]. However, in this paper, we argue that any single category summarized above is not enough to characterize the operationalization of security requirements. Instead, our proposal aims to provide a new category of requirements operationalization, which focuses on capturing various changes on existing requirements specification.

Apart from the type of requirements operationalization, the means of doing the operationalization is also an essential step of the analysis. Letier and van Lamsweerde [2002] have proposed to leverage operationalization patterns to guide the operationalization analysis, while Alrajeh et al. [2009] leverage machine learning techniques to operationalize goals. As these approaches help to guarantee the correctness of the obtained operational specification, they can complement our work during the step of enforcing security constraints, specifically, validating the enforcement rules.

Security, as a cross-cutting concern, has been investigated in an aspect-oriented manner. Gunawan et al. [2009] model both system functional designs and security mechanisms by using the collaboration-oriented behavior model, and propose to treat each security mechanism as a security aspect that can be inserted into different places of the system design. de Sousa et al. [2003] adapt the NFR framework to support aspect-oriented analysis. Specifically, they illustrate their approach with a security requirements example, as they treat security requirements as a NFR. However, the above approaches do not consider the quality influences imposed by security mechanisms.

The impact of security mechanisms has been enforced by using model transformation techniques. Shiroma et al. [2010] focus on applying security mechanisms onto UML class diagrams. They automatically enforce the security mechanism by defining transformation rules in ATLAS transformation language. However, this work focuses on the design phase and does not consider the impact on the system requirements. Yu et al. [2008] use i* constructs to model the context, problem, and solution of a security pattern, and automate the problem matching and application of the security solution by using ATL. However, their approach highly depends on the semantics of the constructs of i*, such as dependencies and roles, and cannot be generalized for all security mechanisms, such as encryption.

7.8 Chapter Summary

In this chapter, we focus on dealing with a particular challenge which we encountered during the application of our holistic security requirements framework. In particular, we propose to capture and enforce the impact that security mechanisms impose over system requirements in order to completely and correctly account for their integration. To this end, we first described an enriched requirements specification which can reflect the impact imposed by security mechanisms (Section 7.3). Then we propose a conceptual model in Section 7.4, which characterizes security mechanisms as security tasks, assumptions, security constraints, and quality influences. Built on such conceptual model, we propose a systematic way to analyze and enforce the impact that security mechanisms impose over the system requirements, which helps analysts to rewrite the initial requirements specification into a security-enhanced version (Section 7.5). In particular, a set of reasoning rules have been proposed in order to semi-automatically identify the exact part of the requirements specification constrained by security mechanisms. In Section 7.6, we have evaluated the expressiveness of our conceptual model against 20 security mechanisms documented in existing security pattern repositories. Moreover, we have applied the proposed impact analysis process to a HCN scenario, the results of which show that our proposal can help analysts to capture and enforce the impact of a security mechanism over a medium-scale scenario within reasonable time. Lastly, we compare our proposal with related work in Section 7.7.

Chapter 8

A Prototype Tool

Give us the tools, and we will finish the job.

Winston Churchill

In this chapter, we present a prototype tool MUSER (MUltilayer SEcurity Requirements analysis tool), which is developed to support our analytical methods proposed in Chapter 4-7, semi-automating holistic security requirements analysis. Generally speaking, this tool allows analysts to graphically model different models and perform reasoning on top of such models. We first introduce the architecture of the tool in Section 8.1, based on which we present a list of features that are offered by the tool in Section 8.2. After that we describe detailed use cases for each primary feature, shedding light on how to use the tool to perform holistic security requirements analysis (Section 8.3).

8.1 Architecture

MUSER is a Java-based program, which is developed on top of a professional diagramming application *OmniGraffle*[1] in order to leverage its powerful modeling features and thus efficiently model requirements models in three separate layers. The graphical models are automatically transformed into formal expressions (i.e., Disjunctive Datalog), which can be inferred using the inference engine DLV[2]. Specifically, upon the requests from analysts, the tool passes the models together with the analysis requests to the inference engine, which performs corresponding reasoning tasks. The inference results are then interpreted as corresponding graphical models in the canvas, presenting to users. In particular, the architecture of the tool consists of four components: *control*, *view*, *model*, and *inference*, as shown in Fig. 8.1. We introduce each of them in detail below.

---

[1] `http://www.omnigroup.com/omnigraffle`

[2] http://www.dlvsystem.com/

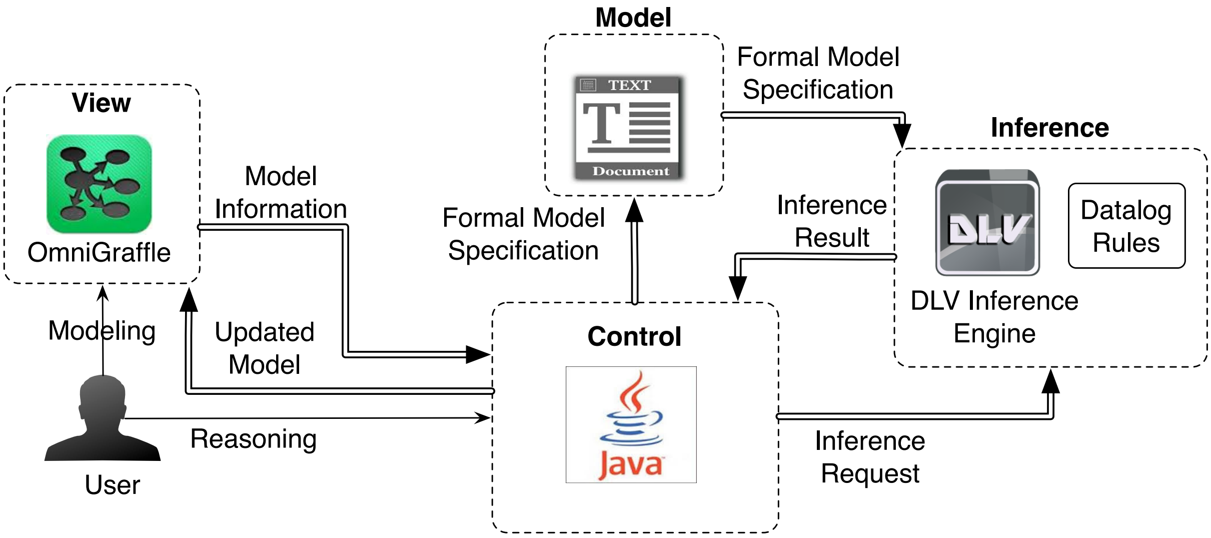


Figure 8.1: An overall architecture of MUSER

Control component controls the logic of the prototype tool and coordinates other components in order to deliver inference functions to users. When receiving user's inference requests, it imports related graphical models from the view component, and automatically generates a formal model specification using the formal predicates we have defined in Chapter 4-5. Then, the control component automatically calls the inference component to carry out corresponding inference tasks based on the formal model specification. Once receiving the inference results from the inference component, the control component automatically updates related model information and graphically presents them in the view component.

View component supports users with graphical modeling and can graphically show inference results to users. The major requirements for this component include: 1) support goal-oriented modeling and allow customized notations; 2) support multilayer modeling, i.e. modeling in different views while keeping connections among them; 3) be connected with the inference component to support reasoning. We choose a professional diagramming application OmniGraffle as the view component of our prototype, meeting all the above requirements. Especially, this application has many useful modeling features, such as automatic layout, outline view, and various export formats. In particular, we have defined a collection of interfaces for the view component, through which the control component can interact with graphical models in the canvas. In such a way, the prototype tool is flexible in the sense that the view component can be replaced by other applications that comply with the interfaces.

Inference component implements the inference rules proposed in our approach, and automates corresponding analytical tasks. In particular, we leverage DLV inference engine[3] to carry out inference based on the rules and facts. This component receives requests from the control component, performs corresponding inference tasks, and then returns results back to the control component.

Model component is responsible for storing the formal models, which are used by the inference component. Such formal models are automatically derived from graphical models built in the view component, and are then stored in text files. Note that if our analysis involves very complex and large models in the future, the model component can be upgraded and incorporate specialized database application.

8.2 Features

In this section, we summarize all the features provided by MUSER, which support the holistic security requirements analysis. In particular, we introduce modeling features and analysis features, respectively.

Modeling features. Since MUSER is built on top of *OmniGraffle*, it inherits many powerful modeling features from that professional diagramming application, such as automatic layout, outline view, and various export formats. Apart from such shared features, we have further customize *OmniGraffle* in order to provide two particular features for the holistic security requirements analysis. Firstly, we have produced a stencil including all the modeling constructs proposed in Section 4.2, assisting analysts in building all analytical models we have proposed in the thesis. Secondly, we have customized the model canvas to including three different layers. In particular, the visibility of each of these three layers is configurable, allowing users to view any of the three layers. As such, the tool can perfectly implement the modularity we advocate in this thesis.

Take Fig. 8.2 as an example for illustration. The modeling canvas is set in the middle, where users can build their models by leveraging all the modeling features offered by *OmniGraffle*. In the right part presents, users can find the stencil we have customized according to our modeling language. Moreover, there is an outline view provided in the left part, through which analysts can navigate modeling in different layers.

Analysis features. Using the modeling features described above, users can produce various analytical models required by our holistic security requirements analysis. Based

---

[3] `http://www.dlvsystem.com/dlv/`

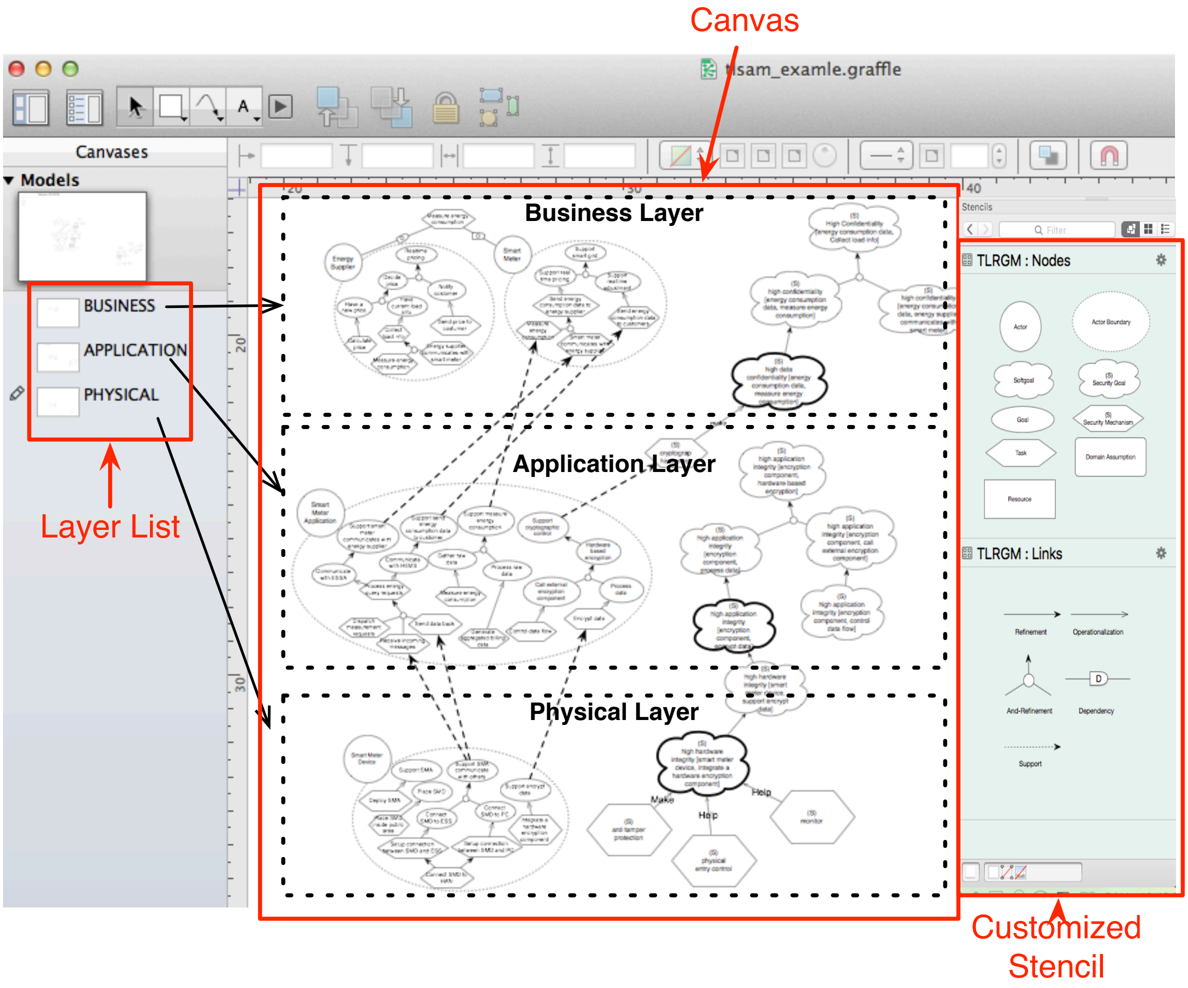


Figure 8.2: The modeling interface of MUSER

on such models, our prototype tool offers a collection of features to perform reasoning over the models, (semi-)automating corresponding analytical methods we have proposed in this thesis (Chapter 4-7). In particular, we list all the features below, which are grouped based on the analysis methods they implement.

Thus far, MUSER has been implemented with the following features:

- *Holistic security requirements analysis.*
  - *Refine security goals.* The tool can automatically perform both step-by-step refinement and exhaustive refinement, completing security goal refinement analysis (Section 4.3.1).
  - *Identify critical security goals.* The tool can automatically check the applicability of security goals and identify threats that are related to security goals.

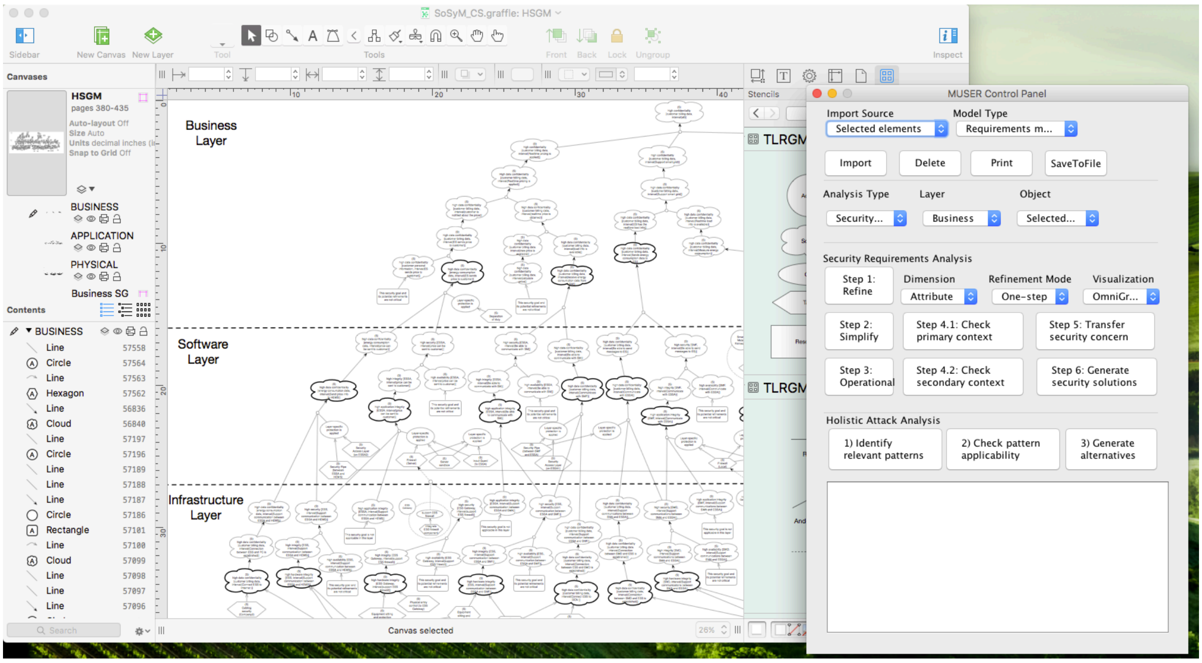
  Based on such analysis results, analysts can semi-automatically identify critical security goals (Section 4.3.2).

  - *Operationalize security goals.* The tool can help analysts to semi-automatically operationalize critical security goals in terms of corresponding security mechanisms using security patterns, facilitating the analysis described in Section 4.3.3. In particular, this feature includes the following sub-features, assisting elaborated analysis methods that are introduced in Chapter 6.
    - *Identify security pattern candidates.* Given a critical security goal, the tool can first identify all the relevant security patterns that may be applied to operationalize the security goal, assisting the analysis described in Section 6.3.1.
    - *Check applicability of security pattern candidates.* The tool can help analysts to determine the applicability of each security pattern candidate by checking the context required by the candidate. If certain context cannot be checked, the tool will interact with users to obtain corresponding context information. In such a way, analysts can semi-automatically select applicable security patterns from candidates, as described in Section 6.3.2.
  - *Propagate security requirements across layers.* The tool can automate the cross-layer analysis described in Section 4.3.4, i.e., generating security requirements in the next layer down and enabling iterative security requirements analysis in that layer.
  - *Generate holistic security solutions.* Once having the entire security goal model that covers security concerns in all three layers, our tool can generate all the alternative security solutions that satisfy the root security goal, automating the analysis described in Section 4.3.5.

- *Holistic security attack analysis.*
  - *Operationalize anti-goals into attack pattern candidates.* Given an anti-goal, the tool can identify all the relevant attack patterns that may be able to operationalize this anti-goal, automating the analysis presented in Section 5.3.2.
  - *Check applicability of attack pattern candidates.* The tool can automatically determine the applicability of an attack pattern by checking the context required by the pattern. If some context cannot be checked, the tool will ask for manual check via pop-up dialog. In such a way, analysts are able to semi-automatically select applicable attack patterns to operationalize anti-goals, as described in Section 5.3.2.

- *Generate of alternative (multistage) attacks.* Once a complete attack model is established, our tool can automatically generate all the alternative (multistage) attacks, supporting the analysis introduced in Section 5.3.2.

- *Security mechanism impact analysis.*
  - *Identify constrained requirements tasks.* The tool can automatically identify all the requirements tasks that are affected by a particular security constraint, assisting the impact analysis described in Section 7.5.
  - *Enforce impact of security mechanisms.* Once the constrained requirements tasks have been identified, our tool can automatically enforce the impact of the security constraints on such requirements tasks, based on corresponding rules introduced in Section 7.5.



Auto-layout Off
Size Auto
Units decimal inches
Snap to Grid Off

Import Source: Selected elements
Model Type: Requirements m...
Analysis Type: Security...
Layer: Business
Object: Selected...
Dimension: Attribute
Refinement Mode: One-step
Visualization: OmniGr...

Figure 8.3: Perform analysis using MUSER control panel

Since MUSER is developed on top of *OmniGraffle*, all the above analysis features are executed via the MUSER control panel, as shown in the right part of Fig. 8.3. In particular, analysts should first select elements from the canvas, which they want to analyze. After that, they can perform specific analysis by clicking corresponding buttons in the control panel. Once the analysis is completed, our tool will update the canvas in order to show the analysis results.

8.3 Use Cases

In this section, we present detailed use case specifications for each analysis feature we have introduced in the last section in order to guide potential users. In particular, we specify the following attributes for each use case: *Name*, *Actors*, *Preconditions*, *Basic Flow*, *Post-conditions*. Note that we here only present the basic execution flow of each use case, and ignore the alternative flows. This is because the tool currently is not robust enough to tackle different exceptions, and thus the users are expected to follow the basic execution flow. In the future, we will keep improving the tool to deal with exceptions.

Table 8.1: The use case specification for refining security goals

Name: Refine security goals
Actors: Analysts, Stakeholders
Preconditions: • Stakeholder's initial security needs (i.e., the root security goals) have been elicited • The three-layer requirements goal model has been built • The resource schema has been imported • The security property hierarchy has been specified
Basic Flow: 1. Select one or multiple security goals from the canvas, which are intended to be refined 2. Choose the refinement dimension in the control panel (e.g., asset-based refinement) 3. Choose the refinement type in the control panel (e.g., one-step refinement) 4. Click the refinement button in the control panel
Post-conditions: • The selected security goals as been refined into expected sub-goals

Table 8.2: The use case specification for identifying critical security goals

Name: Identify critical security goals
Actors: Analysts, Security experts
Preconditions: • The data flow information has been specified • The threat information has been imported • The resource schema has been imported
Basic Flow: 1. Select one or multiple security goals from the canvas 2. Click the simplification button in the control panel 3. Manually determine the criticality for each security goal that is applicable but non-critical.
Post-conditions: • The criticality of the selected security goals has been determined

Table 8.3: The use case specification for identifying security pattern candidates

Name: Identify security pattern candidates
Actors: Analysts
Preconditions: • Critical security goals have been identified
Basic Flow: 1. Select one or multiple critical security goals from the canvas 2. Click the operationalization button in the control panel
Post-conditions: • The selected security goals are operationalized into a list of security pattern candidates

Table 8.4: The use case specification for checking applicability of security pattern candidates

Name: Checking applicability of security pattern candidates
Actors: Analysts, Domain experts
Preconditions: • Security pattern candidates have been generated • The context specification has been loaded
Basic Flow: 1. Select one or multiple security pattern candidates from the canvas 2. Click the operationalization button in the control panel 3. For contexts that cannot be automatically checked, users need to manually check them via pop-up dialog
Post-conditions: • The applicability of the selected security pattern candidates have been determined

Table 8.5: The use case specification for propagating security requirements across layers

Name: Propagate security requirements across layers
Actors: Analysts
Preconditions: • Critical security goals have been operationalized • The three-layer requirements goal model has been built
Basic Flow: 1. Select one or multiple security mechanisms or critical security goals from the canvas 2. Click the cross-layer analysis button in the control panel
Post-conditions: • The selected security mechanisms or critical security goals have propagated their corresponding security concerns into the next layer down

Table 8.6: The use case specification for generating holistic security solutions

Name: Generate holistic security solutions
Actors: Analysts
Preconditions: • The three-layer security goal model has been built
Basic Flow: 1. Select the entire three-layer security goal model from the canvas 2. Click the holistic analysis button in the control panel
Post-conditions: • All holistic security solutions are generated and listed in the control panel

Table 8.7: The use case specification for operationalizing anti-goals

Name: Operationalize anti-goals into attack pattern candidates
Actors: Analysts
Preconditions: • The goal-oriented attack strategy model has been established
Basic Flow: 1. Select one or multiple leaf anti-goals from the canvas 2. Click the attack operationalization button in the control panel
Post-conditions: • The selected anti-goals have been operationalized into a list of relevant attack patterns

Table 8.8: The use case specification for checking applicability of attack pattern candidates

Name: Check applicability of attack pattern candidates
Actors: Analysts, Domain experts
Preconditions: • Attack pattern candidates have been generated • The three-layer requirements goal model has been built
Basic Flow: 1. Select one or multiple attack pattern candidates 2. Click the attack context analysis button in the control panel 3. For contexts that cannot be automatically checked, users need to manually check them via pop-up dialog
Post-conditions: • The applicability of the selected attack pattern candidates have been determined

Table 8.9: The use case specification for generating alternative (multistage) attacks

Name: Generate of alternative (multistage) attacks
Actors: Analysts
Preconditions: • The goal-oriented attack model has been established
Basic Flow: 1. Select the entire goal-oriented attack model from the canvas 2. Click the attack generation button in the control panel
Post-conditions: • All attack alternatives are generated and shown in the control panel

Table 8.10: The use case specification for identifying constrained requirements tasks

Name: Identify constrained requirements tasks
Actors: Analysts
Preconditions: • The enriched requirements model has been built • The applied security tasks have been contextualized
Basic Flow: 1. Select one or multiple security constraints from the canvas 2. Click influence analysis button in the control panel
Post-conditions: • All the requirements tasks that are impacted by the selected security constraints are identified

Table 8.11: The use case specification for enforcing impact of security constraints

Name: Enforce impact of security constraints
Actors: Analysts
Preconditions: • The requirements tasks that are constrained by the security constraints
Basic Flow: 1. Select one or multiple security constraints from the canvas 2. Click the enforce impact button in the control panel
Post-conditions: • The impact of the security constraints has been enforced on the constrained requirements tasks

8.4 Chapter Summary

In this chapter, we introduce a prototype tool (MUSER), which was developed to support various analysis methods we have proposed in Chapter 4-7. The prototype tool is implemented in Java, its architecture has been introduced in detail in Section 8.1. We then present a list of features that are accommodated by the tool in Section 8.2. In order to facilitate users to easily use the tool, in Section 8.3, we have described detailed use cases for each analysis feature of the tool.

Chapter 9

Validation

> *We need science. We need empirical evidence. We can't just use mathematical reasoning to deduce the nature of the world.*
>
> Rebecca Goldstein

In this Chapter, we validate our holistic security requirements framework by applying it to two case studies. The first case study was performed by myself based on a smart grid scenario, in which I focused on evaluating the efficiency of both the three-layer requirements modeling language and the holistic analysis framework. The detailed research questions are described in Section 9.1. In particular, we first describe this scenario in detail in Section 9.1.1. After that we evaluate the modeling of the three-layer requirements goal model in Section 9.1.2, and report our experiences in performing the holistic security requirements analysis and evaluate the analysis results in Section 9.1.3. It is worth noting that, during this case study, we adopt the approaches proposed in Chapter 5 and Chapter 6 (i.e., holistic attack analysis and security pattern analysis) to produce models/information required by the holistic security requirements framework, supporting corresponding analysis. Lastly, we discuss the threats to validity in this case study.

The second case study was performed by a Master student based on a large-scale medical emergency response system. The focus of this case study is to evaluate whether our approach has the potential to be adopted in reality by people who were not involved in the development of the approach. We first describe the scenario in detail in Section 9.2.2 Then, we present the three-layer requirements model constructed by the student and evaluate usability of our modeling language in Section 9.2.2. Based on that model, in Section 9.2.3, we describe and evaluate the holistic security requirements analysis results.

9.1 Case Study 1: A Smart Grid Real-Time Pricing Scenario

In this section, we validate our approach by applying it to a real-time pricing scenario of smart grid advanced metering infrastructure, which is a typical STS. In particular, we intend to investigate scalability and efficacy of our approach, each of which corresponds to one research question as presented below. In addition, we have further decomposed these questions into fine-grained ones, which can be measured and answered during the case study.

- ***RQ1***: *Is our approach scalable for large-scale STSs?*
    - ***RQ1.1***: *Can we efficiently construct three-layer requirements models for large-scale STSs?*
        - ***RQ1.1.1***: *Is it easy to collect domain information for constructing the models?*
        - ***RQ1.1.2***: *How much effort is required to build the models?*
    - ***RQ1.2***: *Can we efficiently perform holistic security analysis for large-scale STSs?*
        - ***RQ1.2.1***: *Is additional security knowledge/expertise required?*
        - ***RQ1.2.2***: *Are there any difficulties in applying the analysis methods?*
        - ***RQ1.2.3***: *How much effort is required to perform the security analysis?*
- ***RQ2***: *To what extent can our approach is effective for holistic security protection?*
    - ***RQ2.1***: *Can holistic security solutions be identified?*
    - ***RQ2.2***: *How many security solution alternatives are identified?*

This case study was performed based on the information of the scenario we have collected from literature [NIST, 2012; Cuellar and Suppan, 2013], which is detailed in Section 9.1.1. In particular, we first constructed a three-layer requirements model, which captured requirements of different artifacts involved in this scenario, as well as connections among them (Section 9.1.2). On the basis of this three-layer requirements model, we applied our holistic security analysis framework step by step, establishing a three-layer security goal model. On the basis of this model, we explored alternative security solutions that can provide comprehensive protections to the entire system (Section 9.1.3). It is worth noting that, at the end of Section 9.1.2 and Section 9.1.3, we also evaluate corresponding research questions. Finally, we discuss the threats to validity and corresponding countermeasures in Section 9.1.4.

9.1.1 Scenario Description

Real-time pricing is a scenario based on the smart grid advanced metering infrastructure, in which the energy supplier collects real-time energy consumption data and balances loads on the power grid. This scenario presents a typical socio-technical system. Firstly, it includes a business process for price generation. Specifically, the energy supplier periodically collects load information of the power grid, and generates appropriate energy prices accordingly in order to regulate the load of the power grid. On the other side, the energy consumers will adjust their energy usage based on the real-time prices. Secondly, a number of applications are involved in this scenario to support the interactive process. In particular, a home energy management system is used by the energy consumer to communicate with the energy supplier and control the smart appliances in her apartment. Thirdly, physical devices (e.g., personal PC) are required to deploy the software applications, and the network needs to be appropriately configured to support communications. Detailed information of this scenario can be found in [Cuellar and Suppan, 2013]. As our approach requires particular information of the smart grid scenario across different layers, we also refer to other complementary sources of information (e.g., [NIST, 2012]).

As this system involves a wide range of artifacts that vary from business processes to physical devices, it is difficult to carry out a thorough security analysis to protect the entire system. As reported by the National Vulnerability Database, on average, 15 new vulnerabilities of the Supervisory Control And Data Acquisition (SCADA, a system for remotely monitoring and controlling power grids) are publicly disclosed each day. Not surprisingly, the presence of these vulnerabilities leads to many attacks on smart grid systems [Flick and Morehouse, 2010]. As such, our approach aims to provide a systematic and holistic way to analyze security requirements for all parts of the system.

9.1.2 Building Three-Layer Requirements Models

As the first step of the case study, we built the three-layer requirements model for the smart grid by following the conceptual model proposed in Section 4.2.

We constructed the requirements models layer by layer, starting from the social layer. Since we have separated concerns into different layers, in this first layer, we exclusively focused on the social and business aspects of the scenario, while ignored technical issues. In particular, we modeled three social actors *Energy Supplier*, *Customer*, and *Smart Meter*, where *Customer* depends on *Energy Supplier* for sending the real-time price of energies, and *Energy Supplier* depends on *Smart Meter* for measuring energy consumption information. Fig. 9.1 shows the requirements model we have built for the social layer. When elaborating the requirements model inside each actor, we respected the operational def-

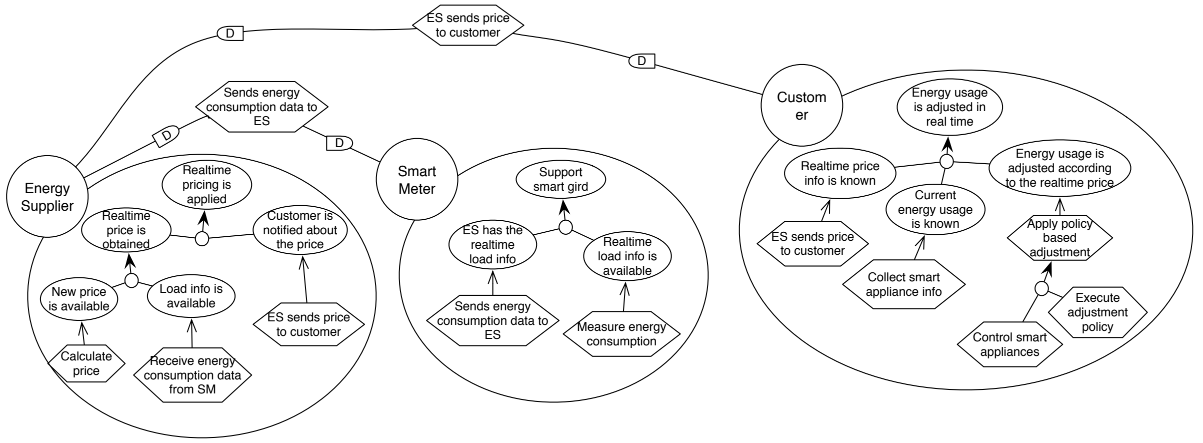

ES sends price to customer
D
D
Sends energy consumption data to ES
D
D
Energy Supplier
Realtime pricing is applied
Realtime price is obtained
Customer is notified about the price
New price is available
Load info is available
ES sends price to customer
Calculate price
Receive energy consumption data from SM
Smart Meter
Support smart gird
ES has the realtime load info
Realtime load info is available
Sends energy consumption data to ES
Measure energy consumption
Custom er
Energy usage is adjusted in real time
Realtime price info is known
Energy usage is adjusted according to the realtime price
Current energy usage is known
ES sends price to customer
Apply policy based adjustment
Collect smart appliance info
Execute adjustment policy
Control smart appliances

Figure 9.1: The requirements model in the social layer

inition we have assigned to the social layer. For example, we iteratively elaborated the business goal *Realtime pricing is applied* until obtaining concrete business process activities that are performed within this scenario, such as *Calculate price*. Benefiting from the operational definition, we are able to know when to finish the requirements modeling in the social layer. It is worth noting that we here followed a top-down process to create the requirements model, however, it can also be performed in a bottom-up manner (or a hybrid way) by using existing business process models, if available. In particular, we can first model each business process activity as a task in the social layer, and then identify goals achieved by such tasks by asking "why" questions.

Once the requirements model in the social layer was finished, we then analyzed support links between the social layer and the to-be-created application layer, following the instructions we presented in Section 4.2.1. Specifically, we checked whether each leaf-task is purely performed by people. If not, we then identified the software applications involved in this activity, based on which we modeled elements of the application layer including supporting links relating the two layers. For example, based on the domain specification, we identified that the task *Receive energy consumption data from SM* is executed by using *Energy Supplier Server Application*. Thus, we modeled this application as an actor in the software application layer, and further identified a requirements goal of the application, i.e., *Be able to communicate with SM*, which supports the task in the social layer.

By analyzing the support links between layers, we obtained a list of application actors, each of which has one or several requirements goals. Based on such models, we then elaborated requirements for each actor in the software layer, until reaching detailed application functions (i.e., the operational definition of software tasks). Similar to the analysis in the social layer, the construction of requirements model in this layer can be done via a bottom-up manner, if there are software architecture models available for use as inputs. Once the requirements model in the software layer was finished, we then moved to the infrastructure layer, building this requirements model in a similar way. Eventually, we obtained the complete three-layer requirements model of the smart grid scenario, shown in Fig. 9.2[1].

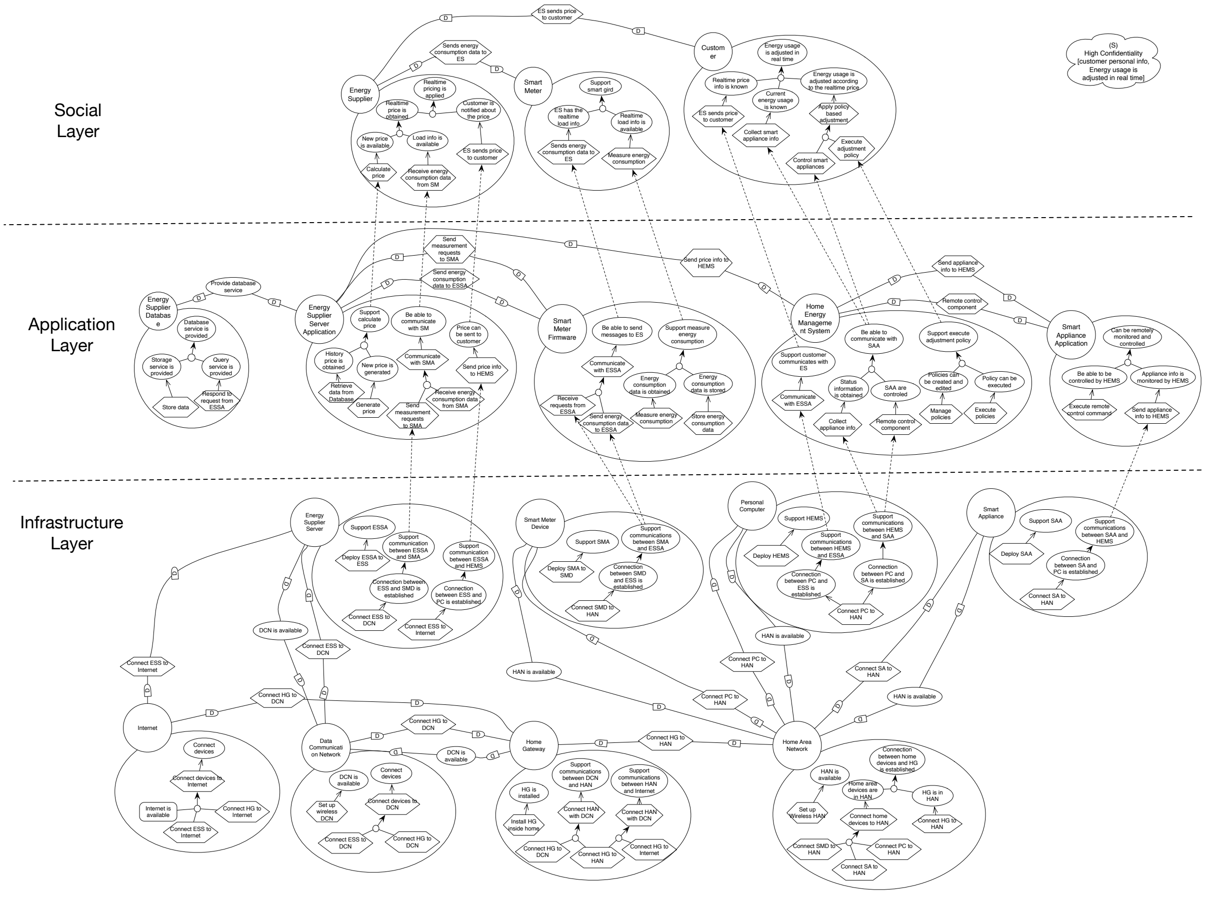

Social Layer
Application Layer
Infrastructure Layer
D
ES sends price to customer
Sends energy consumption data to ES
Energy Supplier
Realtime pricing is applied
Realtime price is obtained
Customer is notified about the price
New price is available
Load info is available
ES sends price to customer
Calculate price
Receive energy consumption data from SM
Smart Meter
Support smart gird
ES has the realtime load info
Realtime load info is available
Sends energy consumption data to ES
Measure energy consumption
Custom er
Energy usage is adjusted in real time
Realtime price info is known
Energy usage is adjusted according to the realtime price
Current energy usage is known
ES sends price to customer
Apply policy based adjustment
Collect smart appliance info
Execute adjustment policy
Control smart appliances
(S) High Confidentiality [customer personal info, Energy usage is adjusted in real time]
Send measurement requests to SMA
Send energy consumption data to ESSA
Provide database service
Energy Supplier Databas e
Database service is provided
Storage service is provided
Query service is provided
Store data
Respond to request from ESSA
Energy Supplier Server Application
Support calculate price
Be able to communicate with SM
Price can be sent to customer
History price is obtained
New price is generated
Communicate with SMA
Send price info to HEMS
Retrieve data from Database
Generate price
Send measurement requests to SMA
Receive energy consumption data from SMA
Send price info to HEMS
Smart Meter Firmware
Be able to send messages to ES
Support measure energy consumption
Communicate with ESSA
Energy consumption data is obtained
Energy consumption data is stored
Receive requests from ESSA
Send energy consumption data to ESSA
Measure energy consumption
Store energy consumption data
Send appliance info to HEMS
Remote control component
Home Energy Manageme nt System
Support customer communicates with ES
Be able to communicate with SAA
Support execute adjustment policy
Communicate with ESSA
Status information is obtained
SAA are controled
Policies can be created and edited
Policy can be executed
Collect appliance info
Remote control component
Manage policies
Execute policies
Smart Appliance Application
Can be remotely monitored and controlled
Be able to be controlled by HEMS
Appliance info is monitored by HEMS
Execute remote control command
Send appliance info to HEMS
Energy Supplier Server
Support ESSA
Support communication between ESSA and SMA
Support communication between ESSA and HEMS
Deploy ESSA to ESS
Connection between ESS and SMD is established
Connection between ESS and PC is established
Connect ESS to DCN
Connect ESS to Internet
DCN is available
Connect ESS to DCN
Connect ESS to Internet
Smart Meter Device
Support SMA
Support communications between SMA and ESSA
Deploy SMA to SMD
Connection between SMD and ESS is established
Connect SMD to HAN
HAN is available
Personal Computer
Support HEMS
Support communications between HEMS and ESSA
Support communications between HEMS and SAA
Deploy HEMS
Connection between PC and ESS is established
Connection between PC and SA is established
Connect PC to HAN
HAN is available
Connect PC to HAN
Connect PC to HAN
Smart Appliance
Support SAA
Support communications between SAA and HEMS
Deploy SAA
Connection between SA and PC is established
Connect SA to HAN
Connect SA to HAN
HAN is available
Internet
Connect HG to DCN
Connect devices
Connect devices to Internet
Internet is available
Connect HG to Internet
Connect ESS to Internet
Data Communicati on Network
Connect HG to DCN
DCN is available
Connect devices
Set up wireless DCN
Connect devices to DCN
Connect ESS to DCN
Connect HG to DCN
Home Gateway
Connect HG to HAN
HG is installed
Support communications between DCN and HAN
Support communications between HAN and Internet
Install HG inside home
Connect HAN with DCN
Connect HAN with DCN
Connect HG to DCN
Connect HG to HAN
Connect HG to Internet
Home Area Network
Connection between home devices and HG is established
HAN is available
Home area devices are in HAN
HG is in HAN
Set up Wireless HAN
Connect home devices to HAN
Connect HG to HAN
Connect SMD to HAN
Connect PC to HAN
Connect SA to HAN

Figure 9.2: A three-layer requirements model of the smart grid realtime pricing scenario

Evaluation. I spent around four days gathering information from different sources. Most available specifications do not provide information required for all of the layers, as they were not originally developed for holistic analysis. Therefore, we have to synthesize such information in order to gather a complete understanding of the system. Additional effort is required in order to make information from different sources consistent. Overall, in this

---

[1] The full model file, `http://disi.unitn.it/~li/SoSyM/model_rtp.pdf`

Table 9.1: Statistics of the three-layer requirements model (smart grid)

Layer	Actor	Goal	Task	(and)Refine	Operationalize	Dependency
All	16	60	57	44	43	21
Social	3	12	9	11	8	2
Software	5	22	20	14	16	6
Infrastructure	8	26	28	19	19	13

case study, it was not easy to collect related domain information (***RQ1.1.1***).

The main reason for this difficulty is the method of data collection. According to Lethbridge et al. [2005], data collection technique can be classified into three levels: directly interacting with subjects to collect data in real time; directly collecting raw data without interacting with subjects; reusing available information from other independent studies. Our data collection belongs to the third level, which leads to the difficulties that we have to synthesize information from different sources while make them consistent We argue the other two levels can avoid those difficulties and simplify the data collection, which we plan to adopt in subsequent studies.

After collecting and understanding sufficient information for this scenario, I took one day to construct the three-layer requirements model (***RQ1.1.2***), as shown in Fig. 9.2. The statistics of the model is presented in Table 9.1, which shows the entire model contains 133 nodes and 108 links. Thanks to the conceptually divided layers, the modeler only needed to take into account layer-specific concerns when constructing models for a particular layer. In addition, the prototype tool can easily show/hide particular layers, enabling us to quickly switch views between a specific layer and the entire model. In particular, as we build our tool on top of a professional diagramming tool, our prototype tool inherits good usability of that tool and thus easily facilitates the graphical modeling task. Overall, because of both the conceptually separated layers and the prototype tool, we believe a typical analyst is able to tackle the complexity of STSs and build the three-layer requirements model once related information has been collected and available (***RQ1.1***).

9.1.3 Analyze Security Requirements in Three Layers

Having the above three-layer requirements model as input (Fig. 9.2), we applied our approach step by step to generate the holistic security goal model, starting from the root security goal. In particular, we started from analyzing a high-level security need of stakeholders, i.e., protecting confidentiality of customer information during the time interval of applying realtime pricing. We iteratively performed security requirements analysis throughout all three layers in order to construct a holistic security goal model, based on which we generated a collection of holistic security solutions that satisfy the root security goal. In the remaining part of this subsection, we report our experiences about the application of our approach, as well as evaluate our approach based on the research questions.

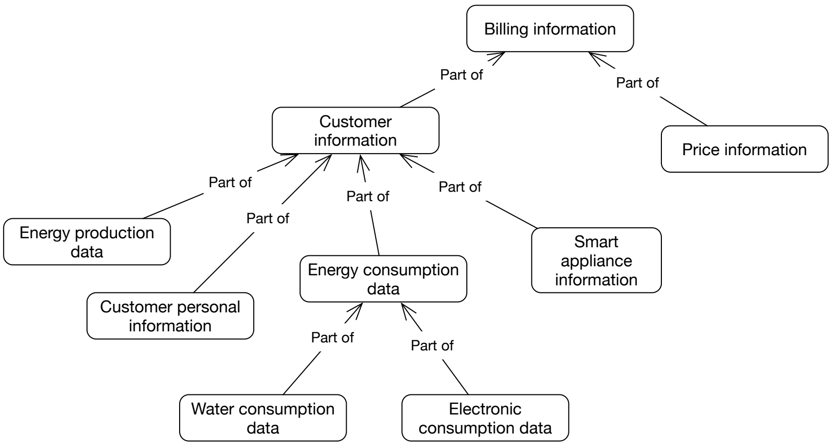


Figure 9.3: A full resource schema considered in this case study

Security goal refinement and simplification. Given a high-level security goal, we need to refine it until we are able to identify critical security goals. We applied a hybrid refinement strategy (as described in Section 4.3.1), leveraging the advantages of both the step-by-step strategy and the exhaustive strategy. We first adopt the step-by step strategy to elaborate an initial security goal into more fine-grained goals, in order to better understand the exact security needs of stakeholders. In particular, we followed an intuitive order for performing the refinement analysis, i.e., first refine via security properties (reference to the taxonomy in Fig. 4.5), then via assets (reference to a full resource schema shown in Fig. 9.3), and finally via interval (reference to Fig. 9.2). Our prototype tool can automate such refinement and pop up an alert if one dimension cannot be refined anymore, i.e., achieving the bottom elements in the reference model.

After each refinement, we evaluated the results by asking *"are all the elaborated security goals needed by stakeholders?"*. If a refined security goal is not needed by stakeholders, then it was excluded from subsequent analysis. In particular, we here used our collected knowledge of the case to answer the above question from stakeholder's perspective. As shown in the top part of Fig. 9.4, after elaborating security goal *SG2* via the asset dimension, we found out the stakeholder's exact security need is to protect the confidentiality of *customer personal information*, and the stakeholder does not care about other parts of *customer information*, e.g., *smart appliance information*. Thus, in the subsequent refinement analysis, we exclusively focus on elaborating the security goal *SG4*.

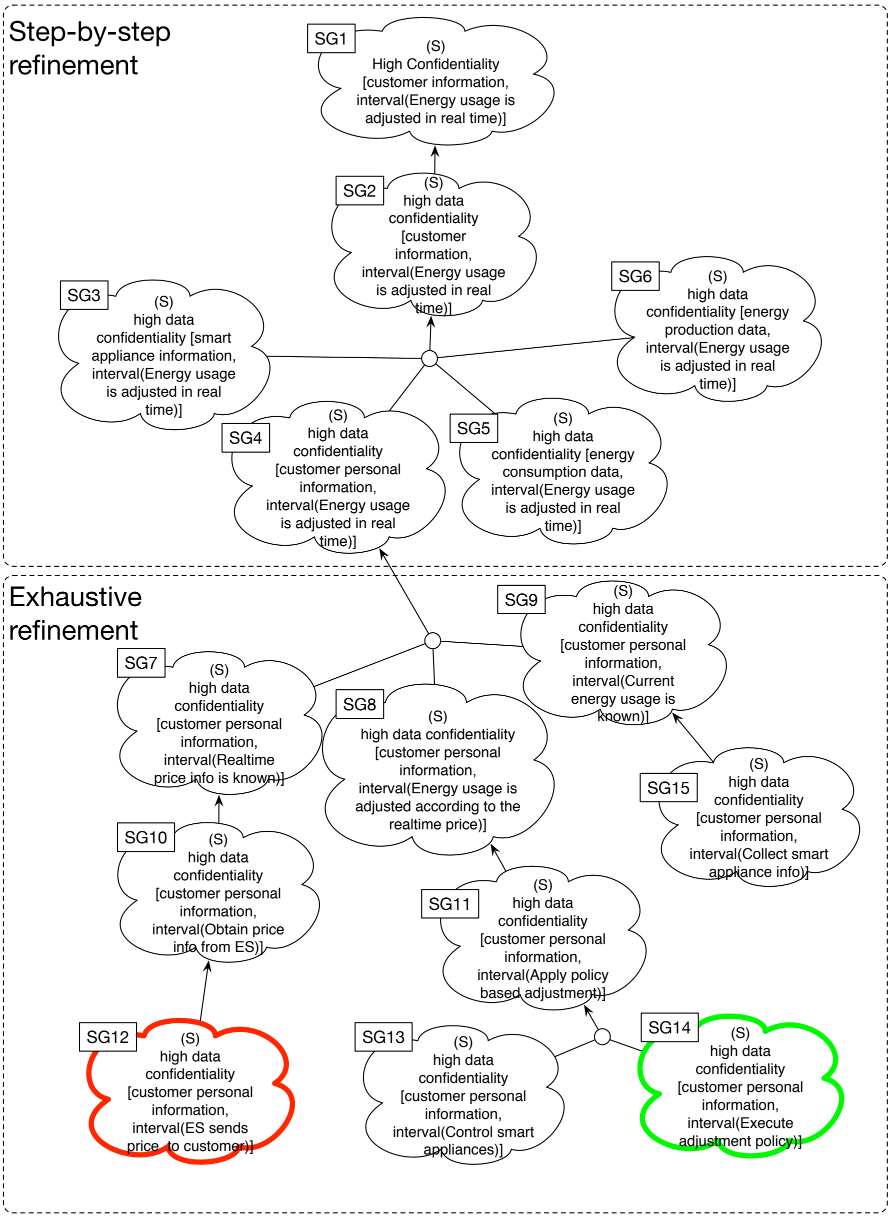

Step-by-step refinement
SG1 (S) High Confidentiality [customer information, interval(Energy usage is adjusted in real time)]
SG2 (S) high data confidentiality [customer information, interval(Energy usage is adjusted in real time)]
SG3 (S) high data confidentiality [smart appliance information, interval(Energy usage is adjusted in real time)]
SG6 (S) high data confidentiality [energy production data, interval(Energy usage is adjusted in real time)]
SG4 (S) high data confidentiality [customer personal information, interval(Energy usage is adjusted in real time)]
SG5 (S) high data confidentiality [energy consumption data, interval(Energy usage is adjusted in real time)]
Exhaustive refinement
SG9 (S) high data confidentiality [customer personal information, interval(Current energy usage is known)]
SG7 (S) high data confidentiality [customer personal information, interval(Realtime price info is known)]
SG8 (S) high data confidentiality [customer personal information, interval(Energy usage is adjusted according to the realtime price)]
SG15 (S) high data confidentiality [customer personal information, interval(Collect smart appliance info)]
SG10 (S) high data confidentiality [customer personal information, interval(Obtain price info from ES)]
SG11 (S) high data confidentiality [customer personal information, interval(Apply policy based adjustment)]
SG12 (S) high data confidentiality [customer personal information, interval(ES sends price to customer)]
SG13 (S) high data confidentiality [customer personal information, interval(Control smart appliances)]
SG14 (S) high data confidentiality [customer personal information, interval(Execute adjustment policy)]

Figure 9.4: Refine security goals in the social layer

As we decided that the security goal *SG4* can reflect the stakeholder's needs, we performed exhaustive refinement analysis to explore all possible refinements of *SG4*. Once the exhaustive refinements were obtained, we then performed the simplification analysis over all the exhaustively refined security goals (Section 4.3.2), which was automated by the prototype tool. It is worth noting that the simplification analysis requires threat knowledge about the smart grid system, which we imported from our holistic attack analysis as introduced in Chapter 5.

The results of the simplification analysis are shown in the bottom part of Fig. 9.4, where *SG12* was identified as a critical security goal (highlighted in red), *SG14* was identified as an applicable one (highlighted in green) that requires manual assessment for its criticality, and all other security goals are not applicable and should be excluded from subsequent analysis. As *SG14* expresses the security need for protecting data confidentiality of customer personal information during the energy usage adjustment, we assessed that it is very likely to be threatened by threats originate from lower-layers and thus classified it as a critical security goal.

Security goal operationalization. Once critical security goals are identified, we performed the security pattern-based operationalization analysis (as elaborated in Section 6.3) to operationalize the critical security goals. We first automatically generated a list of security pattern candidates for each critical security goal, and then check the context of each candidate in order to determine whether they are applicable. In particular, among the current collection of the selected security patterns in the social layer (see Fig. 4.11 in Chapter 4), there was only one security pattern (i.e., *access control*) can be applied for tackling *data confidentiality.* As such, each of the two critical security goals identified from the last step (as shown in Fig. 9.4) was first operationalized by an *access control* mechanism. Then, we checked the context of *access control* which is "*Any environment in which we have resources whose access needs to be controlled*" [Asnar et al., 2011a]. Note that for our approach, such context holds by default, which can be inferred by the particular attributes of security goals (i.e., *security property* and *asset*). As such, according to our pattern analysis method, this *access control* pattern was applicable.

Cross-layer analysis. Once we finished operationalization analysis in one layer, we then performed the cross-layer analysis (Section 4.3.4). Based on cross-layer links modeled in the three-layer functional goal model, we automatically performed such analysis which transferred security concerns to lower layers, targeting corresponding system components there (e.g., software applications, physical devices). In particular, the critical security goal *SG12 high data confidentiality [customer personal information, interval(ES sends price to customer)]* (Fig. 9.4) was and-refined into two security goals in the application layer: one goal concerned the same asset and security property with *SG12*, but focused on a particular interval when the supporting application sends the price to customer; the other security goal exclusively concerned the security of the supporting application (i.e., *energy* *supplier server application*) instead of the original asset (i.e., *customer personal information*). Similarly, we performed such cross-layer analysis for another critical security goal *SG14* (Fig. 9.4). We did not encounter any difficulties at this step.

Holistic security solution analysis. By iteratively performing the security analysis in each of the three layers, we finally ended up with an entire holistic security goal model, which is shown in Fig. 9.5 (The full model file can be found online[2]). We performed the backwards satisfaction analysis over this holistic security goal model in order to identify all possible holistic security solutions for the entire system (Section 4.3.5), resulting in 21 holistic security solutions in total.

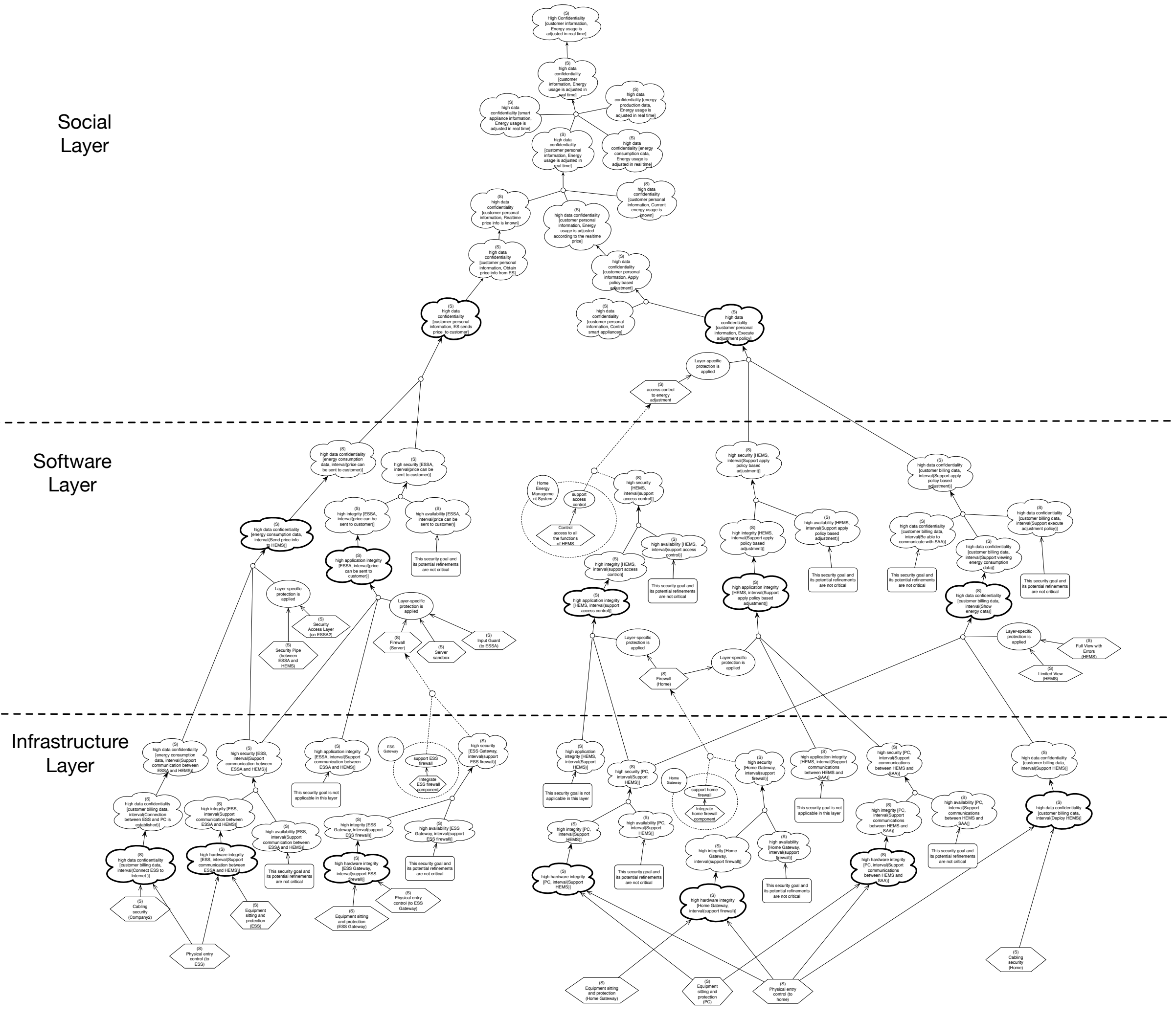


Figure 9.5: The entire security goal model across three layers

---

[2] `http://disi.unitn.it/~li/thesis/validation_hsgm.pdf`

Each of these holistic security solutions consists of specific solutions from different layers, covering different system components. For example, we list one holistic security solution below, which consists of ten security mechanisms (each mechanism is a solution introduced by a security pattern). In particular, there is one security mechanism in the social layer, four security mechanisms in the software layer, and five security mechanisms in the infrastructure layer. It is worth noting that the satisfaction of the stakeholder's root security goal requires all these ten security mechanisms to be implemented, which may cost a large amount of money. This is mainly because we have taken a holistic viewpoint, considering all the related systems components.

An exemplary solution (10 mechanisms):

- *Social Layer: Access control to energy adjustment*
- *Software Layer: Firewall (Home), Security Pipe (between ESSA and HEMS), Server sandbox, Limited View (HEMS)*
- *Infrastructure Layer: Cabling security (Home), Equipment sitting and protection (PC), Equipment sitting and protection (Home Gateway), Equipment sitting and protection (ESS), Cabling security (Company2)*

Given the alternative holistic security solutions, we need to select the best alternative in subsequent analysis. To this end, we can elicit both the positive and negative influences of security patterns from their specification (e.g., [Fernandez-Buglioni, 2013]), as detailed in Chapter 6. Based on such information, different goal model selection algorithms can be applied to evaluate the solutions against desired system goals and qualities, allowing us to choose the best alternative [Horkoff and Yu, 2013]. This part of analysis is not covered in this case study.

Evaluation and reflection. Throughout the entire case study, we realized that additional security expertise was required in several places (***RQ1.2.1***). Firstly, during the simplification analysis, for security goals that have been classified as applicable but not critical by our inference rules, analysts need to manually assess their criticality. In particular, analysts should assess whether the security goal might be threatened by threats originate from lower-layers, in which the expertise of analysts can affect the assessment results. If an analyst has little security knowledge, she can adopt a conservative strategy, i.e., treating all applicable security goals as critical and further analyze them in the lower layers. However, as a result, the complexity of subsequent analysis can be increased. Secondly, during the security pattern analysis, we noticed the need of manual check over the pattern analysis results, requiring additional security knowledge. Specifically, when determining whether a security pattern is applicable to operationalize a security goal, we should take into account the interval of the security goal which is normally not captured in the pattern context. For example, in our case (Fig. 9.4), security goal *SG12* focuses on protecting the data confidentiality of customer personal information during the time interval *ES sends price to customer*, during which the *access control* mechanism is inappropriate to apply. It is worth noting that the need of performing manual check over the pattern analysis results is due to the incomplete context description of security patterns. Thus, such problem can be relieved along with the improvements of security patterns.

During the case study, we discovered two difficulties in performing our analysis (***RQ1.2.2***). Firstly, when performing the security goal refinement, we noticed that refining a security goal in the social layer is more complicated compared to the analysis in other layers, as it involves more manual analysis. In the social layer, we started from a high-level security goal, which is by nature broad and may not reflect the stakeholder's real needs. Thus, we needed to adopt a hybrid refinement strategy, which first uncovers the stakeholder's security needs more precisely. In the software layer and the infrastructure layer, the root security goals were derived from the security concerns in the upper layers, which had already captured stakeholder's more precise security needs. As such, we can directly apply the exhaustive refinement strategy, which is fully automated by our tool. Secondly, we have realized that our current collection of security patterns were not enough to cover all threats we analyzed in the case study. Specifically, in our case study, the critical security goal *SG12* we identified in the social layer (Fig. 9.4) was threatened by a social threat *"Inappropriately post customer data to public media"*, which should be tackled by security solutions, such as *security training*. However, our set of security patterns (Fig. 4.10) only included *access control* as a candidate solution to this problem, which is not applicable in this case. Considering this challenge, we plan to further enrich the collection of selected security patterns in the future.

After collecting all required information and constructing the three-layer requirement goal model, with the support of our tool, I spent six hours to perform the holistic security analysis (***RQ1.2.3***). Regarding the scale of this scenario (Table 9.1), the time span is reasonable. Although there were several difficulties we encountered during the case study (as discussed above), overall, we argue that our approach can be efficient for dealing with holistic security requirements analysis (***RQ1.2***).

As we have specified and illustrated before, our case study finally resulted in 21 holistic security solutions in total (***RQ2.2***), each of which consists of specific solutions from different layers, covering different system components (***RQ2.1***). Thus, we argue our security analysis results have a good coverage of security concerns and effectively contribute to holistic security protection.

An important advantage of our approach is systematically guiding analysts through a comprehensive security analysis process, covering different layers of an STSs. During this process, we have observed that the support links modeled in the three-layer requirements model are especially important for connecting security analysis across layers. With the tool support, the entire analysis can be performed in a fast and reliable way. In addition, by capturing and arranging security requirements in a holistic security goal model, we can identify alternative security solutions. Since implementing holistic security solutions is likely to be expensive, it is particularly important to identify and evaluate all the possible alternatives upfront, among which analysts can select the most appropriate solution within their budget.

Due to the intrinsic complexity and heterogeneity of STSs, the holistic security analysis involves a wide spectrum of security issues, which requires a large amount of security knowledge and thus makes the analysis even more complicated. As a response to this challenge, our approach has a particular focus on knowledge reuse, facilitating this knowledge-intensive analysis. Firstly, we have encapsulated part of required security knowledge into corresponding inference rules, which can be automatically inferred by the prototype tool. It is worth noting the inference rules we defined in this paper are domain-independent, which can be applied to all types of systems. In the future, we can incrementally define more inference rules to capture domain-specific knowledge in order to facilitate analysis in the corresponding domains. Secondly, we leverage reusable security patterns to help analysts to effectively generate proved security solutions that operationalize security goals. Although the entire analysis still requires some security expertise (most of which is domain-specific), this is difficult to avoid all together. Based on our case study, we argue that analysts with basic security knowledge are able to apply the security analysis within a reasonable time. As a future work, we plan to perform case studies with more participants to further evaluate this claim.

Through the case study, we have identified factors that can affect the quality of the analysis results, which should be carefully improved in future work. In particular, the collection of selected security patterns directly determines the holistic security solutions we can produce at the end of our security analysis. We have identified in the case study that the current set of security patterns used by our approach is not enough for tackling some threats identified in the case study, in the future, we will incorporate more patterns so as to have a better coverage.

9.1.4 Threats to Validity

As there are different ways of classifying validity in the literature, we here adopt the classification used by Runeson and Höst [2009], which has an focus on case study research in software engineering. In particular, we discuss interval validity, external validity, construct validity, and reliability, respectively.

Internal validity. Internal validity considers the causal relations between factors investigated in the case study. When evaluating the difficulty of modeling three-layer requirements models, we focused on whether the modeler is able to conceptually build the three-layer requirements model. However, in practice, both the utility of the modeling tool and the analyst's modeling experiences with the tool can affect the overall difficulty of this modeling task, which introducing a threat to internal validity. Although our tool is developed on top of a professional diagramming tool which has very good usability, people who are not familiar with this tool or have little experiences with modeling may encounter difficulties when building the three-layer requirements model. In the future, we intend to evaluate the difficulty of conceptually modeling the three-layer model and the difficulty of graphically modeling the three-layer model, respectively.

In addition, as our approach relies on external security knowledge (i.e., security pattern) for generating security solutions, the effectiveness of our analysis results is not only determined by our approach, but also by the security patterns used by our approach. Although we have already incorporated 21 security patterns in our approach, covering security solutions in different layers, we have already noticed that current selection of security patterns is not enough for dealing with threats that we have identified in this case study. As a result, in subsequent research, we will incrementally incorporate additional security patterns in our approach, keeping the reused security knowledge with the recent advances in security patterns.

External validity. External validity is concerned with to what extent it is possible to generalize the findings of our case study. Because of the inherent complexity of STSs (e.g., our analysis model contains more than two hundred elements), a holistic security requirements analysis takes a considerable amount of effort. As a result, in this case study we only focus on one particular security goal, imposing a threat to external validity. To tackle this threat, in the future, we plan to analyze more security goals concerning the same scenario. In addition, considering the reason of this threat (i.e., the overall complexity of the case study), we plan to separate the validation into two parts in order to reduce the inherent complexity of the holistic security requirements analysis. Specifically, the first part will focus on constructing the three-layer requirements model of STSs, i.e., the validation described in Section 9.1.2; and the second part will focus on performing holistic security requirements based on the three-layer requirements model, as described in Section 9.1.3.

Another threat concerns the security background of the analyst. The case study reported in this paper is performed by me, who is the method developer and has related security expertise. However, we have not evaluated our approach with participants with little security knowledge. As such we do not have practical evidence yet that shows our approach can be applied by those who are not experts in security, although our approach does incorporate security patterns to help analysts to reuse security knowledge. Therefore, we intend to further evaluate our approach with people who is not involved in the method development and has limited security knowledge, observing differences in their performance when compared to this case study. Driven by this need, in our subsequent research, we have further evaluates our approach with a Master student who meets the above criteria. We will report our subsequent study in detail in the next section.

Construct validity. Construct validity concerns to which degree a test measures what it claims to be measuring. In our case study, we measure both the number and coverage of holistic security solution alternatives, based on which we determine the effectiveness of our analysis results. We acknowledge the quality of the obtained security solutions is a further factor, which may play a role in assessing the effectiveness of our approach. As a countermeasure to this threat, we intend to have security experts to assess the quality of the generated holistic security solutions.

Reliability. Reliability is concerned with to what extent the data and the analysis are dependent on the specific researchers. Since the case study is performed by only one person, there is a threat to reliability of the analysis results. To tackle this issue, in the future, we first plan to have peer researchers to evaluate the analysis results. In addition, we will plan to include more than one analysts who work together to apply our approach.

9.2 Case Study 2: Medical Emergency Response System

In this section, we report another case study in which we applied our holistic security requirements framework to a medical emergency response system. This case study is grounded on a Master thesis [Robin, 2015]. The Master student was first taught the three-layer security requirements modeling language, and then used the prototype to perform the holistic security requirements analysis under the supervision of Prof. Mylopoulos and myself. In particular, the focus of this case study is preliminarily assessing whether our approach can be adopted by people who were not involved in the development of our approach and has limited security knowledge.

9.2.1 Scenario Description

The medical emergency response system we analyzed in this study is based on the technical report Serenity-Consortium [March 2007], taken from an EU project *SERENITY*[3]. The

---

[3] `http://eu-serenity.sourceforge.net/`

essential components of this system are smart items, which can measure data and exchange it with other smart equipments. For example, a smart item equipped with an infrared thermometer measures the temperature of a patient and transmits such information to medical emergency system via its communication interfaces. In such a way, different smart items that monitor particular parameters (such as temperature, sound, vibration, pressure, and motion) can be connected in a flexible network, delivering real-time data of patients to the medical emergency system.

The medical emergency system is subject to an increasing number of attacks, e.g., illegal disclosure of information, transmission of false data, authentication and/or authorization violations, and a holistic security protection is required. In particular, vulnerabilities originate from different layers, all of which should be taken into account when analyzing security requirements. For example, users may misuse the smart items and make it function incorrectly; the communication among smart items are in plain text; the smart item devices may be exposed to the public and be manipulated or destroyed.

In this case study, we focused on a particular healthcare scenario, in which patients are continuously monitored after hospitalization (e.g., after cardiac infarction), the system is responsible to react to abnormal situations, which are determined based on the monitored data. A patient must be kept constantly in contact with her doctor and hospital and transfers information about her cardiac activity to the medical emergence system. In addition, both the patient's personal information (e.g., age, sex, and family history) and his body condition information (e.g., asleep or awake, walking or sit down) are also collected and passed to the system. As such, there is a strong security need that the confidentiality of such information should be protected from disclosure.

Such a scenario involves a multitude of social actors (patients, doctors, social workers, etc.), software applications (e-health application, control station application, etc.), physical devices (PDAs, computers, smart items, etc.), existing in different layers. In addition, the scenario requires the integration of different medical services, such as remote diagnosis and first-aid services.

9.2.2 Modeling the Medical Emergency Response System

In this section, we report the modeling practices performed by the Master student. In particular, different from the previous case study, we decided to follow a bottom-up manner to construct the three-layer requirements goal model in the social layer, helping us to grasp the overall understanding of the scenario.

Modeling business process. Fig. 9.6 shows the business process that was built for the medical emergency scenario. In particular, this process starts from the event that the patient feels dizzy and request for assistance. Once MERC (Medical Emergency Response Center) received the request from the patient, she needs to check the availability of designated doctors and social workers. The doctor then asks patients for related medical data through his e-health terminal and analyzes them to determine appropriate treatments. After that the doctor writes an e-prescription and uploads it to the system, which can be accessed by the patient.

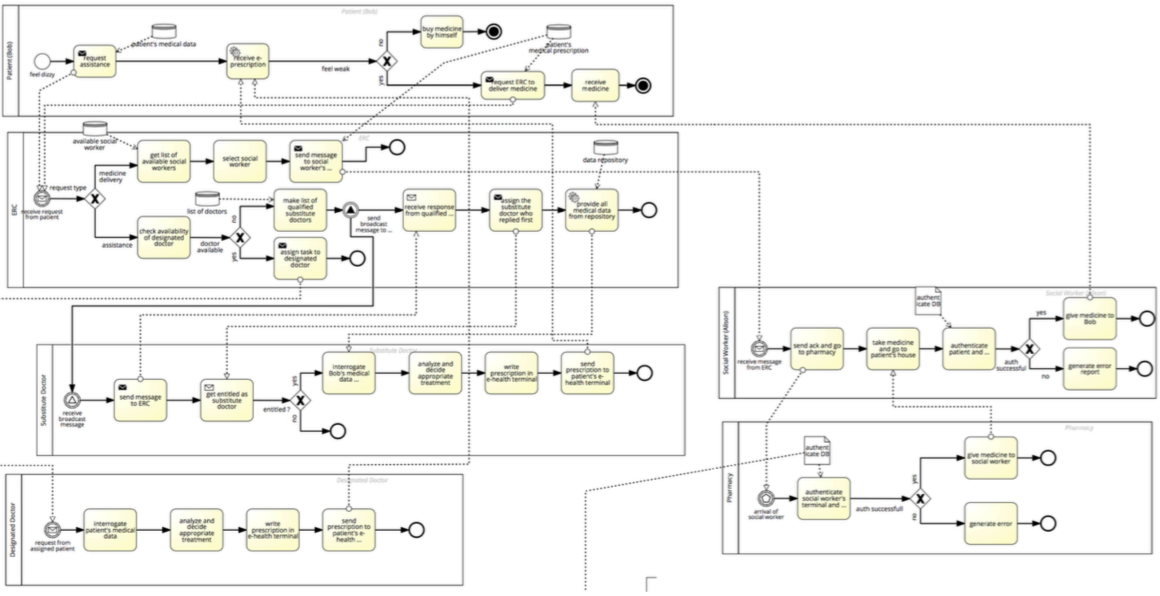

Patient (Bob)
feel dizzy
request assistance
patient's medical data
receive e-prescription
feel weak
no
yes
buy medicine by himself
patient's medical prescription
request ERC to deliver medicine
receive medicine
ERC
available social worker
get list of available social workers
select social worker
send message to social worker's ...
request type
receive request from patient
medicine delivery
assistance
check availability of designated doctor
doctor available
list of doctors
make list of qualified substitute doctors
assign task to designated doctor
send broadcast message to ...
receive response from qualified ...
assign the substitute doctor who replied first
data repository
provide all medical data from repository
Substitute Doctor
receive broadcast message
send message to ERC
get entitled as substitute doctor
entitled ?
interrogate Bob's medical data ...
analyze and decide appropriate treatment
write prescription in e-health terminal
send prescription to patient's e-health terminal
Designated Doctor
request from assigned patient
interrogate patient's medical data
analyze and decide appropriate treatment
write prescription in e-health terminal
send prescription to patient's e-health ...
Social Worker (Alison)
receive message from ERC
send ack and go to pharmacy
take medicine and go to patient's house
authenticate DB
authenticate patient and ...
auth successful
give medicine to Bob
generate error report
Pharmacy
arrival of social worker
authenticate social worker's terminal and ...
auth successfull
give medicine to social worker
generate error

Figure 9.6: The business process model of the medical emergency scenario

In case that the designated doctor is not available, MERC obtains a list of qualified doctors from the database and check their availability by sending them a message. Then, MERC selects the doctor who first replies the message and assigns him the emergency task. All the medical data of the patient is passed to that doctor in the meantime. Once receiving all the patient data, the doctor should determine appropriate treatments and issue an e-prescription. In case the doctor needs additional information, she can interrogate the patient through her e-health terminal.

After the patient receives the e-prescription via her e-health terminal, she can either go to pharmacy in person to buy the prescribed medicines or ask MERC to deliver such medicines to her home. In the later case, MERC should check the availability of social workers. After selecting the most suitable social worker, MERC needs to notify that social worker and authorizes her to access the patient's e-prescription. Once a social worker received the task assignment from MERC, she then goes to the nearest pharmacy to buy all the required medicines. In particular, when the social worker reaches pharmacy, the pharmacy worker needs to authenticate the social worker through her personal terminal. If and only if the authentication is successful, the social worker can get the prescribed medicines. Otherwise an error report will be generated and sent to MERC. When delivering the medicines to the patients, the patient and the social worker should authenticate each other via their personal terminals.

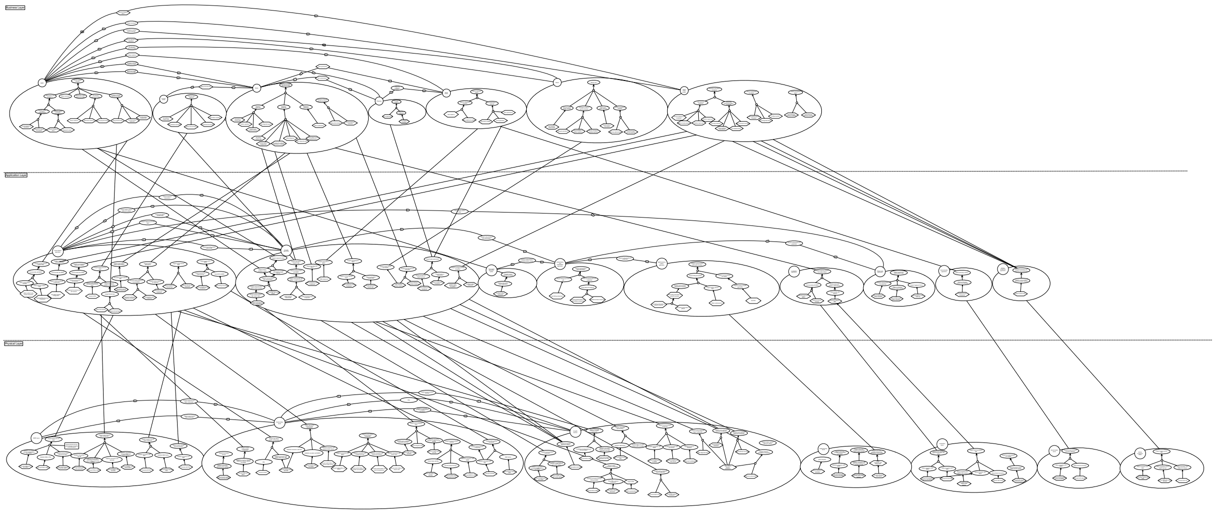


Figure 9.7: The three-layer requirements goal model for the medical emergency scenario

Build the three-layer requirements goal model. Based on the above business process model, the student then built the requirements goal model in the social layer. In particular, each business process activity was mapped into a business task, from which he established the goal model following a bottom-up manner, i.e., asking "why a business task is required". Once he built the goal model in the social layer, he then analyzed the cross-layer links between the social layer and the to-be-created application layer, based on which the requirements goal model in the application layer was established. By further analyzing the cross-layer links between the application layer and the infrastructure layer, he then finished the requirements goal model in the infrastructure layer. Eventually, a full model was obtained, as shown in Fig. 9.7[4].

Specifically, Table 9.2 shows the detailed statistics of the entire model, which is even larger than the full model of the previous case study. We here briefly describe the strategic dependencies and cross-layer support relations within this model. In the business layer, *MERC*'s primary goal *Provide medical services* depends upon all other actors in this layer. For example, *MERC* has a business goal *Specialized health service provided*, the satisfaction of which depends on *Doctor* who further depends on *First Aid Worker* to

---

[4] The vector model file can be found at, `http://disi.unitn.it/~li/thesis/validation_model_mers.pdf`

Table 9.2: Statistics of the three-layer requirements model (medical emergency)

Layer	Actor	Goal	Task	(and)Refine	Operationalize	Dependency
All	24	182	166	117	157	20
Social	7	29	56	19	55	12
Software	9	64	46	43	47	6
Infrastructure	8	89	64	55	55	2

execute some tasks. Other social actors also depend on *MERC* for certain goals, such as *Patient* depends on *MERC* for medical services.

In the application layer, *e-Health Application* is used by many social actors, including *Doctor*, *Social Worker*, *Pharmacy* and *Patient*. As such, the corresponding business tasks of such actors are supported by *e-Health Application*. In the meantime, *e-Health Application* further depends on *Control Station Application*, which is mainly used by MERC, for fulfilling a couple of goals.

In the physical layer, *MERC Server* and *Control Station Server* depends on each other, functioning together to support *Control Station Application*. In addition, *e-Health Terminal* depends on *Control Station Server* for several goals so as to support the functions of *e-Health Application*.

Evaluation and discussion. As reported by the Master student, the overall modeling practice is time-consuming. In particular, he spent around a month in collecting data for the case study and comprehending the entire scenario. There are several reasons for such a long time: firstly, this medical emergency system is a large STS and the analyzed scenario has more complex business logic than the smart grid scenario we described in the first case study. Secondly, similar with the previous case study, we did not have the opportunity to directly interview stakeholders and collect data from them. Instead, we can only indirectly search for existing reports or documents from different sources to complete the entire picture of the system. Especially, the Master student has little background knowledge about this scenario before. Overall, this result further emphasizes the need of involving different experts in the modeling practice, from whom the analyst can directly and easily obtain required information.

Once all the information has been collected, the Master student spent another month in building the three-layer model: "*it took me approximately 2 months to complete the three-layer holistic model for our case with numerous revisions*". Compared to the first case study, the modeling time has sharply increased. For one thing, we realized that the difficulty of modeling can grow very fast when the scenario became complicated. For another thing, as the student was unfamiliar with the modeling tool (i.e., OmniGraffle), we found out afterwards that he did not exert the full power of the tool, which we believe affected his modeling performance. For example, the tool offers the feature to *select all similar objects (i.e., goals)*, without using this feature the student has to manually operate on each individual object one-by-one. As a result, in the future study, we should spend more time in teaching the powerful features of our modeling tool.

Although the modeling is time-consuming, as acknowledged by the student:"*The process became a lot easier because of the framework and its three layer approach, as the multilayer security requirement analysis framework was easy to understand and as I had to model in different layers the process became intuitive after the first layer*". As such, we argue that the complexity of the model is inherited from the scenario itself, and our framework can contribute to the modularity of such complex systems.

Lastly, within this modeling practice, we have discovered that building the requirements model in a bottom-up manner is an efficient strategy. This is because that we have assigned layer-specific operational definitions to tasks in each layer. In particular, for this study, the student first built a business process model for this scenario, the benefits of which are twofold: firstly, by modeling the business process, the student was able to have a better understanding of the scenario, facilitating his subsequent modeling tasks. Secondly, the activities of the business process model can be mapped to tasks in social layer, directly facilitating the modeling in the social layer. Especially, as the student mentioned that "*the modeling in social layer is most difficult part of the three-layer modeling*", we should pay more attention to support the social layer modeling. As such, we propose to follow such bottom-up process to construct requirements goal model in the social layer.

9.2.3 Security Requirements Analysis Results

Build reference models. In order to perform security requirements analysis for the medical emergency system, the student was asked to collect additional information and to build corresponding models, which were used in refining and simplifying security patterns (as introduced in Section 4.3). In particular, the student generated a resource model, shown in Fig. 9.8. Each resource presented in this figure is involved in a particular part of the scenario. Moreover, the student produced a data flow model, which was described in terms of the input and output of requirements tasks. Thanks to the business process model built before, this data flow information can be easily obtained. As our analysis needs to know potential threats to the system, the student was asked to collect the threat information, and eight possible threats were eventually identified from literature. In particular, each of the eight threats was classified and specified in terms of *threat type*, *threatened asset*, and *threatened interval*, as introduced in Section 4.3.2. It is worth noting that we here did not require the student to adopt our holistic threat analysis approach (Chapter 5) to identify system threats, because the focus of this case study is evaluating the practical adoption of the holistic security requirements analysis approach. Also, the full application of our attack analysis approach (including the learning period) may require another considerable amount of time, which cannot be afforded in this case study. As future work, we definitely need to further evaluate the attack analysis approach, which we will discuss in Chapter 10.

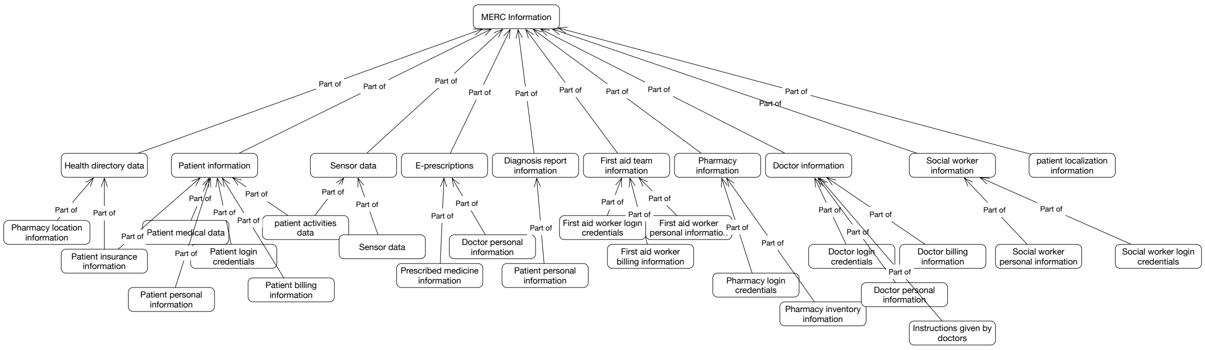

MERC Information
Part of
Health directory data
Patient information
Sensor data
E-prescriptions
Diagnosis report information
First aid team information
Pharmacy information
Doctor information
Social worker information
patient localization information
Pharmacy location information
Patient insurance information
Patient medical data
Patient login credentials
Patient personal information
Patient billing information
patient activities data
Sensor data
Prescribed medicine information
Doctor personal information
Patient personal information
First aid worker login credentials
First aid worker personal informatio...
First aid worker billing information
Pharmacy login credentials
Pharmacy inventory infomation
Doctor login credentials
Doctor billing information
Doctor personal information
Instructions given by doctors
Social worker personal information
Social worker login credentials

Figure 9.8: The resource model related to the medical emergency system

Holistic security requirements analysis. Once all reference models have been built, the student performed holistic security requirements analysis, following the analysis process described in Section 4.3. In particular, the student iteratively refined and concretized security goals with the aim of identifying security mechanisms to operationalize such security goals. When using our prototype tool to semi-automate the analysis, the student strictly followed the use cases we have specified in Section 8.3. During the analysis process, the student was asked to perform the analysis by himself. In case he had questions or doubts about the analysis process, he turned to me for clarification and help.

Overall, the student spent one week going through the entire analysis process and established the complete holistic security goal model as shown in Fig. 9.9 (the corresponding model file can be found online[5]). This model was much more complicated than the one we obtained in the first case study. Especially, by performing the holistic security solution analysis, it ended up with more than 73000 alternatives. For example, Table 9.3 shows two exemplary holistic security solution alternatives, both of which provided security solutions in all three layer. Note that these two alternatives had the same solutions in the social and software layers and only differed in the infrastructure layer. This was a common case among all the alternatives, i.e., different alternatives may share most of their solutions

---

[5] `http://disi.unitn.it/~li/thesis/validation_case2_hsgm.pdf`

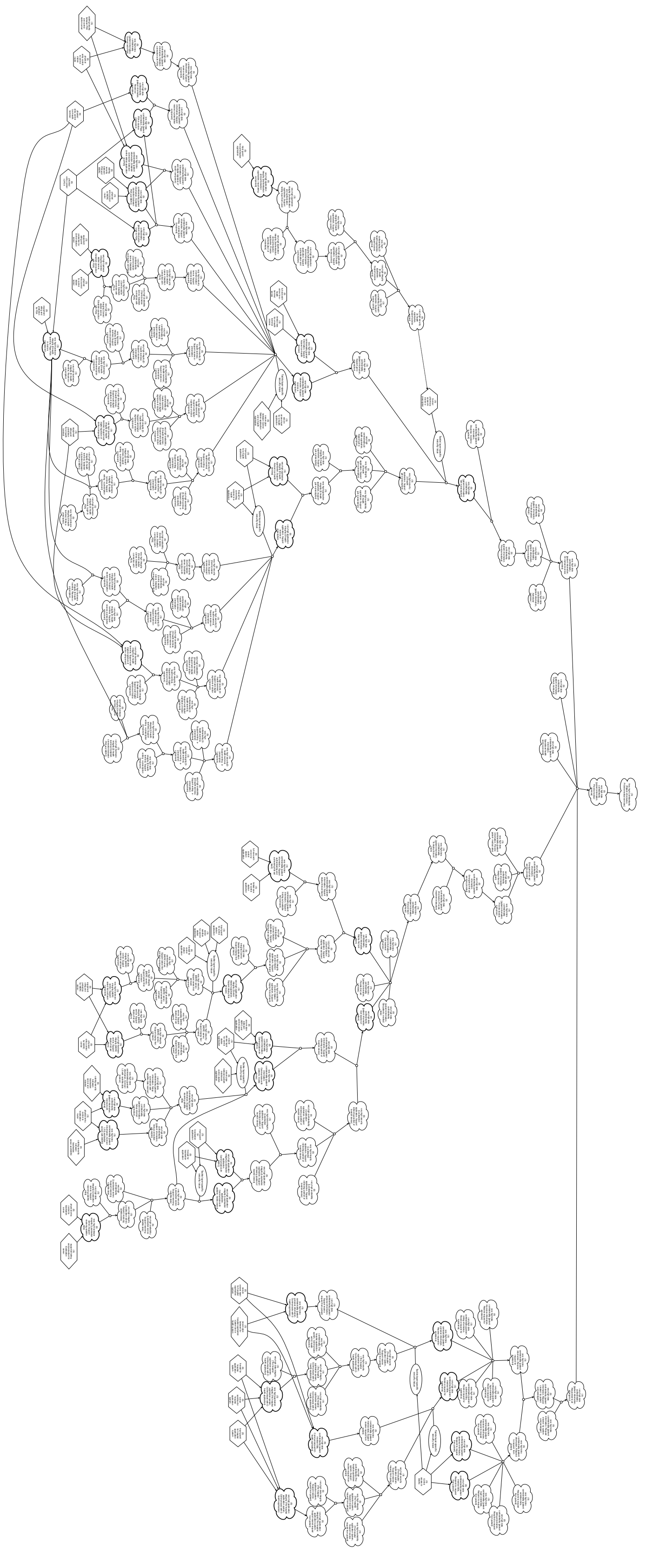


Figure 9.9: The entire holistic security goal model for the medical emergency system

and only differ in some particular parts. Given such a huge amount of alternatives, there was a strong need of selecting the best solution, which will be implemented in our future work.

Evaluation. Overall, the Master student was able to carry out the entire approach with the tool support, including both the modeling of the three-layer requirements goal model and the holistic security analysis over that model. However, the time spent for performing the approach was much longer than the first case study performed by myself, especially in the modeling session. There are several reasons account for this result: firstly, this case study is concerned with a large-scale STS, which is more complicated than the first case study. Secondly, the student was not familiar with the modeling tool, while the proficiency for using the modeling tool does matter the modeling performance. Especially, such a problem was further exacerbated by the complexity of the case study. As a result, in the future study, we should spend more time teaching analysts to exert the full power of the modeling tool. Thirdly, similar to the first case study, the Master student collected domain information of all three layers from different sources and then synthesized them all together, which was a non-trivial task. As we have discussed about the framework in Section 4.4, the ideal case for applying our approach is interacting with stakeholders in specific layers (e.g., business analysts, software architects, etc.) and directly extracting necessary domain information from them.

Although the application of our approach was time-consuming in this study, we argue this is due to the inherent complexity of the large-scale system, which cannot be avoided altogether. As reported by the student, on one hand, the student acknowledged the complexity of performing holistic security requirements analysis on such a complicated STS; on the other hand, the student did appreciate the (semi-)automation of analysis supported by our tool, without which the corresponding analysis task is impossible to complete. Therefore, we conclude that our approach did help the student to deal with such complexity.

Discussion. During the analysis process, we observed several issues, based on which we further discuss our approach as below.

Firstly, during the security goal refinement analysis in the social layer, the student tried to use the exhaustive refinement feature offered by the tool. Due to the high complexity of this scenario, the exhaustive refinement results in more than 800 security goals. After performing the simplification analysis over such security goals, 9 critical security goals were identified and another 58 security goals were identified as applicable (but non-critical). Then, the student went through all the applicable security goals to determine

Table 9.3: Two exemplary holistic security solutions

Layer	Alternative 1	Alternative 2
Social	Control pharmacy access to doctor e-prescription; Control access to MERC	Control pharmacy access to doctor e-prescription; Control access to MERC
Software	Input guard for authorization application; Limited view for e-health terminal access; Secure access layer for e-health application; Firewall for controlling access to e-health application; Server sandbox for control station application; Firewall for localization application; Limited view for control station application; Secure access layer for control station application; Server sandbox for hospital application system; Secure access layer for hospital application system	Input guard for authorization application; Limited view for e-health terminal access; Secure access layer for e-health application; Firewall for controlling access to e-health application; Server sandbox for control station application; Firewall for localization application; Limited view for control station application; Secure access layer for control station application; Server sandbox for hospital application system; Secure access layer for hospital application system
Infrastructure	Physical entry control to PDA; Physical entry control to e-health terminal; Cabling security between PDA and MERC; Cabling security between PDA and control station server; Physical entry control to control station server; Physical entry control for MERC server; Physical entry control to localization server; Physical entry control to localization router	Physical entry control to PDA; Physical entry control to e-health terminal; Cabling security between PDA and MERC; Cabling security between PDA and control station server physical entry control to control station server; Physical entry control for MERC server; Physical entry control to localization server; Cabling security between localization server and MERC; Cabling security between localization server and internet
Number of mechanisms	20	21

their criticality, which was a time-consuming task. Such scale of analysis complexity was due to the complicated domain model (Fig. 9.7) and resource model (Fig. 9.8). As we have proposed in Section 4.3.1, it is advisable to adopt a hybrid refinement strategy to deal with such complexity, which can avoid analyzing security goals that were actually not desired by stakeholders.

Secondly, during the security goal operationalization analysis, we noticed that some detailed security goals, which were concerned with the same security property and asset but differed in their concerned interval, may share the same security mechanisms. Therefore, once such security goals were operationalized into corresponding security mechanisms, the obtained security mechanism *instances* should be merged as one single instance. In other words, such phenomena require a manual check after the operationalization analysis.

Lastly, the security analysis results of this case study turned out to be over 73000 alternative security solutions. Intuitively, such a huge amount of alternatives is due to the complicated scenario. In order to have a better understanding of the scalability of our approach, We delveed into the holistic security goal model (Fig. 9.9) to further examine where the alternatives come from, . In particular, by looking at the syntax of the holistic security goal model, we identified that some security goals were *and-refined* into multiple sub-goals (up to eight), which was the main cause of the complexity. For example, if each of the eight sub-goal is operationalized into two security mechanisms, only this branch of model can have 2^8 alternatives, which can exponentially grow when other branches of models are taken into account. By having a close look at the semantics of the holistic security goal model, the above complex *and-refinement* structure is reasonable and does need to be captured and analyzed. For example, one business activity involved multiple stakeholders and software applications, among which confidential information was transferred. As such, it was reasonable to identify all such related actors and to impose corresponding security requirements on each of them (one security goal per actor), which then lead to the complex *and-refinement* structure.

9.3 Chapter Summary

In this chapter, we present two case studies that we have performed to validate our holistic security requirements framework. The first case study was performed by myself, focusing on evaluating the expressiveness of the three-layer requirements modeling language and the efficacy of the holistic analysis framework, which is reported in Section 9.1. The results of the study show that our approach is able to be applied to large-scale STSs and to generate holistic security solutions to satisfy system security requirements. Based on the threats to validity we have discussed for the first case study (Section 9.1.4), we have carried out the second case study, which was performed by a Master student under the supervision of Prof. Mylopoulos and myself. This case study is presented in Section 9.2, the results of which shows our approach can be adopt by people who were not involved in the development of our approach and has limited security knowledge.

Chapter 10

Conclusions and Future Work

This chapter summarizes the entire thesis and sheds light on our future work. In Section 10.1 we conclude the contributions of each part of this thesis. In Section 10.2 we discuss the limitations of our proposal and envision future research topics we intend to investigate.

10.1 Conclusions

The main objective of this thesis is to develop a comprehensive framework that assists security analysts in analyzing security requirements and generating security solutions for STSs in a holistic manner. To this respect, instead of dealing with security in a piecemeal fashion, we proposed a three-layer security requirements analysis framework, which takes into account the security concerns in different parts of STSs. Also, this framework can deal with the inherent complexity of STSs by dividing them into three conceptual layers, and thus tackles the original problem in a divide-and-conquer manner. In particular, we consider a social layer, a software layer, and an infrastructure layer. Each of these layers accounts for particular artifacts that need to be designed in STSs. Based on such division, we concern specific security requirements and solutions in each layer while take into account the connections among layers. In such a way, we eventually generate holistic security solutions which provide comprehensive protection for STSs. The proposed framework consists of a three-layer requirements modeling language and a systematic analysis process for holistically analyzing security requirements across three layers. Specifically, different analysis methods have been formalized in Disjunctive Datalog, based on which we have implemented semi-automated support for the entire analysis process.

An important input required by the above holistic security requirements analysis framework is a collection of threats to the system, which are used to determine the criticality of security requirements. Because of the complexity of STSs, it is challenging to identify all potential attacks, especially multistage attacks which assemble individual attacks from different parts of STSs. The identification of multistage attack within a socio-technical setting has not not been addressed by existing approaches, imposing a new research challenge. Therefore, we have developed a holistic security attack analysis approach as part of our holistic security requirements framework in order to deal with this particular challenge. This approach takes an attacker's perspective to analyze attack strategies, and then operationalizes the strategies into specific attack actions based on a collection of attack patterns. In particular, we not only advocate taking the attacker's perspective but pragmatically analyze attacker's intentions via realistic attack scenarios. Moreover, driven by the same pragmatic thought, we leverage practical attack knowledge from existing attack patterns to operationalize attacker's anti-goal in terms of concrete attack behaviors. Specifically, we have modeled 102 existing attack patterns as contextual goal models, enabling semi-automatic selection among such attack patterns.

One important analysis step of our holistic security requirements analysis is to operationalizing security requirements in terms of security mechanisms. However, such analysis is a laborious and knowledge-intensive process, especially for large-scale STSs. This challenge has been further exacerbated by the fact that security knowledge is normally hard to acquire for system analysts. As a response to such a challenge, we have developed a systematic approach to efficiently leverage security patterns to accomplish the operationalization analysis, constituting another important part of the holistic security requirements framework. Similar to the attack pattern analysis, we have proposed to model textual security patterns as contextual goal models in order for semi-automatic analysis. In particular, we have delved into the practical details about the modeling of security patterns, and have provided systematic instructions about how to establish contextual goal models from textual security patterns. In addition, we have pragmatically followed such instructions to model 20 reusable security patterns.

During the operationalization analysis, we have realized that two typical operationalization methods (i.e., function operationalization and quality operationalization) are not enough to support the operationalization of security requirements. Specifically, operationalizing security goals into security mechanisms not only introduces new functions to the system specification, but also influences existing system functions. Ignoring such impact will result in incomplete or even faulty requirements specifications. As such, we have proposed a conceptual model for security mechanisms, based on which we have defined a systematic analysis process to capture and enforce the impact of security mechanisms imposed on existing system functions.

We have developed a prototype tool to provide semi-automated support for the holistic security requirements analysis, facilitating the practical adoption of our framework. The tool was initially designed to support the three-layer security requirements modeling and analysis, and has been incrementally enhanced to support the subsequent holistic attack analysis and security pattern analysis. In particular, the tool helps analysts to deal with the scalability issues in two ways: firstly, the tool is built on a professional diagramming application and can well separate the modeling tasks in terms of layers, perfectly implementing the modularity feature provided by our approach. In addition, the prototype tool inherits many useful modeling features from the diagramming application, relieving the scalability problem, especially when modeling large-scale STSs. Apart from the modeling support, our prototype tool also provides useful analysis support, i.e., (semi-)automating different analysis methods proposed in this thesis. Our tool performs automatic analysis over target models, and then reflects the analysis results in the graphical canvas. In particular, the prototype tool can automatically generate part of the analytic models, relieving analysts from manually creating the models.

Apart from the above proposals, we have also performed two case studies with the aim of collecting empirical evidence about the efficacy of our approach. Specifically, the first case study was performed by myself on a real smart grid scenario, in which I adopted the entire proposal in this thesis to holistically analyze security requirements in order for protecting the metering data. The results of the study show that our approach can identify holistic security solutions across three layers, and it is scalable to large-scale STSs. In addition, we have also observed several issues that help us to improve our approach. For the second case study, we applied our approach to another large-scale STS (i.e., medical emergency response system). In particular, we adapted our study design from the first study and had a Master student to apply the holistic analysis approach, who was not involved in the development of our approach and has limited security knowledge. The student was first taught the three-layer security requirements analysis framework, and then used the prototype tool to perform the holistic security requirements analysis under the supervision of Prof. Mylopoulos and myself. According to the evidence we collected from this case study, we contend that our approach can be applied by people other than the method developers.

10.2 Discussion

In this section, we first discuss limitations of the approaches we have proposed in different parts of the thesis. Next, inspired by our current proposal, we envision a number of future research directions,

10.2.1 Limitations

We discuss the limitations of our current proposal from three different aspects: security knowledge, empirical evaluations, and tool supporting.

Security Knowledge. Since our approach relies on external security/attack knowledge to perform corresponding analysis, the analysis results are affected by such external knowledge sources. As identified by our case study, the current selection of security patterns is not enough to cover all threats we have analyzed in the case study; and of course, there can be further threats that turn out to be not covered by selected security pattern. Therefore, we need to continuously incorporate additional security patterns into our approach, reacting to emerging threats. Apart from the coverage of the external knowledge sources, the quality of the external knowledge sources can also affect our analysis results. When practically processing existing security patterns and attack patterns, we have already identified that some of the patterns are incomplete, failing to accommodate all the required knowledge. To deal with this limitation, we should keep updating our processed security/attack patterns with regard to the recent advances in the corresponding research fields.

On the other hand, we argue that this limitation is not specific to our approach, but is generally applied to all the security analysis approaches that propose to reuse existing knowledge. As reported in [Souag et al., 2015], the research about knowledge reuse is still immature, especially lacks automation support. Different from most existing approaches that only mention the idea of knowledge reuse, our approach has delved into the practical adoption of knowledge reuse, and thus has encountered the above limitations.

Empirical Evaluations. Although we have performed case studies to evaluate each part of the thesis, we have acknowledged the needs of collecting further empirical evidence in regard to the follow aspects: firstly, the case study we have performed with a Master student provides preliminary evidence that our approach can be adopted in reality by people that were not involved in the design of the approach. However, we need to further evaluate our approach with a significant number of participants. To this end, controlled experiments would be the ideal empirical method to collect such empirical evidence. For this thesis, the reason of choosing case study as the empirical evaluation method (instead of controlled experiment) is that the full application of our approach is difficult to control in terms of time. This argument has also been further verified by our second case study, in which the student took around three months to completely apply the entire approach. The main reason of this challenge is because our approach is intended to solve a very complex and difficult problem, i.e., holistically analyzing security requirements for large-scale STSs. To address this challenge, we have planned to first divide the entire approach into independent parts, and then perform controlled experiments for each of these parts. For example, we can separate the modeling session from the analysis session. In addition, a more practical solution is to combine the evaluation of the approach with a security course, in which students are required to step-by-step adopt the approach during the entire semester.

Secondly, we need more empirical evidence to account for the quality of our analysis results. To this end, we plan to have experienced security experts to revise and validate the holistic security solutions produced by our approach. In addition, by involving security experts in the empirical evaluation of our approach, we also want them to provide comments and suggestions from a practical perspective, helping us to improve our approach to tackle more practical problems. In such a way, we want to promote the practical adoption of our approach.

Thirdly, we have encountered problems of information collection in both of the two case studies (to different extents). We argue that by involving corresponding domain experts and directly interacting with them, we are able to collect the information required by our approach with much less effort. Consequently, we need to collect more empirical evidence to support such an argument. Ideally, we want to perform further case studies or action research over a realistic project, in which we are able to directly interact with corresponding domain experts.

Integration. In this thesis, we have first proposed the three-layer security requirements analysis framework as an essential contribution (Chapter 4), and then incrementally propose several approaches that support the application of the three-layer framework (Chapter 5-7). As specified in corresponding chapters, such supporting approaches are applied to a particular analysis step of the holistic security requirements analysis. In particular, for each of such supporting approaches, we have demonstrate and evaluate its utility, as presented in corresponding chapters. However, due to a number of theoretical and practical gaps, such supporting approaches cannot be fully integrated with the holistic security requirements analysis, indicating limitations of this thesis.

Specifically, the holistic security attack analysis (Chapter 5) identifies alternative attacks on systems, which are an important input required for identifying critical security goals. However, beyond such alternative attacks, analysts also need to know the risk level caused by such attacks. In other words, risk assessment should be performed based on the alternatives attacks, which is currently not included in our holistic security attack analysis. As a result, in order to fully integrate the holistic attack analysis into the holistic security requirements analysis, we need to further extend our current proposal with an additional risk assessment module.

For the security pattern analysis (Chapter 6), our proposal targets software security patterns Fernandez-Buglioni [2013], which has been specified in POSA templates. However, the organizational security patterns Asnar et al. [2011a] are documented in a different template, which does not specify the *Force* section. In order to apply our approach, we need to manually complement such information, which is a non-trivial task. Moreover, at the infrastructure layer, currently there are little security patterns have been proposed, and thus we have to turn to other knowledge sources (e.g., ISO27002) to tentatively create security patterns by ourselves, which are incomplete and require for verification. Such practical obstacles hinder the full application of our security pattern analysis in social and infrastructure layers, which should be addressed in the future.

Automated Tool Support. In this thesis, we have developed a prototype tool to semi-automate a number of proposed methods, supporting corresponding analysis. However, we have acknowledged that part of thesis is able to be (semi-)automated but has not yet. Specifically, for the attack strategy analysis, which was introduced in Section 5.2, we have learned and summarized a number of anti-goal refinement methods from realistic attack scenarios. Currently, we manually follow those methods to refine attacker's malicious intentions, but we believe this analysis can be automated to further simplify the holistic attack analysis. The challenge for automating such analysis is about the sanity check, which is performed after each step of refinement. This is because such check does require specific domain knowledge and is normally performed by people. Therefore, a semi-automatic support can be an appropriate solution, i.e., the tool first helps analysts to automate one step of anti-goal refinement and then asks for manual check afterwards.

Moreover, as we developed our prototype tool on top of a commercial diagramming tool, our tool cannot be widely distributed because of permission issues. As a result, for people who do not have the permission of the diagramming tool, our prototype tool is unusable. To solve this limitation, a possible solution is to base our prototype tool on other non-commercial diagramming tool. As introduced in Chapter 8, we have specified a collection of interfaces in our prototype tool for interacting with graphical models. Thus, as long as a non-commercial diagramming tool can provide similar interfaces, we are able to easily plant our prototype tool onto that diagramming tool. We argue that this is actually a trade-off between usability and availability. On one hand, the commercial diagramming tool offers very good usability which is important in large-scale modeling; On the other hand, the non-commercial tool can be widely distributed, although its usability might not be good. The reason why we choose to use the commercial tool is because in this thesis we intend to exclusively focus on the efficacy of our holistic analysis approach without bothering to deal with the distribution of the tool. Once our approach has been proven as efficient and useful, then we will start concerning the public availability of our prototype tool.

10.2.2 Ongoing Work and Future Research Directions

Inspired by the proposal in this thesis, we have identified some additional research directions that we intend to investigate. In the remainder of this section, we first introduce a piece of ongoing research about operationalizing requirements with aspectual mechanisms. Apart from this work, we introduce a number of future research lines.

Operationalizing Requirements with Aspectual Mechanism. Requirements operationalization amounts to transforming a problem into corresponding solution space, i.e., from stakeholder requirements to a specification of the system-to-be. Typically, there are two complementary ways of performing such operationalization as specified in [Dalpiaz et al., 2014], targeting functional requirements and quality requirements, respectively. Firstly, a functional requirement is operationalized into a function, which makes the requirement operational and is able to be implemented by the system-to-be. Secondly, a quality requirement is operationalized as a constraint on a metric that makes the quality requirement measurable at runtime. In particular, the functions and quality constraints that have been used to operationalize stakeholder requirements together constitute a specification, which are in the same spirit of IEEE standard for requirements specifications [Committee and Board, 1993].

However, these two ways of operationalization are not enough to represent the operationalization practices in the real world. In particular, we argue that such operationalization methods can be applied to deal with “standalone” requirements that function by themselves instead of functioning over other requirements. For example, “The system should measure energy load” can be operationalized into a measurement function, i.e., applying the functional operationalization. On the other hand, apart from the standalone requirements, there are requirements that interact with other requirements, as introduced by Robinson et al. [2003]. We argue that these two operationalization methods are not enough for operationalizing such “interactive” requirements. For example, considering a persistence requirement “a website should remember the states of user activities (e.g., adding items into the shopping cart)”. If we apply *HTTP cookies* as a solution to satisfy this requirement problem, we not only need to add functions such as *read/write cookies*, but should also modify existing functions that affect the states of user activities, i.e., constraining such existing functions in the specification to appropriately operate cookies once changed the user states.

We propose mechanisms as a complementary operationalization method to tackle the above challenge. A mechanism is a system of causally interacting parts and processes that produce one or more effects. In this respect, mechanisms are aspectual in that they require several elements of a specification to work together to fulfill a requirement, unlike their functional and quality cousins. In particular, we have observed many requirements operationalization instances that fall into the category of mechanisms, such as security, performance, and usability requirements, etc.

Currently, we are investigating a specification language for describing mechanisms. On one hand, we intend to specify a mechanism as a set of rewrite rules over specifications. As such, the requirements operationalization amounts to a specification rewriting problem. On the other hand, we want to preserve the Church-Rosser property during the specification rewriting.

Runtime Adaptive Security Design. Our proposal in this thesis focuses on designing secure STSs at the design time, which constitutes an important step in achieving system security. As contended by Bruce Schneier that "*Security is a process, not a product*", we want to move to the next step of the security process, i.e., designing adaptive security at runtime. In particular, we want to achieve two objectives: firstly, adjust security designs in a real-time manner to protect the system from different attacks. Secondly, as security is not free, the security mechanisms should be switched off whenever possible.

In order to achieve such research objectives, we want to adopt control theory, establishing a feed-back loop to constantly check the system environment and to react correspondingly. To this end, we need to first precisely capture the system context, based on which we can determine the security situation. We argue that our holistic attack analysis approach (Chapter 5) can contribute to this task. Specifically, as we analyze attacker's intentions via anti-goal models and associate them with concrete attack actions, we are able to detect the *attack progress* of multistage attacks, and thus take countermeasures in time. Next, for the countermeasures, we should be able to specify a collection of options of security mechanism reconfiguration, which will be applied at runtime to deal with particular security situation. For this part, we need to grasp more security knowledge regarding this task, and we intend to start from a particular application domain, such as cloud computing.

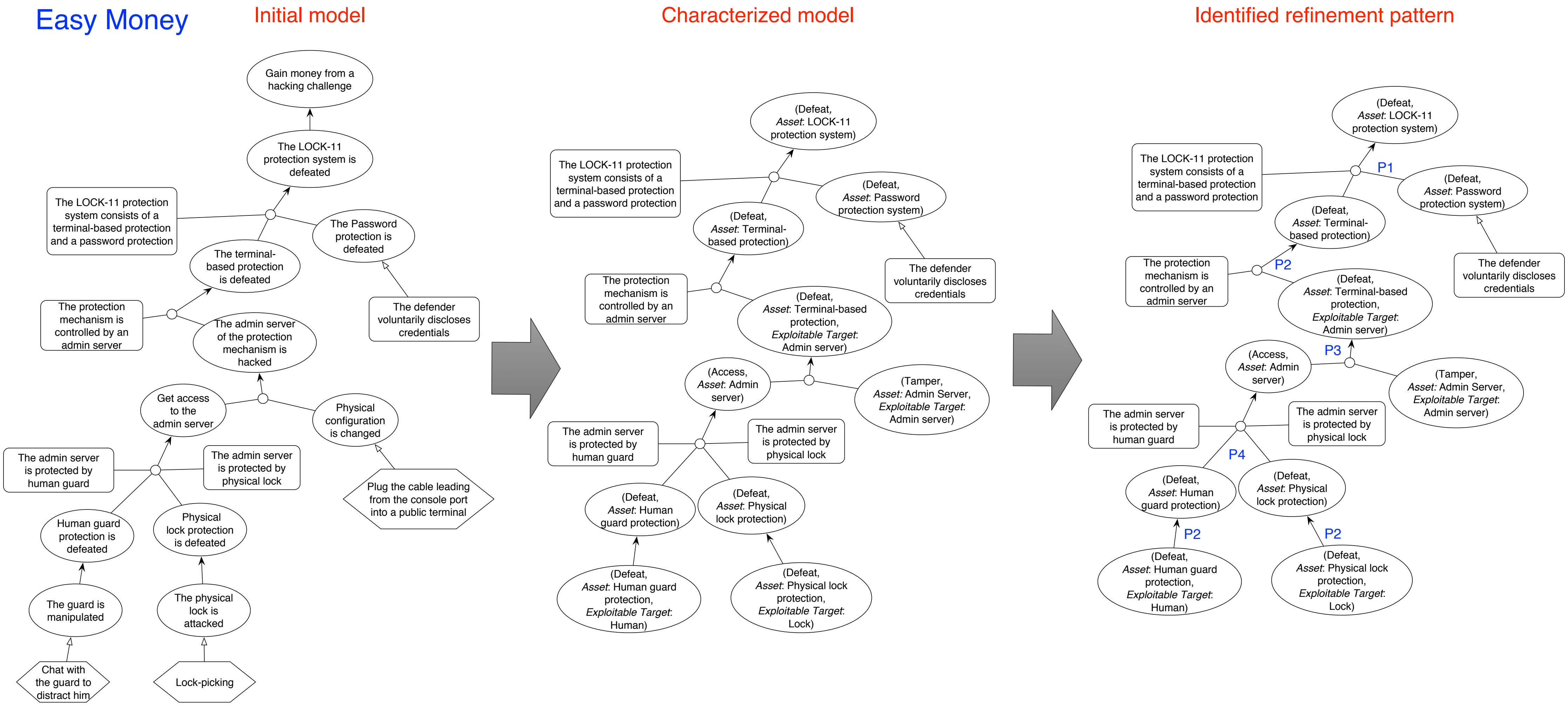
Flexible Multilayer Security Requirements Framework. Our three-layer requirements framework is an initial effort to deal with complicated (security) requirements problems via a multilayer manner, which can be further extended in the future. In particular, the number and content of layers can be flexible, depending on the scope and the complexity of the system. For instance, if the target system is a technical-oriented system and does not involve the design of the business process, then the framework can only focus on the application layer and the physical layer. For another example, if the target system not only deals with the software applications, but also intensively involves issues about operating systems, then it is better to add an additional operating system layer between the software layer and the infrastructure layer. Regardless of the structure of layers, we argue that the analysis of the multilayer framework stays the same.

As such, we intend to propose a flexible multilayer framework based on our current proposal in order to accommodate holistic security requirements analysis in different domains. Specifically, we want to define a systematic process for adding or removing a layer to the framework, by following which analysts are able to customize and obtain their particular multilayer framework that help them to deal with security requirements analysis in particular domains. Furthermore, although the current three-layer framework analyzes the influences between layers via a top-down manner, we acknowledge that the influences can also be propagated in bottom-up manner. For example, an infrastructure task can create software or social requirements. As such, we also aim to define appropriate mechanisms to allow analysts to perform influence analysis via a bottom-up manner. Note that after enabling the bottom-up analysis, the overall analysis can be much more complicated, as there can be an analysis loop among layers. Thus, a criterion for ending the analysis is required.

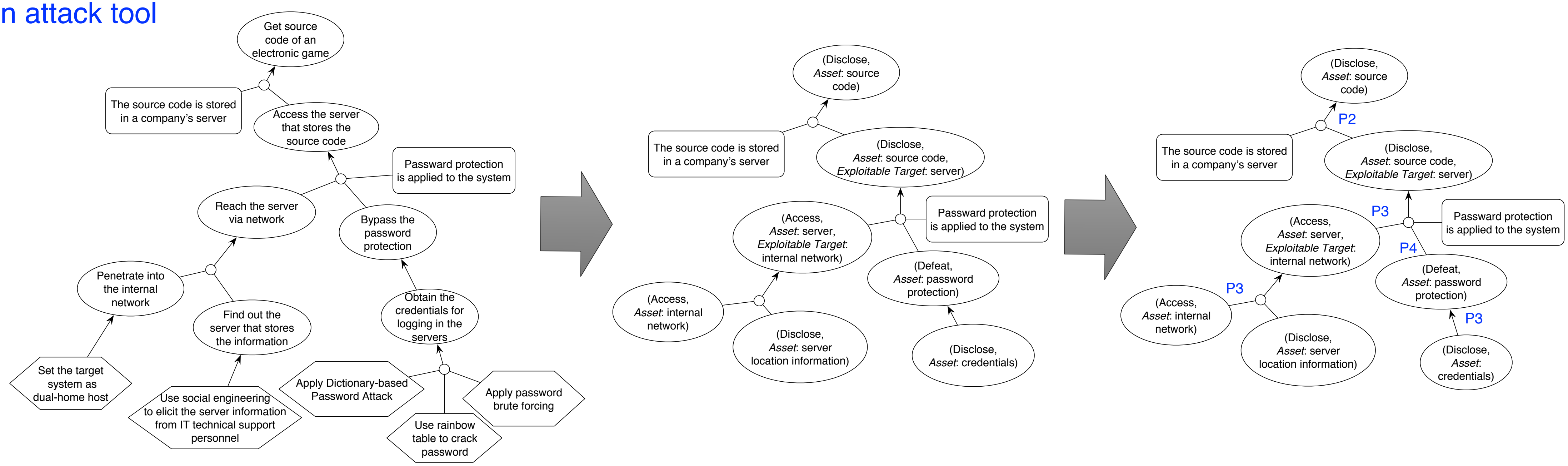
Agent-Oriented Attack Simulation. The essence of our holistic attack analysis approach (Chapter 5) is to pragmatically take an attacker's perspective and analyze potential attacks that can be performed by the attacker. In particular, the proposed approach is intended to reflect how an attacker refines her malicious intentions and operationalizes it into concrete attack actions. Currently, we manually follow the approach to analyze one single attacker. As the next step of research, we intend to automate such analysis by formalizing the proposed anti-goal refinement methods. Therefore, in line with the agent-oriented paradigm that has been applied in Software Engineering (SE), we can also treat each attacker as an intelligent agent and thus explore alternative attacks by simulating different attackers' behaviors. In particular, each attacker can automatically refine her malicious goals by using our proposed anti-goal refinement methods and then operationalize her malicious intentions into attack actions based on attack patterns. In this way, we can better explore the space of attack alternatives. Especially, we contend it is important to take into account the interactions among attackers. This means one attacker can also communicate/negotiate with other attackers and endeavor to deliver more complicated and damaging attacks via appropriate collaborations.

Appendix A

Attack Scenario Studies


Easy Money
Initial model
Gain money from a hacking challenge
The LOCK-11 protection system is defeated
The LOCK-11 protection system consists of a terminal-based protection and a password protection
The terminal-based protection is defeated
The Password protection is defeated
The defender voluntarily discloses credentials
The protection mechanism is controlled by an admin server
The admin server of the protection mechanism is hacked
Get access to the admin server
Physical configuration is changed
Plug the cable leading from the console port into a public terminal
The admin server is protected by human guard
The admin server is protected by physical lock
Human guard protection is defeated
Physical lock protection is defeated
The guard is manipulated
The physical lock is attacked
Chat with the guard to distract him
Lock-picking
Characterized model
(Defeat, Asset: LOCK-11 protection system)
The LOCK-11 protection system consists of a terminal-based protection and a password protection
(Defeat, Asset: Password protection system)
(Defeat, Asset: Terminal-based protection)
The defender voluntarily discloses credentials
The protection mechanism is controlled by an admin server
(Defeat, Asset: Terminal-based protection, Exploitable Target: Admin server)
(Access, Asset: Admin server)
(Tamper, Asset: Admin Server, Exploitable Target: Admin server)
The admin server is protected by human guard
The admin server is protected by physical lock
(Defeat, Asset: Human guard protection)
(Defeat, Asset: Physical lock protection)
(Defeat, Asset: Human guard protection, Exploitable Target: Human)
(Defeat, Asset: Physical lock protection, Exploitable Target: Lock)
Identified refinement pattern
(Defeat, Asset: LOCK-11 protection system)
The LOCK-11 protection system consists of a terminal-based protection and a password protection
P1
(Defeat, Asset: Password protection system)
(Defeat, Asset: Terminal-based protection)
The defender voluntarily discloses credentials
The protection mechanism is controlled by an admin server
P2
(Defeat, Asset: Terminal-based protection, Exploitable Target: Admin server)
P3
(Access, Asset: Admin server)
(Tamper, Asset: Admin Server, Exploitable Target: Admin server)
The admin server is protected by human guard
The admin server is protected by physical lock
P4
(Defeat, Asset: Human guard protection)
(Defeat, Asset: Physical lock protection)
P2
P2
(Defeat, Asset: Human guard protection, Exploitable Target: Human)
(Defeat, Asset: Physical lock protection, Exploitable Target: Lock)

Dictionary as an attack tool



Initial model

Get source code of an electronic game

The source code is stored in a company's server

Access the server that stores the source code

Passward protection is applied to the system

Reach the server via network

Bypass the password protection

Penetrate into the internal network

Find out the server that stores the information

Obtain the credentials for logging in the servers

Set the target system as dual-home host

Use social engineering to elicit the server information from IT technical support personnel

Apply Dictionary-based Password Attack

Use rainbow table to crack password

Apply password brute forcing

Characterized model

(Disclose, *Asset*: source code)

The source code is stored in a company's server

(Disclose, *Asset*: source code, *Exploitable Target*: server)

Passward protection is applied to the system

(Access, *Asset*: server, *Exploitable Target*: internal network)

(Defeat, *Asset*: password protection)

(Access, *Asset*: internal network)

(Disclose, *Asset*: server location information)

(Disclose, *Asset*: credentials)

Identified refinement pattern

(Disclose, *Asset*: source code)

P2

The source code is stored in a company's server

(Disclose, *Asset*: source code, *Exploitable Target*: server)

P3

Passward protection is applied to the system

(Access, *Asset*: server, *Exploitable Target*: internal network)

P4

(Defeat, *Asset*: password protection)

P3

(Access, *Asset*: internal network)

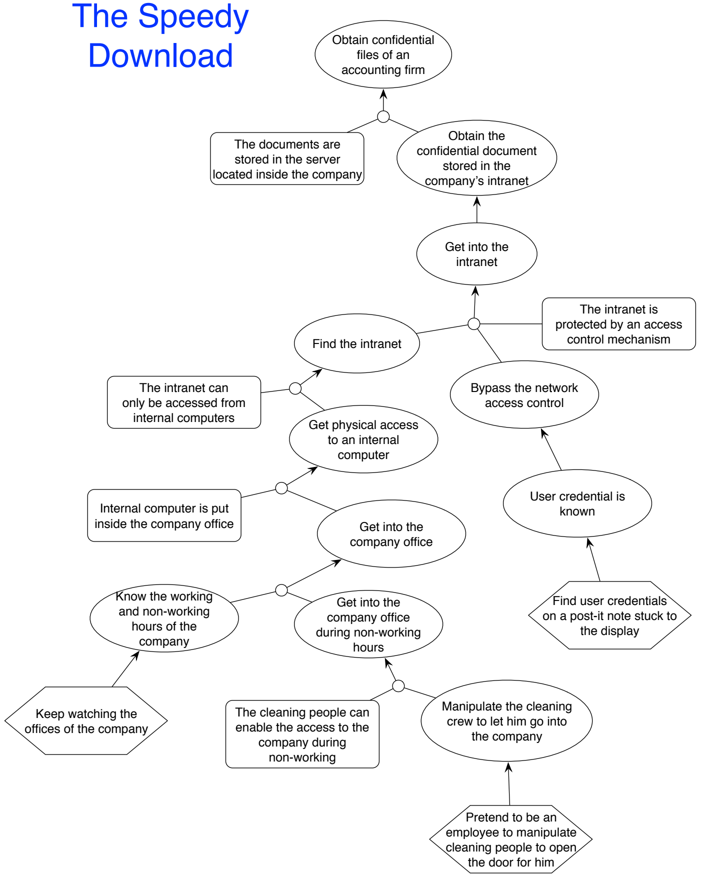
(Disclose, *Asset*: server location information)

P3

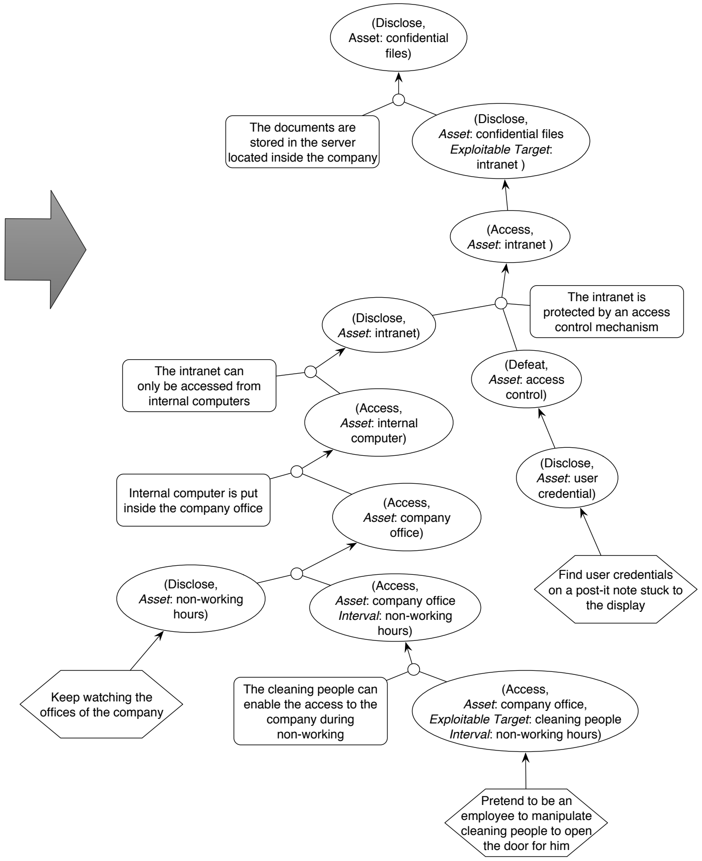
(Disclose, *Asset*: credentials)

The Speedy Download

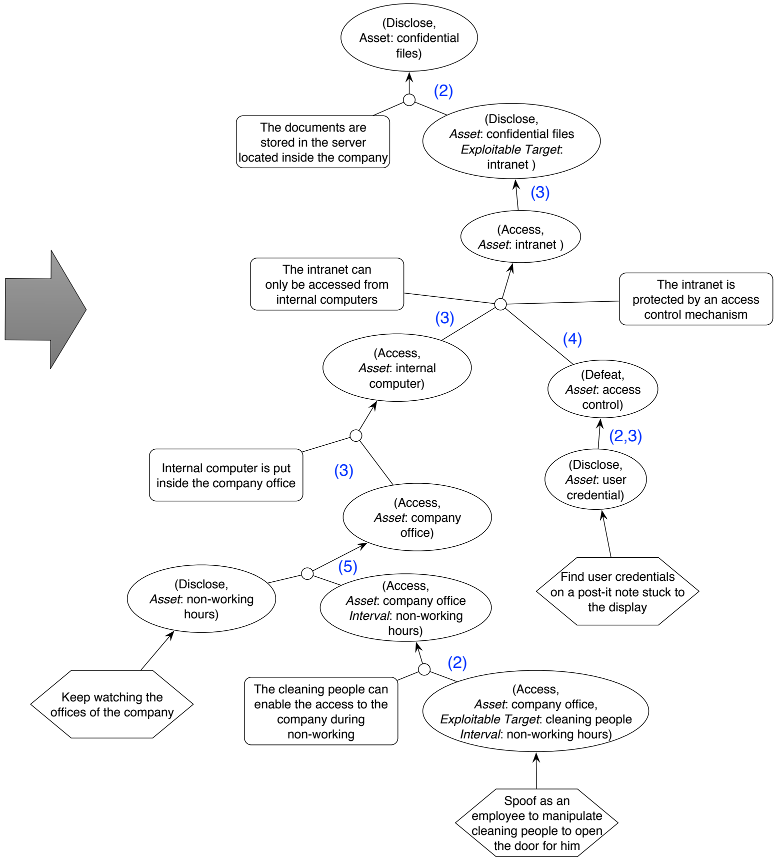
Initial model


Obtain confidential files of an accounting firm
The documents are stored in the server located inside the company
Obtain the confidential document stored in the company's intranet
Get into the intranet
Find the intranet
The intranet is protected by an access control mechanism
Bypass the network access control
The intranet can only be accessed from internal computers
Get physical access to an internal computer
Internal computer is put inside the company office
Get into the company office
User credential is known
Find user credentials on a post-it note stuck to the display
Know the working and non-working hours of the company
Get into the company office during non-working hours
Keep watching the offices of the company
The cleaning people can enable the access to the company during non-working
Manipulate the cleaning crew to let him go into the company
Pretend to be an employee to manipulate cleaning people to open the door for him

Characterized model

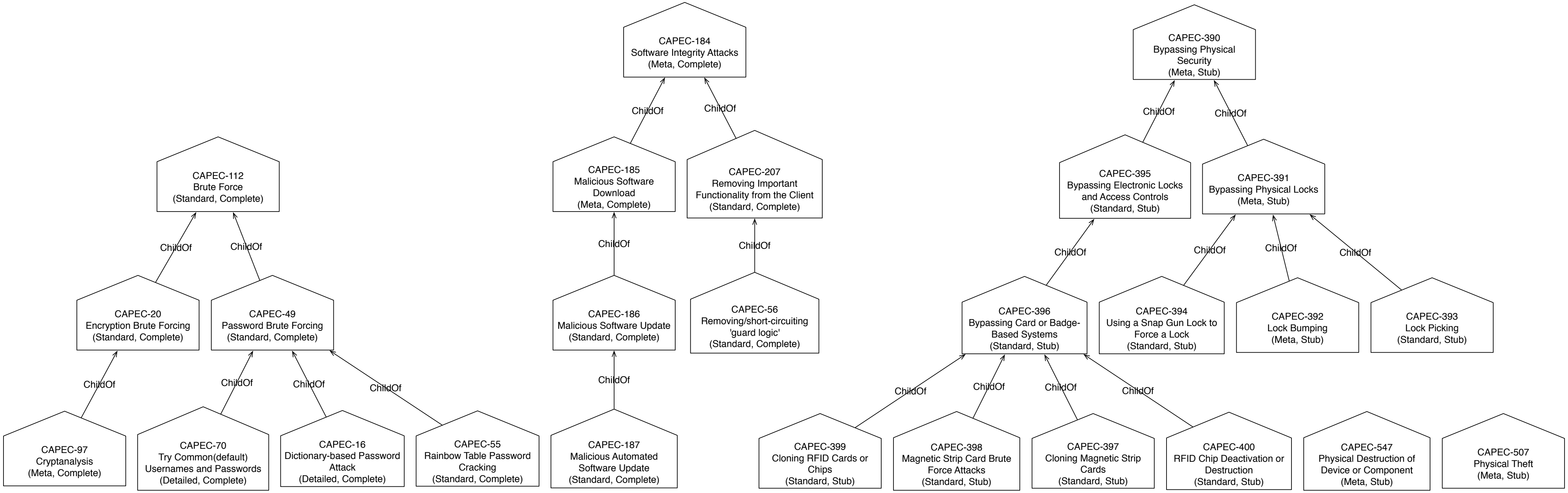

(Disclose, Asset: confidential files)
The documents are stored in the server located inside the company
(Disclose, *Asset*: confidential files *Exploitable Target*: intranet)
(Access, *Asset*: intranet)
(Disclose, *Asset*: intranet)
The intranet is protected by an access control mechanism
The intranet can only be accessed from internal computers
(Defeat, *Asset*: access control)
(Access, *Asset*: internal computer)
(Disclose, *Asset*: user credential)
Internal computer is put inside the company office
(Access, *Asset*: company office)
Find user credentials on a post-it note stuck to the display
(Disclose, *Asset*: non-working hours)
(Access, *Asset*: company office *Interval*: non-working hours)
Keep watching the offices of the company
The cleaning people can enable the access to the company during non-working
(Access, *Asset*: company office, *Exploitable Target*: cleaning people *Interval*: non-working hours)
Pretend to be an employee to manipulate cleaning people to open the door for him

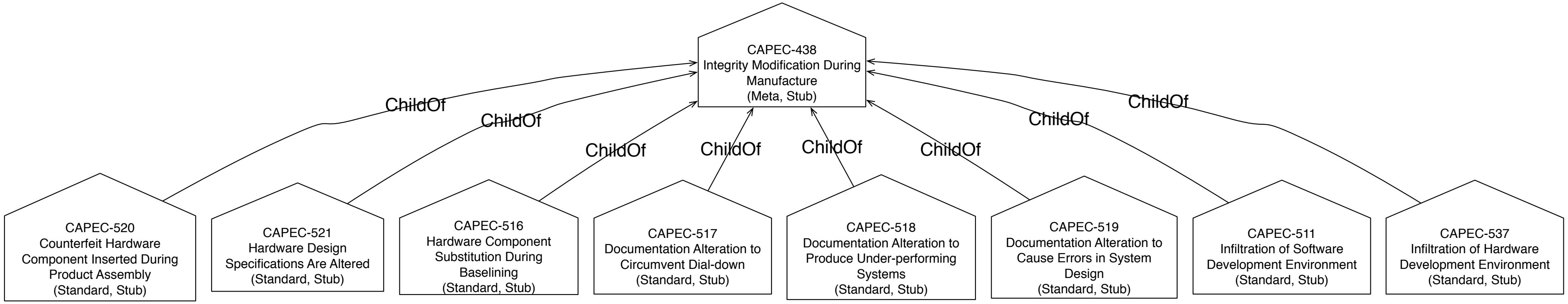
Identified refinement pattern

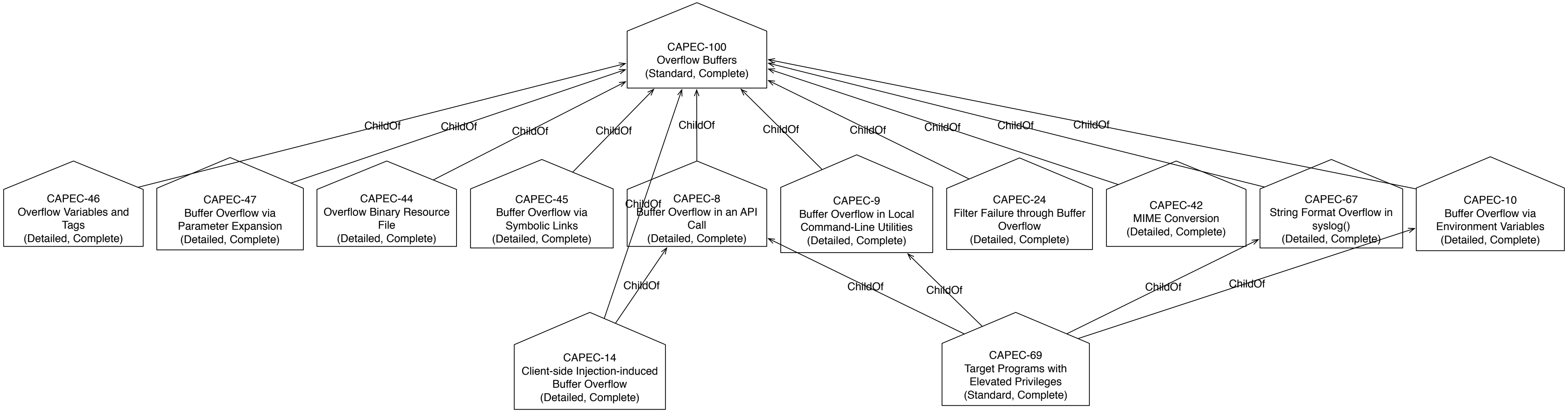

(Disclose, Asset: confidential files)
(2)
The documents are stored in the server located inside the company
(Disclose, *Asset*: confidential files *Exploitable Target*: intranet)
(3)
(Access, *Asset*: intranet)
The intranet can only be accessed from internal computers
The intranet is protected by an access control mechanism
(3)
(4)
(Access, *Asset*: internal computer)
(Defeat, *Asset*: access control)
(2,3)
Internal computer is put inside the company office
(3)
(Disclose, *Asset*: user credential)
(Access, *Asset*: company office)
(5)
(Disclose, *Asset*: non-working hours)
(Access, *Asset*: company office *Interval*: non-working hours)
Find user credentials on a post-it note stuck to the display
(2)
Keep watching the offices of the company
The cleaning people can enable the access to the company during non-working
(Access, *Asset*: company office, *Exploitable Target*: cleaning people *Interval*: non-working hours)
Spoof as an employee to manipulate cleaning people to open the door for him

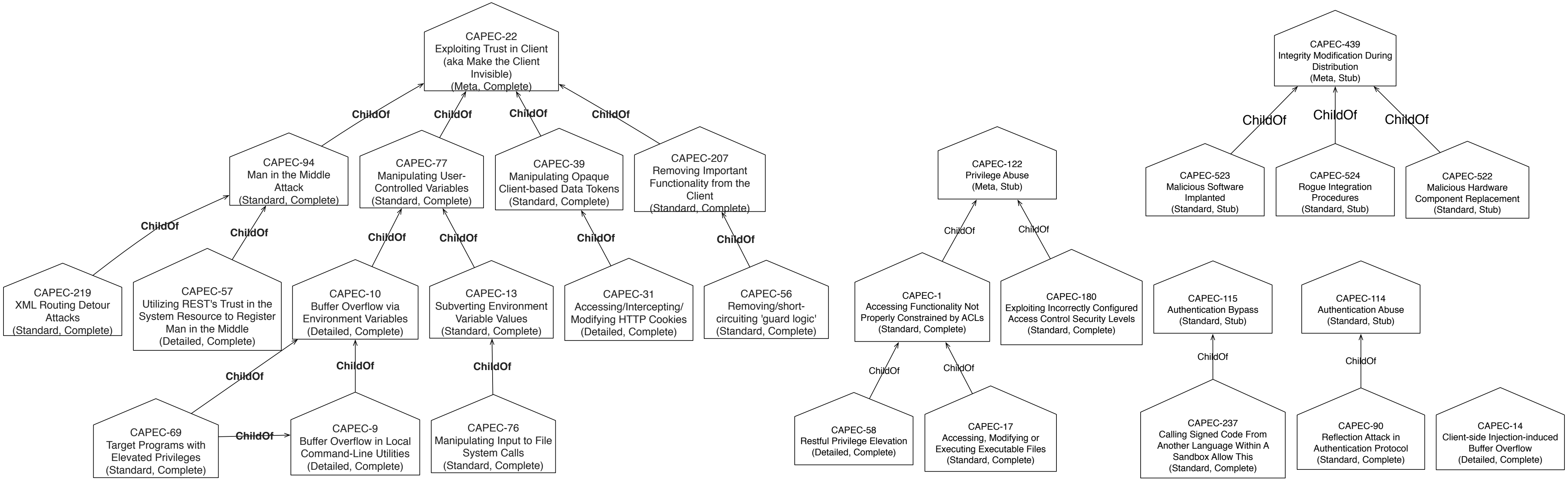
Appendix B

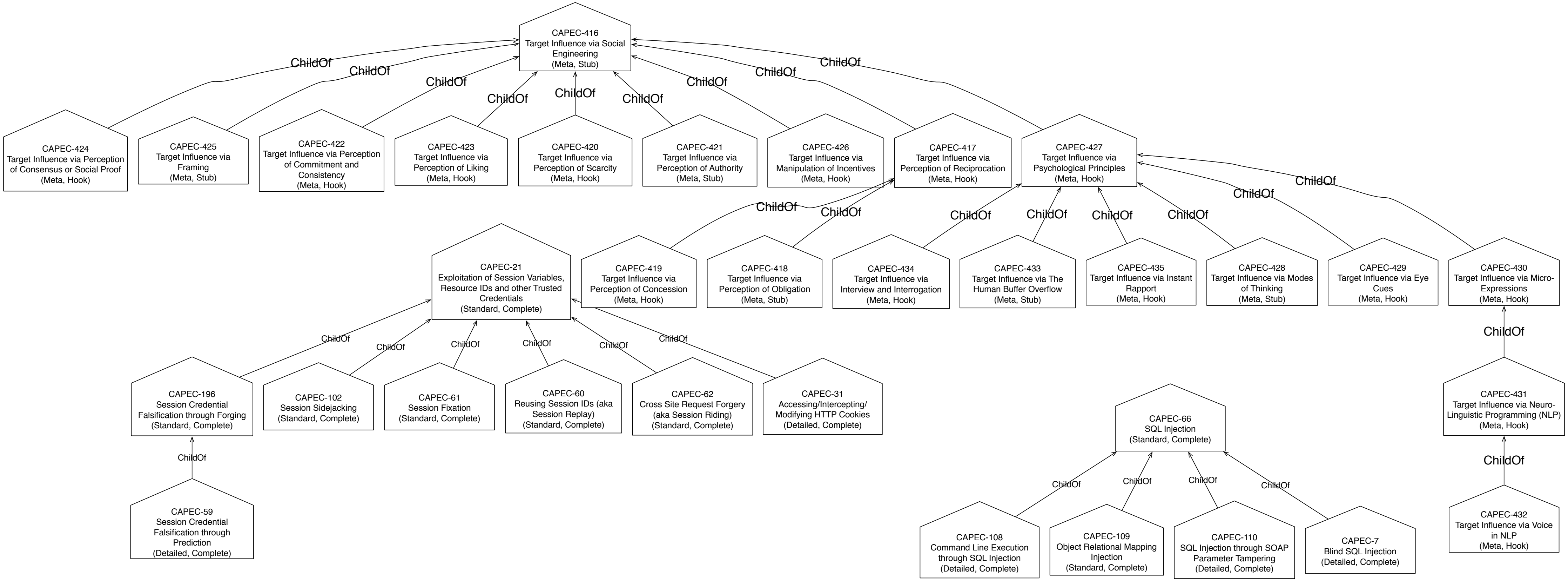
Attack Pattern Hierarchy


CAPEC-112
Brute Force
(Standard, Complete)
ChildOf
ChildOf
CAPEC-20
Encryption Brute Forcing
(Standard, Complete)
CAPEC-49
Password Brute Forcing
(Standard, Complete)
ChildOf
ChildOf
ChildOf
ChildOf
CAPEC-97
Cryptanalysis
(Meta, Complete)
CAPEC-70
Try Common(default) Usernames and Passwords
(Detailed, Complete)
CAPEC-16
Dictionary-based Password Attack
(Detailed, Complete)
CAPEC-55
Rainbow Table Password Cracking
(Standard, Complete)
CAPEC-184
Software Integrity Attacks
(Meta, Complete)
ChildOf
ChildOf
CAPEC-185
Malicious Software Download
(Meta, Complete)
CAPEC-207
Removing Important Functionality from the Client
(Standard, Complete)
ChildOf
ChildOf
CAPEC-186
Malicious Software Update
(Standard, Complete)
CAPEC-56
Removing/short-circuiting 'guard logic'
(Standard, Complete)
ChildOf
CAPEC-187
Malicious Automated Software Update
(Standard, Complete)
CAPEC-390
Bypassing Physical Security
(Meta, Stub)
ChildOf
ChildOf
CAPEC-395
Bypassing Electronic Locks and Access Controls
(Standard, Stub)
CAPEC-391
Bypassing Physical Locks
(Meta, Stub)
ChildOf
ChildOf
ChildOf
ChildOf
CAPEC-396
Bypassing Card or Badge-Based Systems
(Standard, Stub)
CAPEC-394
Using a Snap Gun Lock to Force a Lock
(Standard, Stub)
CAPEC-392
Lock Bumping
(Meta, Stub)
CAPEC-393
Lock Picking
(Standard, Stub)
ChildOf
ChildOf
ChildOf
ChildOf
CAPEC-399
Cloning RFID Cards or Chips
(Standard, Stub)
CAPEC-398
Magnetic Strip Card Brute Force Attacks
(Standard, Stub)
CAPEC-397
Cloning Magnetic Strip Cards
(Standard, Stub)
CAPEC-400
RFID Chip Deactivation or Destruction
(Standard, Stub)
CAPEC-547
Physical Destruction of Device or Component
(Meta, Stub)
CAPEC-507
Physical Theft
(Meta, Stub)


CAPEC-438
Integrity Modification During Manufacture
(Meta, Stub)
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
CAPEC-520
Counterfeit Hardware Component Inserted During Product Assembly
(Standard, Stub)
CAPEC-521
Hardware Design Specifications Are Altered
(Standard, Stub)
CAPEC-516
Hardware Component Substitution During Baselining
(Standard, Stub)
CAPEC-517
Documentation Alteration to Circumvent Dial-down
(Standard, Stub)
CAPEC-518
Documentation Alteration to Produce Under-performing Systems
(Standard, Stub)
CAPEC-519
Documentation Alteration to Cause Errors in System Design
(Standard, Stub)
CAPEC-511
Infiltration of Software Development Environment
(Standard, Stub)
CAPEC-537
Infiltration of Hardware Development Environment
(Standard, Stub)


CAPEC-100
Overflow Buffers
(Standard, Complete)
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
CAPEC-46
Overflow Variables and Tags
(Detailed, Complete)
CAPEC-47
Buffer Overflow via Parameter Expansion
(Detailed, Complete)
CAPEC-44
Overflow Binary Resource File
(Detailed, Complete)
CAPEC-45
Buffer Overflow via Symbolic Links
(Detailed, Complete)
ChildOf
CAPEC-8
Buffer Overflow in an API Call
(Detailed, Complete)
CAPEC-9
Buffer Overflow in Local Command-Line Utilities
(Detailed, Complete)
CAPEC-24
Filter Failure through Buffer Overflow
(Detailed, Complete)
CAPEC-42
MIME Conversion
(Detailed, Complete)
CAPEC-67
String Format Overflow in syslog()
(Detailed, Complete)
CAPEC-10
Buffer Overflow via Environment Variables
(Detailed, Complete)
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
CAPEC-14
Client-side Injection-induced Buffer Overflow
(Detailed, Complete)
CAPEC-69
Target Programs with Elevated Privileges
(Standard, Complete)

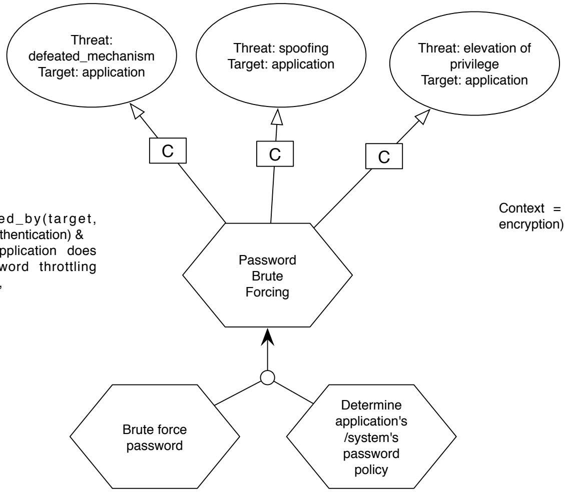

CAPEC-22
Exploiting Trust in Client (aka Make the Client Invisible)
(Meta, Complete)
ChildOf
ChildOf
ChildOf
ChildOf
CAPEC-94
Man in the Middle Attack
(Standard, Complete)
CAPEC-77
Manipulating User-Controlled Variables
(Standard, Complete)
CAPEC-39
Manipulating Opaque Client-based Data Tokens
(Standard, Complete)
CAPEC-207
Removing Important Functionality from the Client
(Standard, Complete)
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
CAPEC-219
XML Routing Detour Attacks
(Standard, Complete)
CAPEC-57
Utilizing REST's Trust in the System Resource to Register Man in the Middle
(Detailed, Complete)
CAPEC-10
Buffer Overflow via Environment Variables
(Detailed, Complete)
CAPEC-13
Subverting Environment Variable Values
(Standard, Complete)
CAPEC-31
Accessing/Intercepting/Modifying HTTP Cookies
(Detailed, Complete)
CAPEC-56
Removing/short-circuiting 'guard logic'
(Standard, Complete)
ChildOf
ChildOf
ChildOf
ChildOf
CAPEC-69
Target Programs with Elevated Privileges
(Standard, Complete)
CAPEC-9
Buffer Overflow in Local Command-Line Utilities
(Detailed, Complete)
CAPEC-76
Manipulating Input to File System Calls
(Standard, Complete)
CAPEC-122
Privilege Abuse
(Meta, Stub)
ChildOf
ChildOf
CAPEC-1
Accessing Functionality Not Properly Constrained by ACLs
(Standard, Complete)
CAPEC-180
Exploiting Incorrectly Configured Access Control Security Levels
(Standard, Complete)
ChildOf
ChildOf
CAPEC-58
Restful Privilege Elevation
(Detailed, Complete)
CAPEC-17
Accessing, Modifying or Executing Executable Files
(Standard, Complete)
CAPEC-439
Integrity Modification During Distribution
(Meta, Stub)
ChildOf
ChildOf
ChildOf
CAPEC-523
Malicious Software Implanted
(Standard, Stub)
CAPEC-524
Rogue Integration Procedures
(Standard, Stub)
CAPEC-522
Malicious Hardware Component Replacement
(Standard, Stub)
CAPEC-115
Authentication Bypass
(Standard, Stub)
ChildOf
CAPEC-237
Calling Signed Code From Another Language Within A Sandbox Allow This
(Standard, Complete)
CAPEC-114
Authentication Abuse
(Standard, Stub)
ChildOf
CAPEC-90
Reflection Attack in Authentication Protocol
(Standard, Complete)
CAPEC-14
Client-side Injection-induced Buffer Overflow
(Detailed, Complete)


CAPEC-416
Target Influence via Social Engineering
(Meta, Stub)
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
CAPEC-424
Target Influence via Perception of Consensus or Social Proof
(Meta, Hook)
CAPEC-425
Target Influence via Framing
(Meta, Stub)
CAPEC-422
Target Influence via Perception of Commitment and Consistency
(Meta, Hook)
CAPEC-423
Target Influence via Perception of Liking
(Meta, Hook)
CAPEC-420
Target Influence via Perception of Scarcity
(Meta, Hook)
CAPEC-421
Target Influence via Perception of Authority
(Meta, Stub)
CAPEC-426
Target Influence via Manipulation of Incentives
(Meta, Hook)
CAPEC-417
Target Influence via Perception of Reciprocation
(Meta, Hook)
CAPEC-427
Target Influence via Psychological Principles
(Meta, Hook)
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
CAPEC-21
Exploitation of Session Variables, Resource IDs and other Trusted Credentials
(Standard, Complete)
CAPEC-419
Target Influence via Perception of Concession
(Meta, Hook)
CAPEC-418
Target Influence via Perception of Obligation
(Meta, Stub)
CAPEC-434
Target Influence via Interview and Interrogation
(Meta, Hook)
CAPEC-433
Target Influence via The Human Buffer Overflow
(Meta, Stub)
CAPEC-435
Target Influence via Instant Rapport
(Meta, Hook)
CAPEC-428
Target Influence via Modes of Thinking
(Meta, Stub)
CAPEC-429
Target Influence via Eye Cues
(Meta, Hook)
CAPEC-430
Target Influence via Micro-Expressions
(Meta, Hook)
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
CAPEC-196
Session Credential Falsification through Forging
(Standard, Complete)
CAPEC-102
Session Sidejacking
(Standard, Complete)
CAPEC-61
Session Fixation
(Standard, Complete)
CAPEC-60
Reusing Session IDs (aka Session Replay)
(Standard, Complete)
CAPEC-62
Cross Site Request Forgery (aka Session Riding)
(Standard, Complete)
CAPEC-31
Accessing/Intercepting/ Modifying HTTP Cookies
(Detailed, Complete)
CAPEC-66
SQL Injection
(Standard, Complete)
CAPEC-431
Target Influence via Neuro-Linguistic Programming (NLP)
(Meta, Hook)
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
ChildOf
CAPEC-59
Session Credential Falsification through Prediction
(Detailed, Complete)
CAPEC-108
Command Line Execution through SQL Injection
(Detailed, Complete)
CAPEC-109
Object Relational Mapping Injection
(Standard, Complete)
CAPEC-110
SQL Injection through SOAP Parameter Tampering
(Detailed, Complete)
CAPEC-7
Blind SQL Injection
(Detailed, Complete)
CAPEC-432
Target Influence via Voice in NLP
(Meta, Hook)

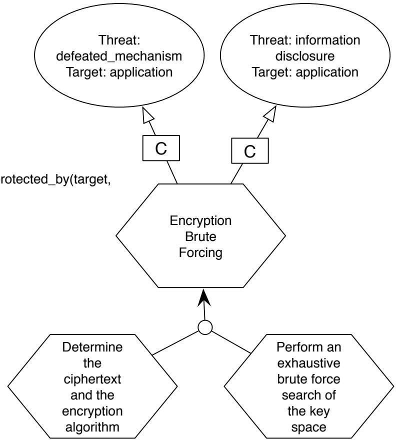
Appendix C

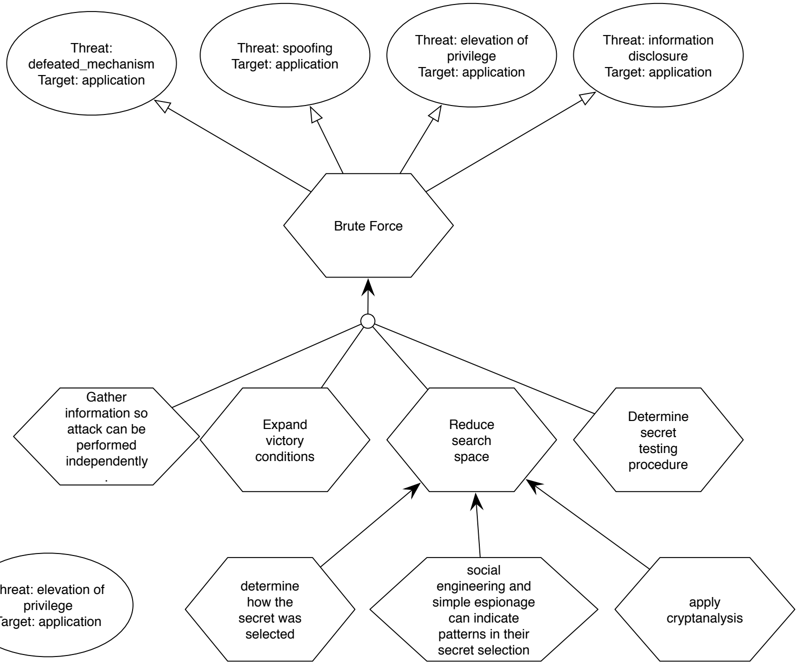
Goal Models Built from Attack Patterns

Context=protected_by(target, password_based_authentication) & other_context(An application does not have a password throttling mechanism in place),

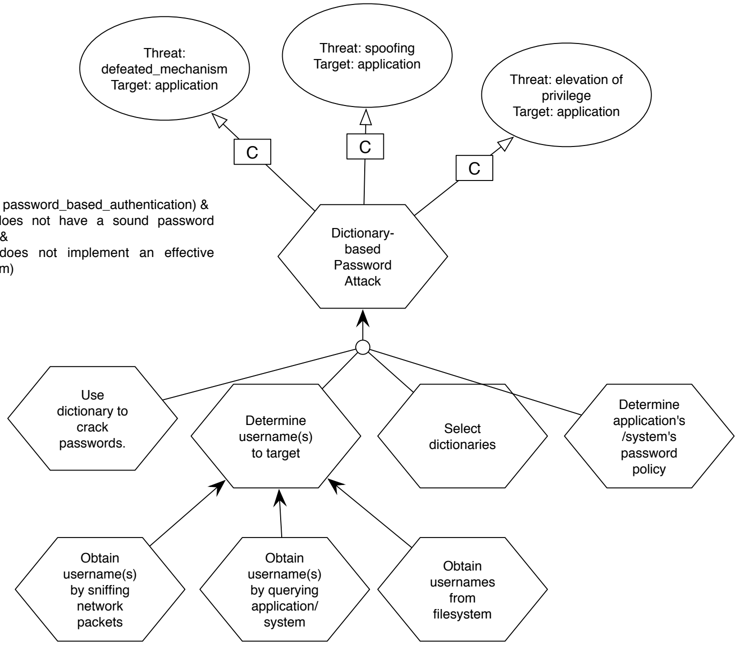

Threat: defeated_mechanism Target: application
Threat: spoofing Target: application
Threat: elevation of privilege Target: application
C
C
C
Password Brute Forcing
Brute force password
Determine application's /system's password policy

Context = protected_by(target, encryption)

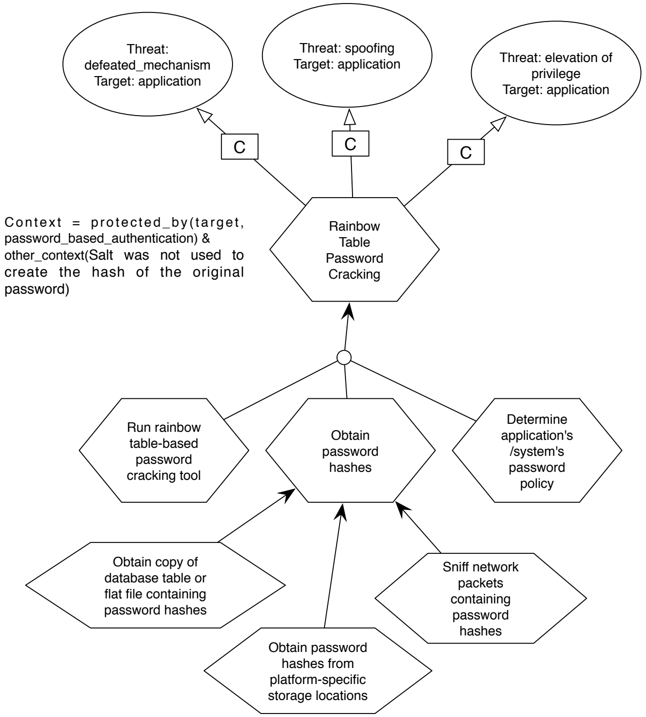

Threat: defeated_mechanism Target: application
Threat: information disclosure Target: application
C
C
Encryption Brute Forcing
Determine the ciphertext and the encryption algorithm
Perform an exhaustive brute force search of the key space

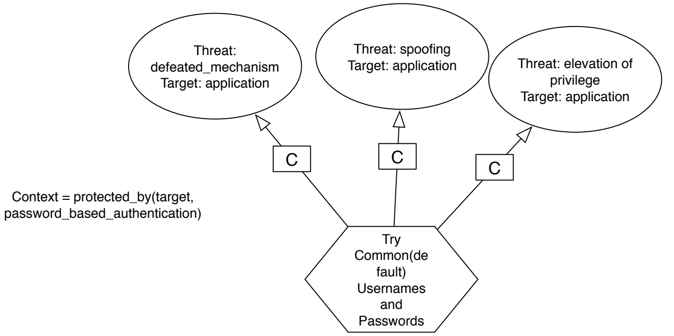

Threat: defeated_mechanism Target: application
Threat: spoofing Target: application
Threat: elevation of privilege Target: application
Threat: information disclosure Target: application
Brute Force
Gather information so attack can be performed independently .
Expand victory conditions
Reduce search space
Determine secret testing procedure
determine how the secret was selected
social engineering and simple espionage can indicate patterns in their secret selection
apply cryptanalysis

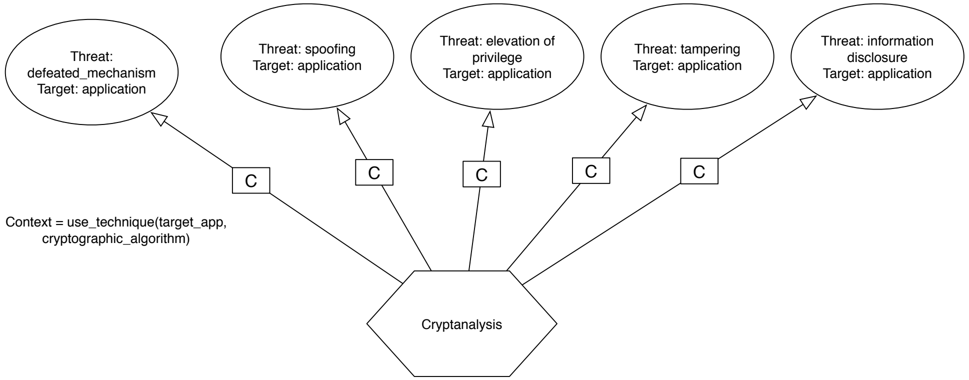
Context=protected_by(target, password_based_authentication) & other_context(The system does not have a sound password policy that is being enforced) & other_context(The system does not implement an effective password throttling mechanism)

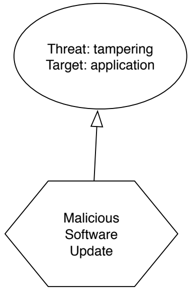

Threat: defeated_mechanism Target: application
Threat: spoofing Target: application
Threat: elevation of privilege Target: application
C
C
C
Dictionary-based Password Attack
Use dictionary to crack passwords.
Determine username(s) to target
Select dictionaries
Determine application's /system's password policy
Obtain username(s) by sniffing network packets
Obtain username(s) by querying application/ system
Obtain usernames from filesystem

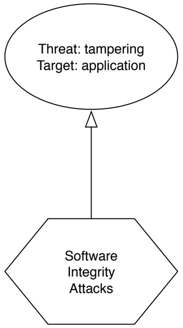
Context = protected_by(target, password_based_authentication) & other_context(Salt was not used to create the hash of the original password)

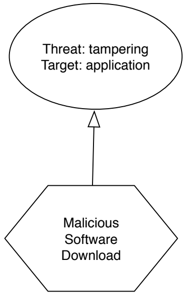

Threat: defeated_mechanism Target: application
Threat: spoofing Target: application
Threat: elevation of privilege Target: application
C
C
C
Rainbow Table Password Cracking
Run rainbow table-based password cracking tool
Obtain password hashes
Determine application's /system's password policy
Obtain copy of database table or flat file containing password hashes
Obtain password hashes from platform-specific storage locations
Sniff network packets containing password hashes

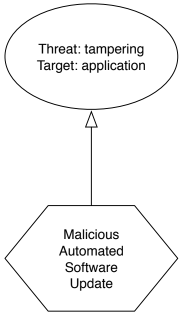

Threat: defeated_mechanism Target: application
Threat: spoofing Target: application
Threat: elevation of privilege Target: application
C
C
C
Context = protected_by(target, password_based_authentication)
Try Common(default) Usernames and Passwords

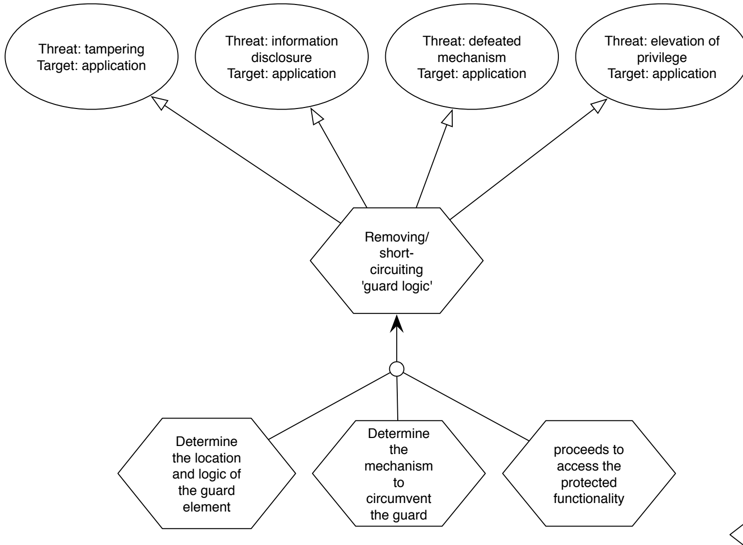

Threat: defeated_mechanism Target: application
Threat: spoofing Target: application
Threat: elevation of privilege Target: application
Threat: tampering Target: application
Threat: information disclosure Target: application
C
C
C
C
C
Context = use_technique(target_app, cryptographic_algorithm)
Cryptanalysis

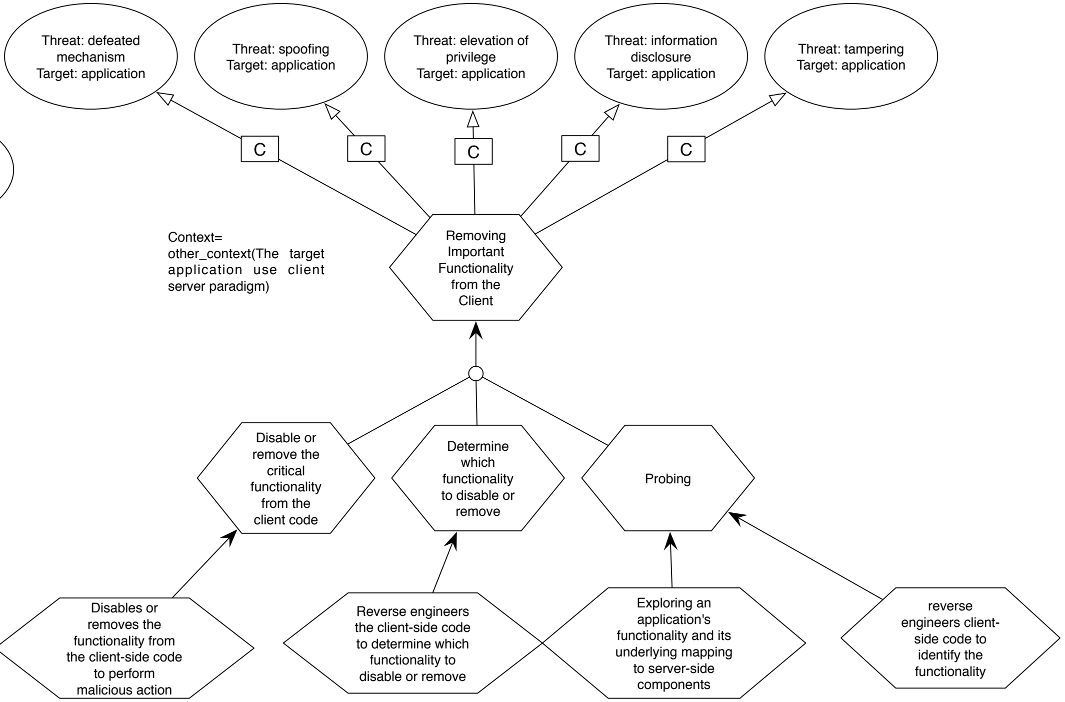

Threat: tampering Target: application
Malicious Software Update

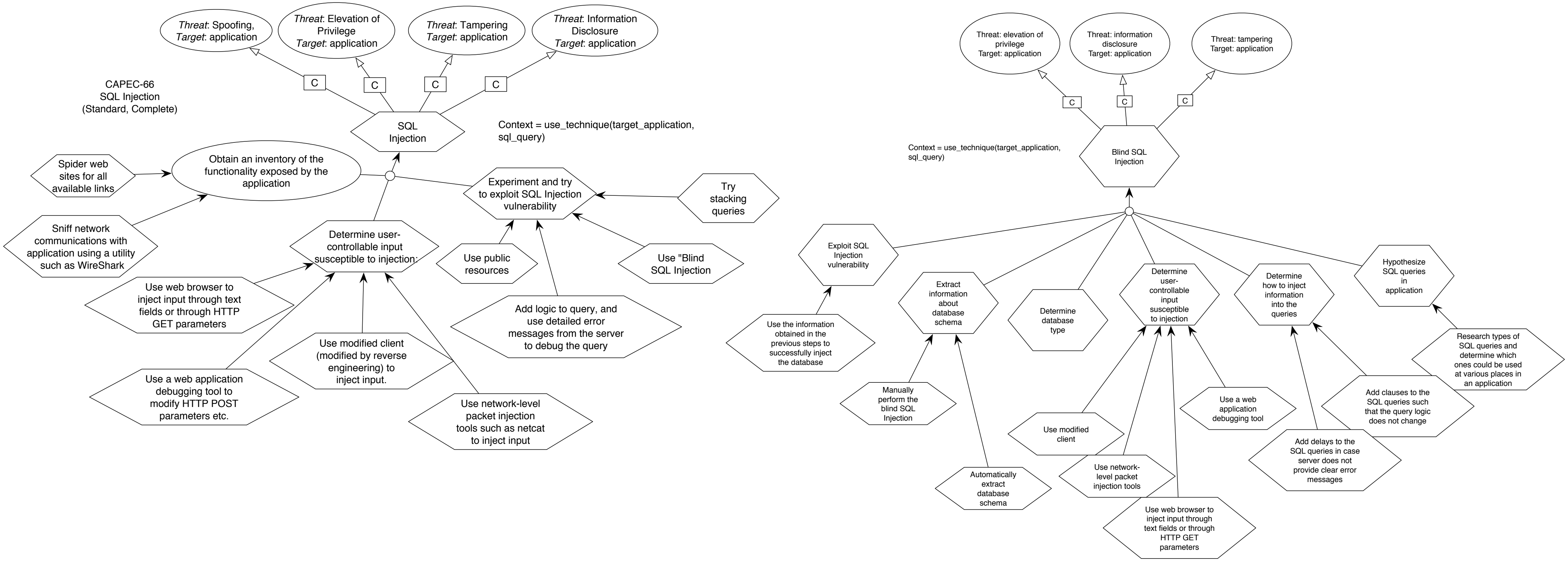

Threat: tampering Target: application
Software Integrity Attacks

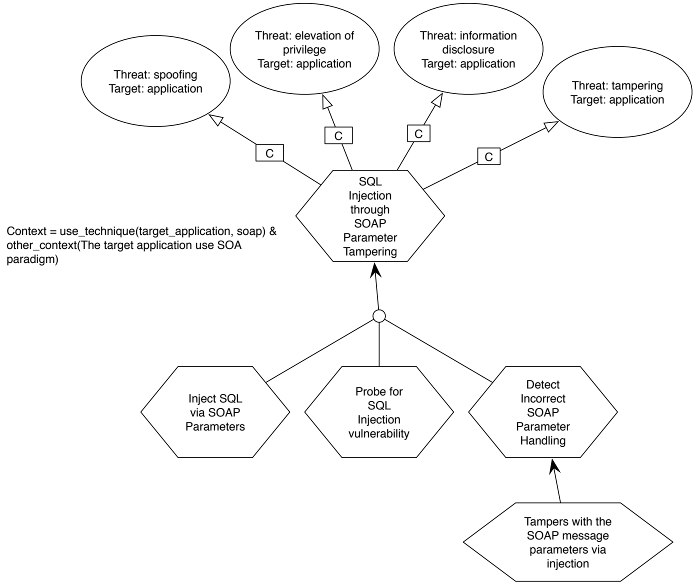

Threat: tampering Target: application
Malicious Software Download


Threat: tampering Target: application
Malicious Automated Software Update

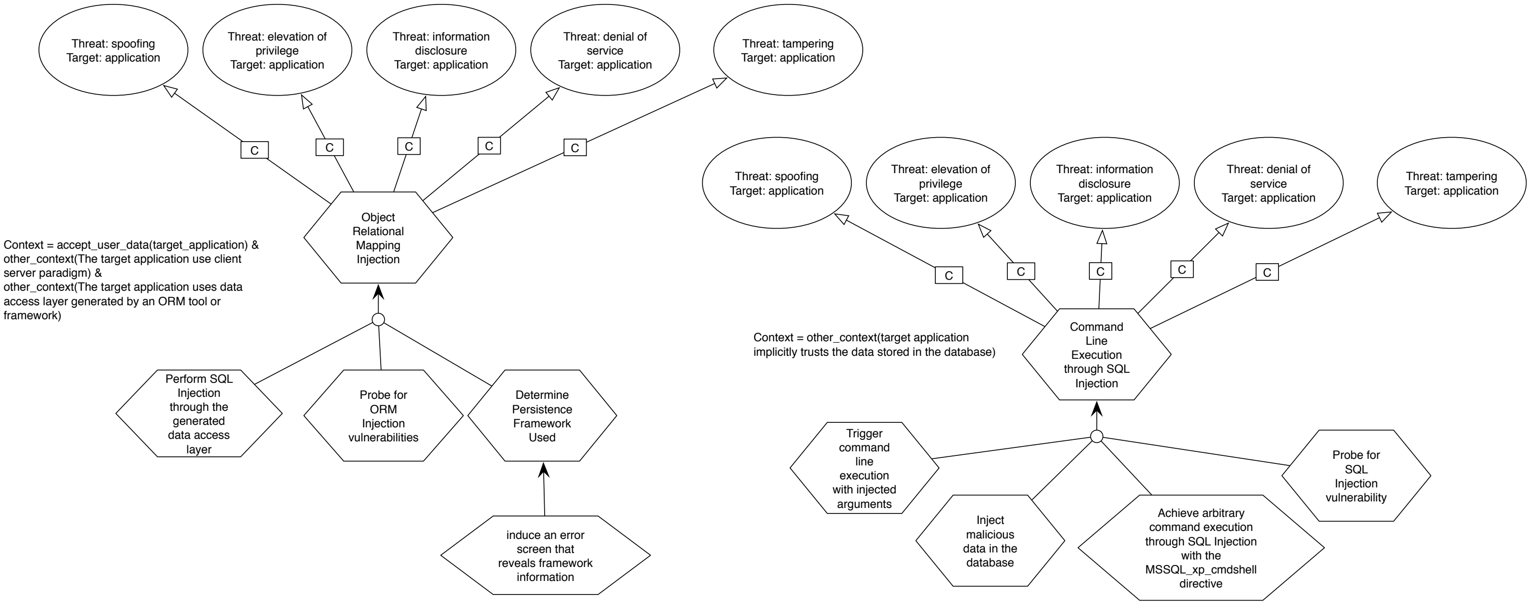

Threat: tampering Target: application
Threat: information disclosure Target: application
Threat: defeated mechanism Target: application
Threat: elevation of privilege Target: application
Removing/ short-circuiting 'guard logic'
Determine the location and logic of the guard element
Determine the mechanism to circumvent the guard
proceeds to access the protected functionality


Threat: defeated mechanism Target: application
Threat: spoofing Target: application
Threat: elevation of privilege Target: application
Threat: information disclosure Target: application
Threat: tampering Target: application
C
C
C
C
C
Context= other_context(The target application use client server paradigm)
Removing Important Functionality from the Client
Disable or remove the critical functionality from the client code
Determine which functionality to disable or remove
Probing
Disables or removes the functionality from the client-side code to perform malicious action
Reverse engineers the client-side code to determine which functionality to disable or remove
Exploring an application's functionality and its underlying mapping to server-side components
reverse engineers client-side code to identify the functionality


CAPEC-66
SQL Injection
(Standard, Complete)
Threat: Spoofing, *Target*: application
Threat: Elevation of Privilege *Target*: application
Threat: Tampering *Target*: application
Threat: Information Disclosure *Target*: application
C
C
C
C
SQL Injection
Context = use_technique(target_application, sql_query)
Spider web sites for all available links
Obtain an inventory of the functionality exposed by the application
Experiment and try to exploit SQL Injection vulnerability
Try stacking queries
Sniff network communications with application using a utility such as WireShark
Determine user-controllable input susceptible to injection:
Use public resources
Use "Blind SQL Injection
Use web browser to inject input through text fields or through HTTP GET parameters
Add logic to query, and use detailed error messages from the server to debug the query
Use modified client (modified by reverse engineering) to inject input.
Use a web application debugging tool to modify HTTP POST parameters etc.
Use network-level packet injection tools such as netcat to inject input
Threat: elevation of privilege Target: application
Threat: information disclosure Target: application
Threat: tampering Target: application
C
C
C
Context = use_technique(target_application, sql_query)
Blind SQL Injection
Exploit SQL Injection vulnerability
Extract information about database schema
Determine database type
Determine user-controllable input susceptible to injection
Determine how to inject information into the queries
Hypothesize SQL queries in application
Use the information obtained in the previous steps to successfully inject the database
Research types of SQL queries and determine which ones could be used at various places in an application
Manually perform the blind SQL Injection
Add clauses to the SQL queries such that the query logic does not change
Use a web application debugging tool
Use modified client
Add delays to the SQL queries in case server does not provide clear error messages
Automatically extract database schema
Use network-level packet injection tools
Use web browser to inject input through text fields or through HTTP GET parameters

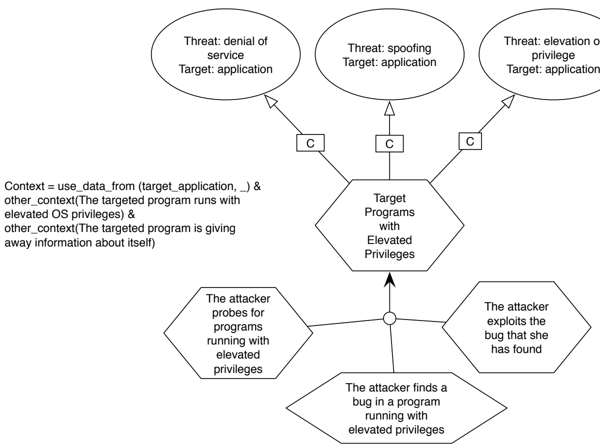

Threat: spoofing
Target: application
Threat: elevation of privilege
Target: application
Threat: information disclosure
Target: application
Threat: tampering
Target: application
C
C
C
C
SQL Injection through SOAP Parameter Tampering
Inject SQL via SOAP Parameters
Probe for SQL Injection vulnerability
Detect Incorrect SOAP Parameter Handling
Tampers with the SOAP message parameters via injection

Context = use_technique(target_application, soap) & other_context(The target application use SOA paradigm)

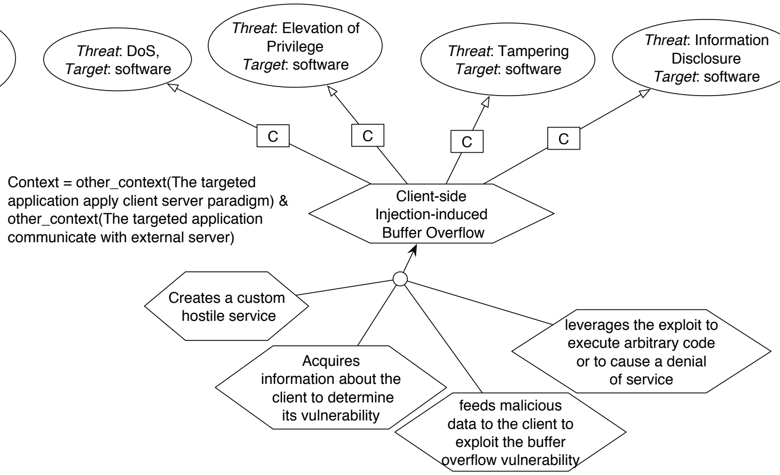

Threat: spoofing
Target: application
Threat: elevation of privilege
Target: application
Threat: information disclosure
Target: application
Threat: denial of service
Target: application
Threat: tampering
Target: application
C
C
C
C
C
Object Relational Mapping Injection
Perform SQL Injection through the generated data access layer
Probe for ORM Injection vulnerabilities
Determine Persistence Framework Used
induce an error screen that reveals framework information
Threat: spoofing
Target: application
Threat: elevation of privilege
Target: application
Threat: information disclosure
Target: application
Threat: denial of service
Target: application
Threat: tampering
Target: application
C
C
C
C
C
Command Line Execution through SQL Injection
Trigger command line execution with injected arguments
Inject malicious data in the database
Achieve arbitrary command execution through SQL Injection with the MSSQL_xp_cmdshell directive
Probe for SQL Injection vulnerability

Context = accept_user_data(target_application) & other_context(The target application use client server paradigm) & other_context(The target application uses data access layer generated by an ORM tool or framework)

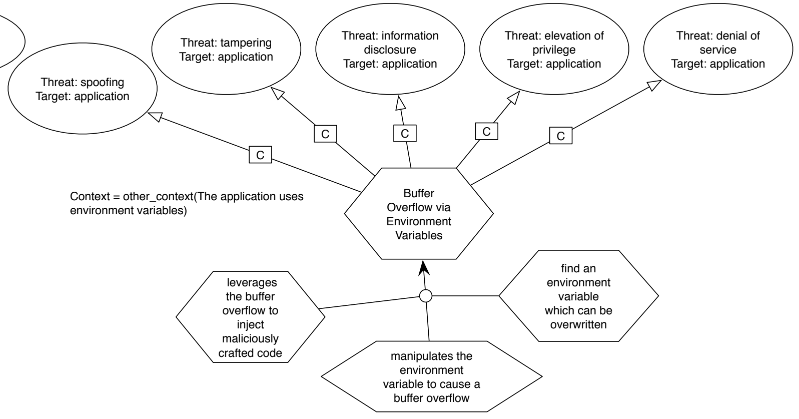
Context = other_context(target application implicitly trusts the data stored in the database)


Threat: denial of service
Target: application
Threat: spoofing
Target: application
Threat: elevation of privilege
Target: application
C
C
C
Target Programs with Elevated Privileges
The attacker probes for programs running with elevated privileges
The attacker exploits the bug that she has found
The attacker finds a bug in a program running with elevated privileges

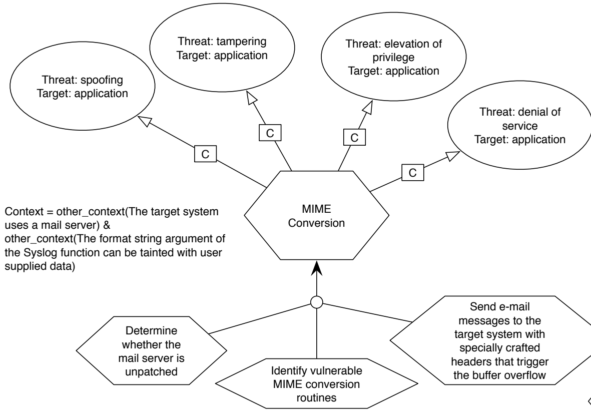
Context = use_data_from (target_application, _) & other_context(The targeted program runs with elevated OS privileges) & other_context(The targeted program is giving away information about itself)


Threat: DoS,
Target: software
Threat: Elevation of Privilege
Target: software
Threat: Tampering
Target: software
Threat: Information Disclosure
Target: software
C
C
C
C
Client-side Injection-induced Buffer Overflow
Creates a custom hostile service
Acquires information about the client to determine its vulnerability
feeds malicious data to the client to exploit the buffer overflow vulnerability
leverages the exploit to execute arbitrary code or to cause a denial of service

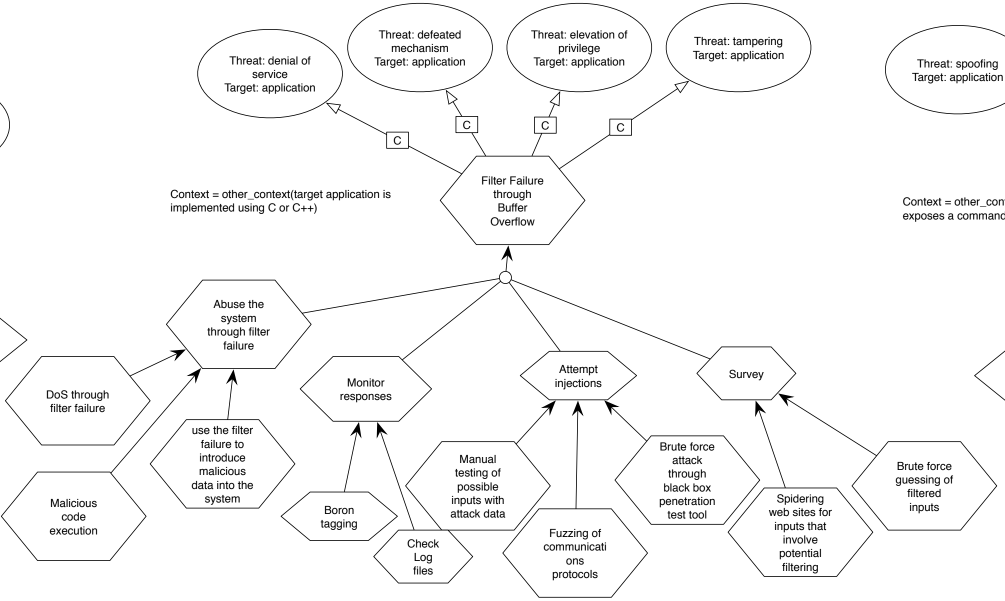
Context = other_context(The targeted application apply client server paradigm) & other_context(The targeted application communicate with external server)


Threat: spoofing
Target: application
Threat: tampering
Target: application
Threat: information disclosure
Target: application
Threat: elevation of privilege
Target: application
Threat: denial of service
Target: application
C
C
C
C
C
Buffer Overflow via Environment Variables
leverages the buffer overflow to inject maliciously crafted code
find an environment variable which can be overwritten
manipulates the environment variable to cause a buffer overflow

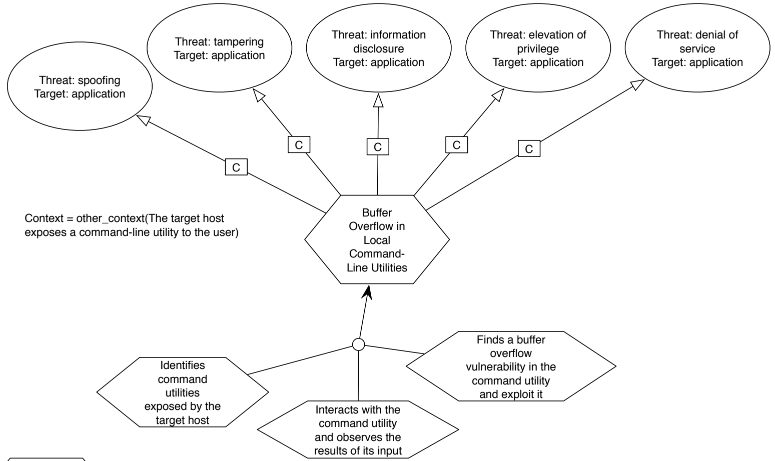
Context = other_context(The application uses environment variables)


Threat: spoofing
Target: application
Threat: tampering
Target: application
Threat: elevation of privilege
Target: application
Threat: denial of service
Target: application
C
C
C
C
MIME Conversion
Determine whether the mail server is unpatched
Identify vulnerable MIME conversion routines
Send e-mail messages to the target system with specially crafted headers that trigger the buffer overflow

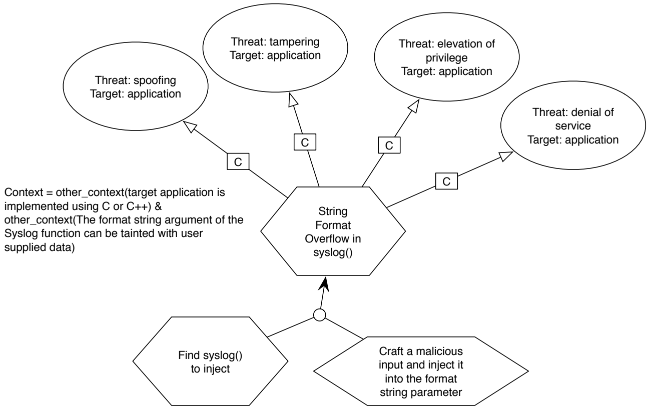
Context = other_context(The target system uses a mail server) & other_context(The format string argument of the Syslog function can be tainted with user supplied data)


Threat: denial of service
Target: application
Threat: defeated mechanism
Target: application
Threat: elevation of privilege
Target: application
Threat: tampering
Target: application
C
C
C
C
Filter Failure through Buffer Overflow
Abuse the system through filter failure
DoS through filter failure
Malicious code execution
use the filter failure to introduce malicious data into the system
Monitor responses
Boron tagging
Check Log files
Attempt injections
Manual testing of possible inputs with attack data
Fuzzing of communications protocols
Brute force attack through black box penetration test tool
Survey
Spidering web sites for inputs that involve potential filtering
Brute force guessing of filtered inputs

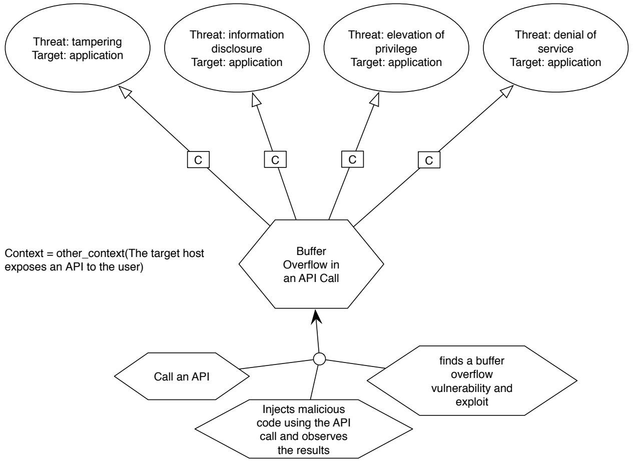
Context = other_context(target application is implemented using C or C++)


Threat: spoofing
Target: application
Threat: tampering
Target: application
Threat: information disclosure
Target: application
Threat: elevation of privilege
Target: application
Threat: denial of service
Target: application
C
C
C
C
C
Buffer Overflow in Local Command-Line Utilities
Identifies command utilities exposed by the target host
Interacts with the command utility and observes the results of its input
Finds a buffer overflow vulnerability in the command utility and exploit it

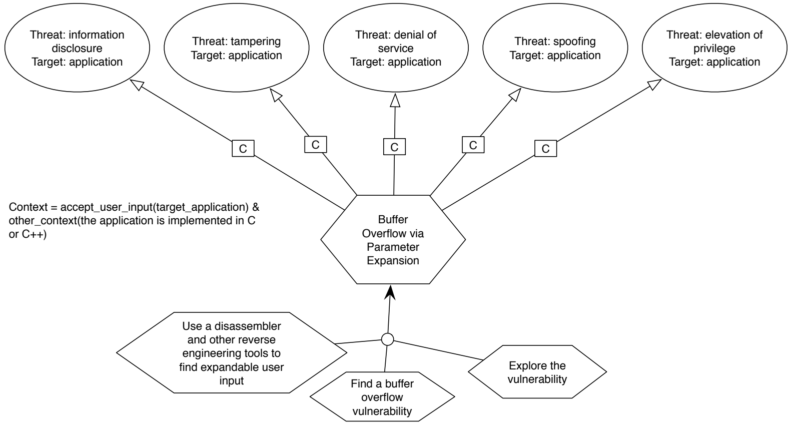
Context = other_context(The target host exposes a command-line utility to the user)


Threat: spoofing
Target: application
Threat: tampering
Target: application
Threat: elevation of privilege
Target: application
Threat: denial of service
Target: application
C
C
C
C
String Format Overflow in syslog()
Find syslog() to inject
Craft a malicious input and inject it into the format string parameter

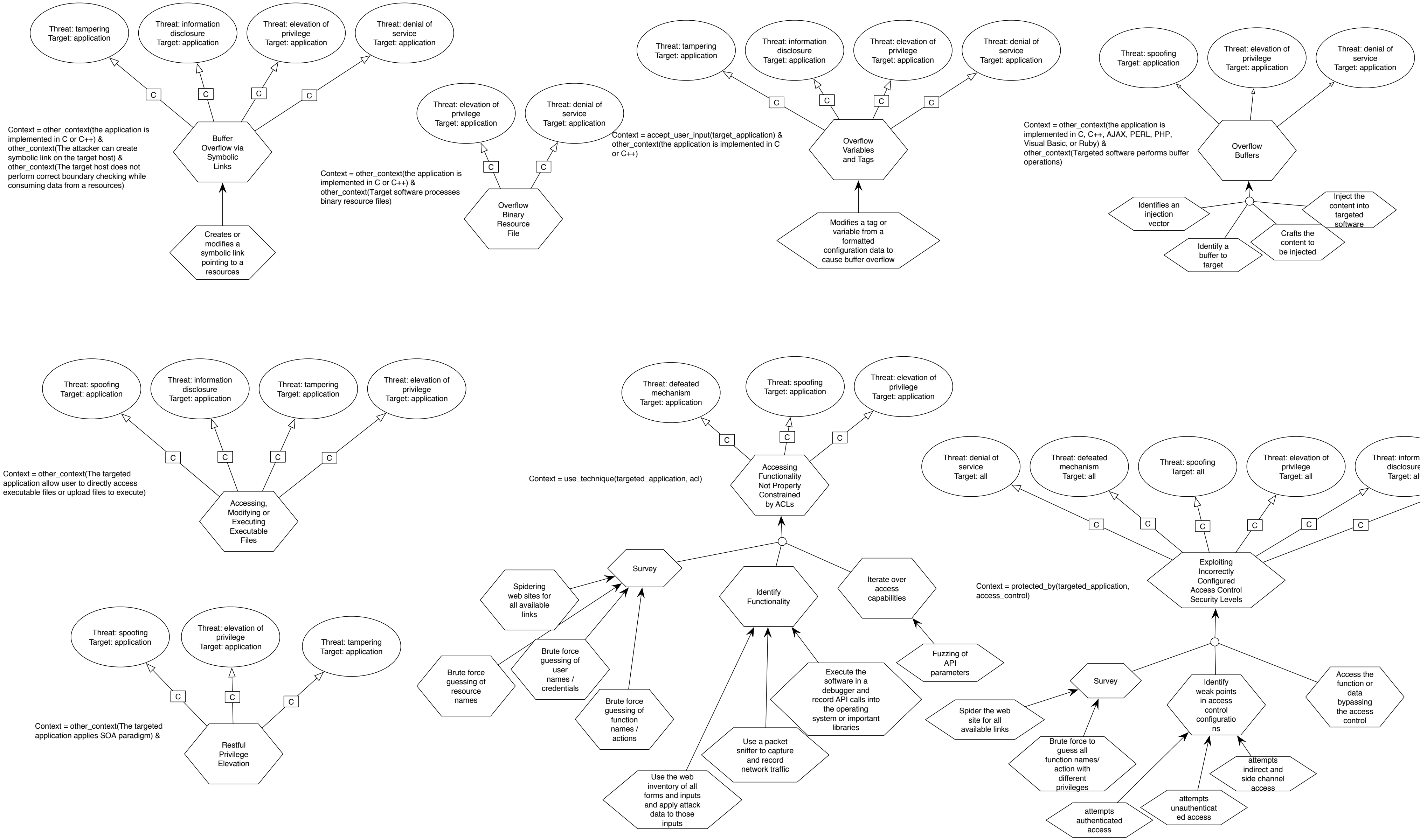
Context = other_context(target application is implemented using C or C++) & other_context(The format string argument of the Syslog function can be tainted with user supplied data)


Threat: tampering
Target: application
Threat: information disclosure
Target: application
Threat: elevation of privilege
Target: application
Threat: denial of service
Target: application
C
C
C
C
Buffer Overflow in an API Call
Call an API
Injects malicious code using the API call and observes the results
finds a buffer overflow vulnerability and exploit

Context = other_context(The target host exposes an API to the user)


Threat: information disclosure
Target: application
Threat: tampering
Target: application
Threat: denial of service
Target: application
Threat: spoofing
Target: application
Threat: elevation of privilege
Target: application
C
C
C
C
C
Buffer Overflow via Parameter Expansion
Use a disassembler and other reverse engineering tools to find expandable user input
Find a buffer overflow vulnerability
Explore the vulnerability

Context = accept_user_input(target_application) & other_context(the application is implemented in C or C++)



Threat: tampering Target: application
Threat: information disclosure Target: application
Threat: elevation of privilege Target: application
Threat: denial of service Target: application
C
C
C
C
Buffer Overflow via Symbolic Links
Creates or modifies a symbolic link pointing to a resources

Context = other_context(the application is implemented in C or C++) & other_context(The attacker can create symbolic link on the target host) & other_context(The target host does not perform correct boundary checking while consuming data from a resources)

Threat: elevation of privilege Target: application
Threat: denial of service Target: application
C
C
Overflow Binary Resource File

Context = other_context(the application is implemented in C or C++) & other_context(Target software processes binary resource files)

Threat: tampering Target: application
Threat: information disclosure Target: application
Threat: elevation of privilege Target: application
Threat: denial of service Target: application
C
C
C
C
Overflow Variables and Tags
Modifies a tag or variable from a formatted configuration data to cause buffer overflow

Context = accept_user_input(target_application) & other_context(the application is implemented in C or C++)

Threat: spoofing Target: application
Threat: elevation of privilege Target: application
Threat: denial of service Target: application
Overflow Buffers
Identifies an injection vector
Identify a buffer to target
Crafts the content to be injected
Inject the content into targeted software

Context = other_context(the application is implemented in C, C++, AJAX, PERL, PHP, Visual Basic, or Ruby) & other_context(Targeted software performs buffer operations)

Threat: spoofing Target: application
Threat: information disclosure Target: application
Threat: tampering Target: application
Threat: elevation of privilege Target: application
C
C
C
C
Accessing, Modifying or Executing Executable Files

Context = other_context(The targeted application allow user to directly access executable files or upload files to execute)

Threat: spoofing Target: application
Threat: elevation of privilege Target: application
Threat: tampering Target: application
C
C
C
Restful Privilege Elevation

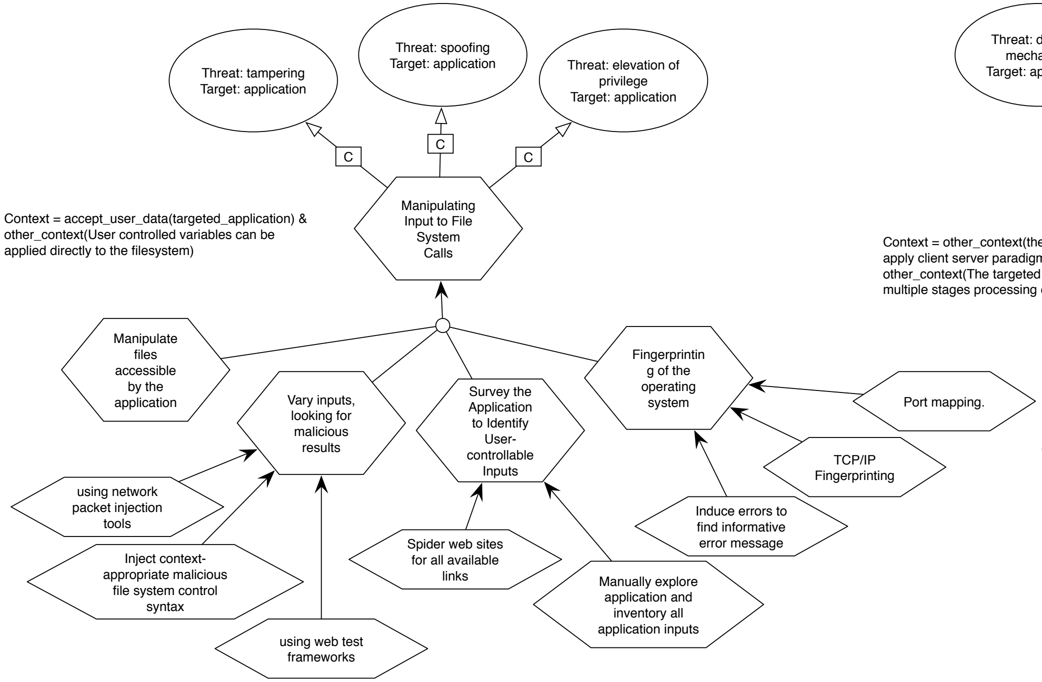
Context = other_context(The targeted application applies SOA paradigm) &

Threat: defeated mechanism Target: application
Threat: spoofing Target: application
Threat: elevation of privilege Target: application
C
C
C
Accessing Functionality Not Properly Constrained by ACLs
Survey
Identify Functionality
Iterate over access capabilities
Spidering web sites for all available links
Brute force guessing of resource names
Brute force guessing of user names / credentials
Brute force guessing of function names / actions
Use the web inventory of all forms and inputs and apply attack data to those inputs
Use a packet sniffer to capture and record network traffic
Execute the software in a debugger and record API calls into the operating system or important libraries
Fuzzing of API parameters

Context = use_technique(targeted_application, acl)

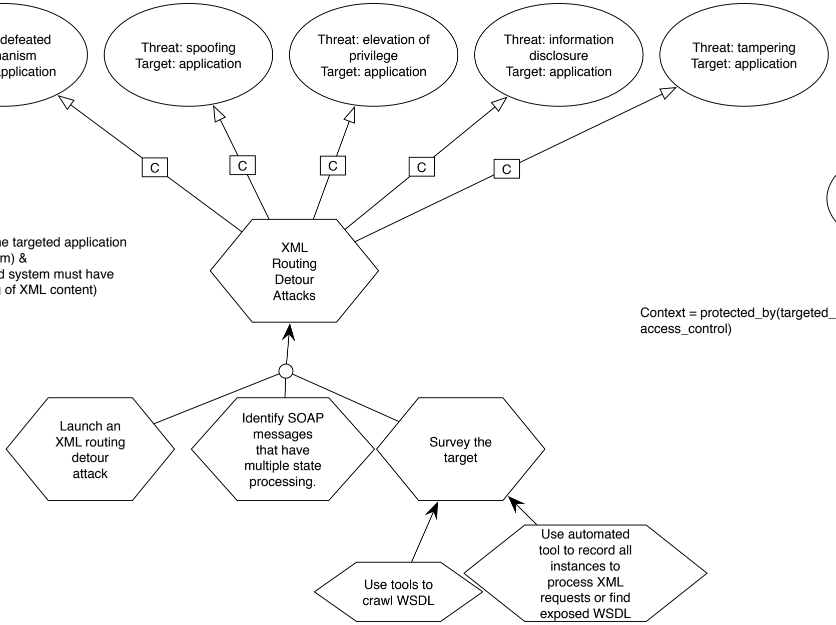
Threat: denial of service Target: all
Threat: defeated mechanism Target: all
Threat: spoofing Target: all
Threat: elevation of privilege Target: all
Threat: information disclosure Target: all
Threat: tampering Target: all
C
C
C
C
C
C
Exploiting Incorrectly Configured Access Control Security Levels
Survey
Identify weak points in access control configurations
Access the function or data bypassing the access control
Spider the web site for all available links
Brute force to guess all function names/ action with different privileges
attempts authenticated access
attempts unauthenticated access
attempts indirect and side channel access

Context = protected_by(targeted_application, access_control)

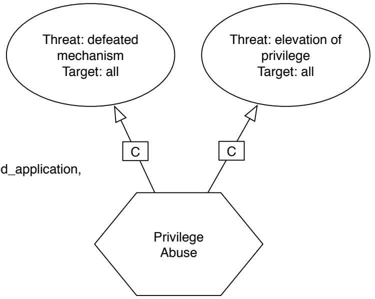

Threat: tampering
Target: application
Threat: spoofing
Target: application
Threat: elevation of privilege
Target: application
C
C
C
Manipulating Input to File System Calls
Manipulate files accessible by the application
Vary inputs, looking for malicious results
Survey the Application to Identify User-controllable Inputs
Fingerprinting of the operating system
Port mapping.
TCP/IP Fingerprinting
Induce errors to find informative error message
using network packet injection tools
Inject context-appropriate malicious file system control syntax
using web test frameworks
Spider web sites for all available links
Manually explore application and inventory all application inputs

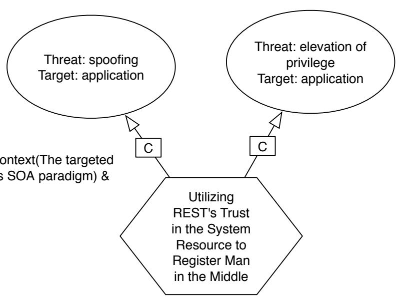
Context = accept_user_data(targeted_application) & other_context(User controlled variables can be applied directly to the filesystem)

Context = other_context(the targeted application apply client server paradigm) & other_context(The targeted system must have multiple stages processing of XML content)


Threat: defeated mechanism
Target: application
Threat: spoofing
Target: application
Threat: elevation of privilege
Target: application
Threat: information disclosure
Target: application
Threat: tampering
Target: application
C
C
C
C
C
XML Routing Detour Attacks
Launch an XML routing detour attack
Identify SOAP messages that have multiple state processing.
Survey the target
Use tools to crawl WSDL
Use automated tool to record all instances to process XML requests or find exposed WSDL

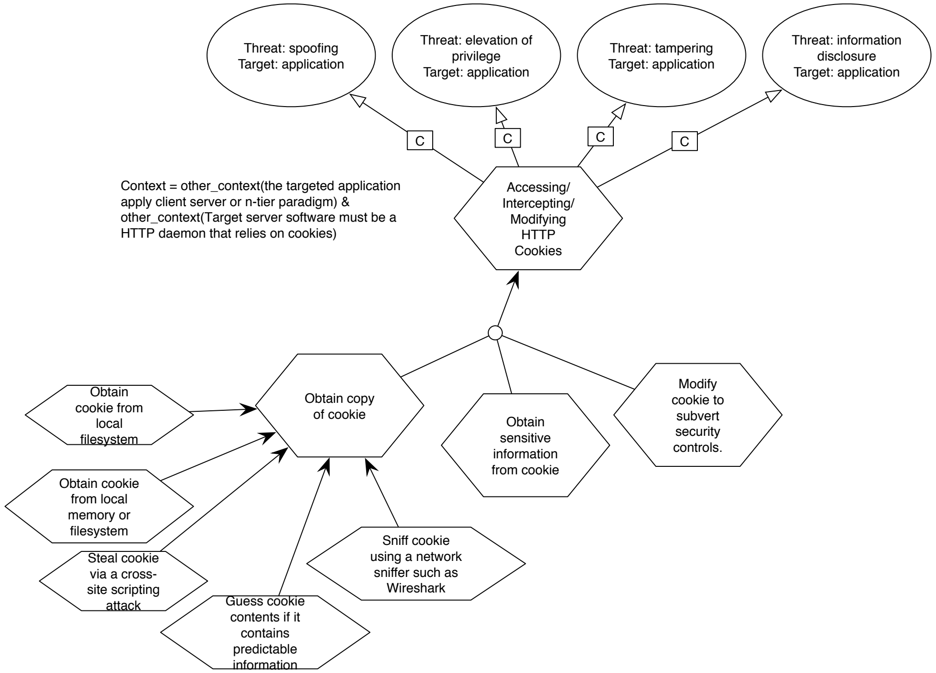
Context = protected_by(targeted_application, access_control)



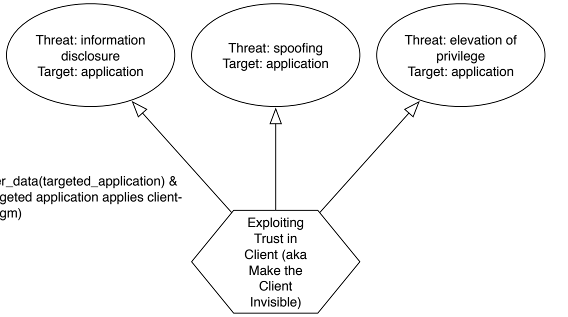

Threat: spoofing
Target: application
Threat: elevation of privilege
Target: application
C
C
Utilizing REST's Trust in the System Resource to Register Man in the Middle

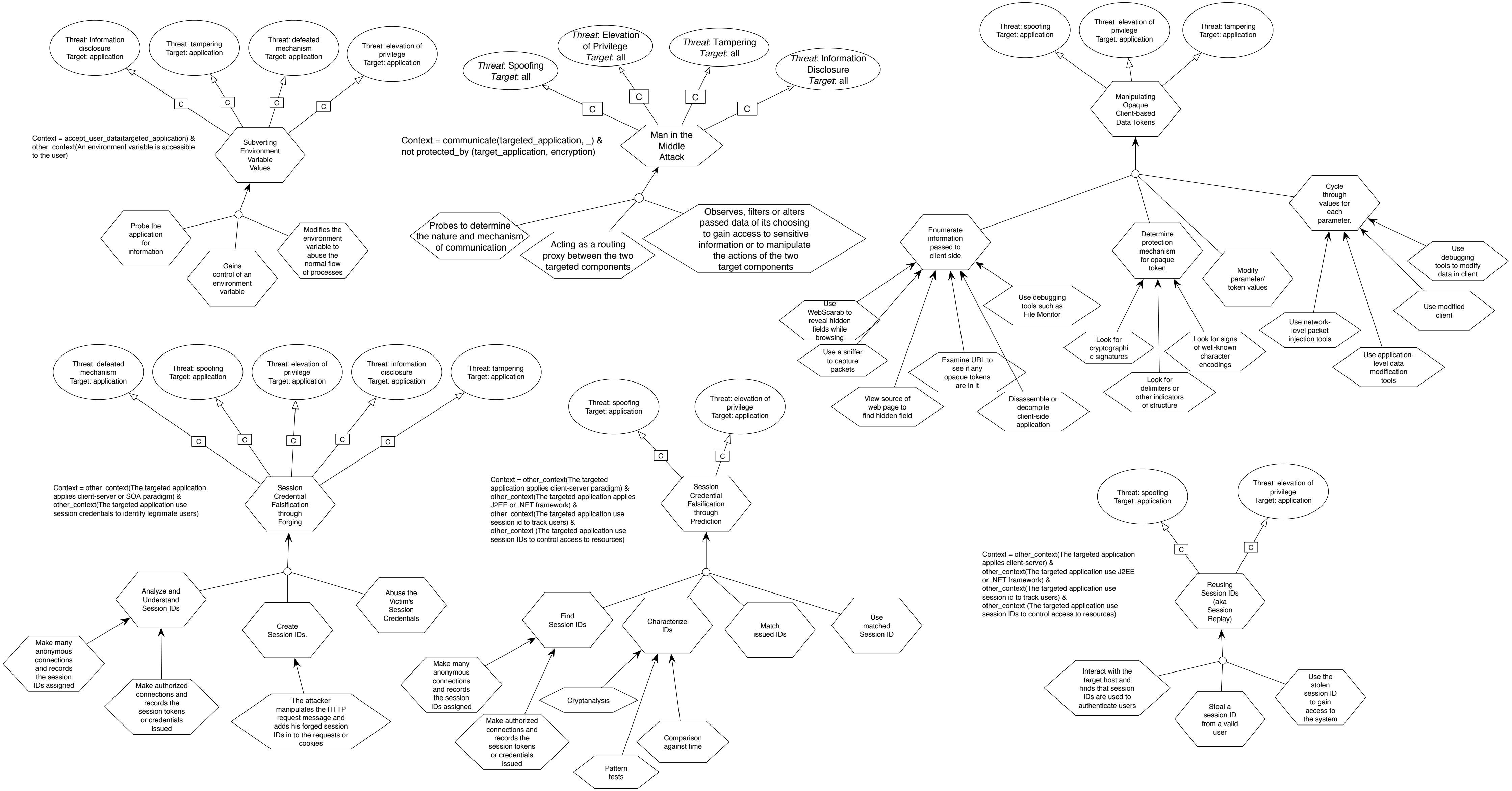
Context = other_context(The targeted application applies SOA paradigm) &

Context = other_context(the targeted application apply client server or n-tier paradigm) & other_context(Target server software must be a HTTP daemon that relies on cookies)


Threat: spoofing
Target: application
Threat: elevation of privilege
Target: application
Threat: tampering
Target: application
Threat: information disclosure
Target: application
C
C
C
C
Accessing/ Intercepting/ Modifying HTTP Cookies
Obtain copy of cookie
Obtain sensitive information from cookie
Modify cookie to subvert security controls.
Obtain cookie from local filesystem
Obtain cookie from local memory or filesystem
Steal cookie via a cross-site scripting attack
Guess cookie contents if it contains predictable information
Sniff cookie using a network sniffer such as Wireshark

Context = accept_user_data(targeted_application) & other_context(The targeted application applies client-server or n-tier paradigm)


Threat: information disclosure
Target: application
Threat: spoofing
Target: application
Threat: elevation of privilege
Target: application
Exploiting Trust in Client (aka Make the Client Invisible)


Threat: information disclosure Target: application
Threat: tampering Target: application
Threat: defeated mechanism Target: application
Threat: elevation of privilege Target: application
C
C
C
C
Subverting Environment Variable Values
Probe the application for information
Gains control of an environment variable
Modifies the environment variable to abuse the normal flow of processes
Threat: Spoofing Target: all
Threat: Elevation of Privilege Target: all
Threat: Tampering Target: all
Threat: Information Disclosure Target: all
C
C
C
C
Man in the Middle Attack
Probes to determine the nature and mechanism of communication
Acting as a routing proxy between the two targeted components
Observes, filters or alters passed data of its choosing to gain access to sensitive information or to manipulate the actions of the two target components
Threat: spoofing Target: application
Threat: elevation of privilege Target: application
Threat: tampering Target: application
Manipulating Opaque Client-based Data Tokens
Enumerate information passed to client side
Determine protection mechanism for opaque token
Modify parameter/ token values
Cycle through values for each parameter.
Use WebScarab to reveal hidden fields while browsing
Use a sniffer to capture packets
View source of web page to find hidden field
Examine URL to see if any opaque tokens are in it
Disassemble or decompile client-side application
Use debugging tools such as File Monitor
Look for cryptographic signatures
Look for delimiters or other indicators of structure
Look for signs of well-known character encodings
Use network-level packet injection tools
Use application-level data modification tools
Use modified client
Use debugging tools to modify data in client
Threat: defeated mechanism Target: application
Threat: spoofing Target: application
Threat: elevation of privilege Target: application
Threat: information disclosure Target: application
Threat: tampering Target: application
C
C
C
C
C
Session Credential Falsification through Forging
Analyze and Understand Session IDs
Create Session IDs.
Abuse the Victim's Session Credentials
Make many anonymous connections and records the session IDs assigned
Make authorized connections and records the session tokens or credentials issued
The attacker manipulates the HTTP request message and adds his forged session IDs in to the requests or cookies
Threat: spoofing Target: application
Threat: elevation of privilege Target: application
C
C
Session Credential Falsification through Prediction
Find Session IDs
Characterize IDs
Match issued IDs
Use matched Session ID
Make many anonymous connections and records the session IDs assigned
Make authorized connections and records the session tokens or credentials issued
Cryptanalysis
Pattern tests
Comparison against time
Threat: spoofing Target: application
Threat: elevation of privilege Target: application
C
C
Reusing Session IDs (aka Session Replay)
Interact with the target host and finds that session IDs are used to authenticate users
Steal a session ID from a valid user
Use the stolen session ID to gain access to the system

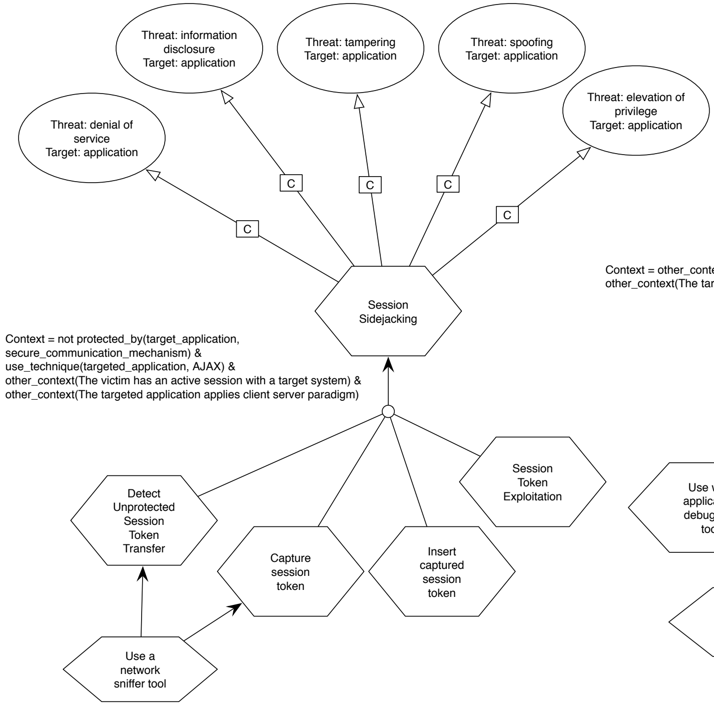
Context = accept_user_data(targeted_application) & other_context(An environment variable is accessible to the user)

Context = communicate(targeted_application, _) & not protected_by (target_application, encryption)

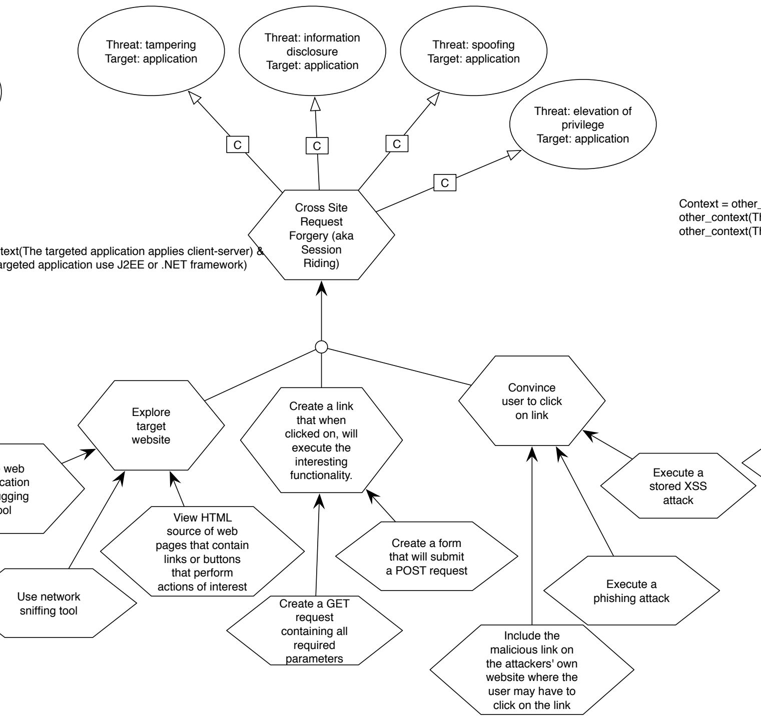
Context = other_context(The targeted application applies client-server or SOA paradigm) & other_context(The targeted application use session credentials to identify legitimate users)

Context = other_context(The targeted application applies client-server paradigm) & other_context(The targeted application applies J2EE or .NET framework) & other_context(The targeted application use session id to track users) & other_context (The targeted application use session IDs to control access to resources)

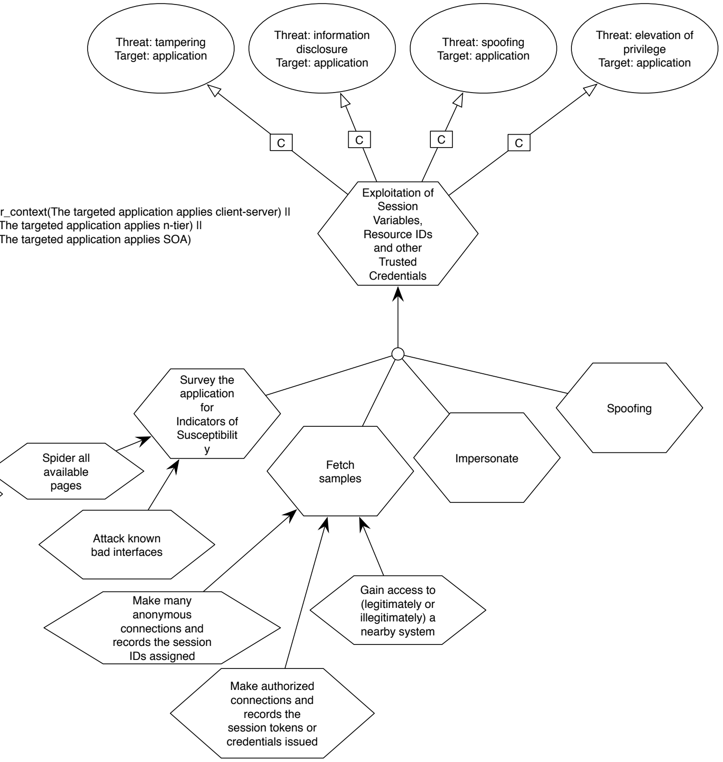
Context = other_context(The targeted application applies client-server) & other_context(The targeted application use J2EE or .NET framework) & other_context(The targeted application use session id to track users) & other_context (The targeted application use session IDs to control access to resources)


Threat: denial of service
Target: application
Threat: information disclosure
Target: application
Threat: tampering
Target: application
Threat: spoofing
Target: application
Threat: elevation of privilege
Target: application
C
C
C
C
C
Session Sidejacking
Detect Unprotected Session Token Transfer
Capture session token
Insert captured session token
Session Token Exploitation
Use a network sniffer tool

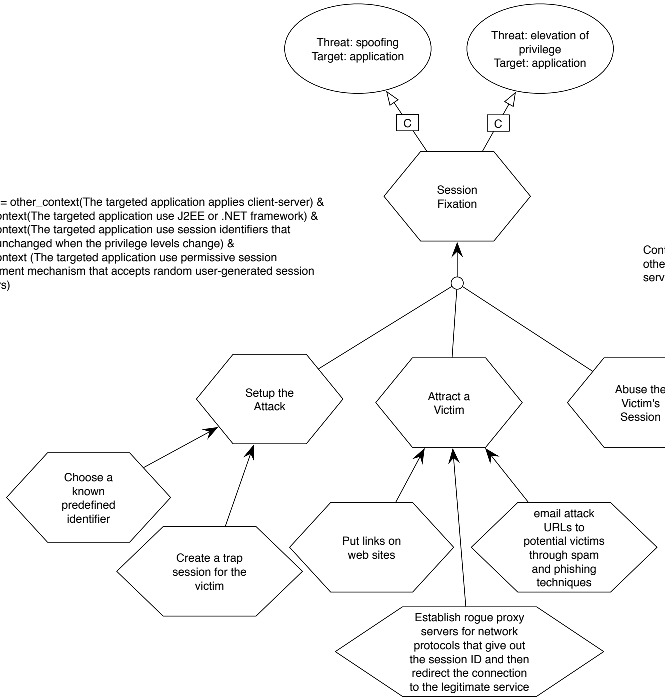
Context = not protected_by(target_application, secure_communication_mechanism) & use_technique(targeted_application, AJAX) & other_context(The victim has an active session with a target system) & other_context(The targeted application applies client server paradigm)


Threat: tampering
Target: application
Threat: information disclosure
Target: application
Threat: spoofing
Target: application
Threat: elevation of privilege
Target: application
C
C
C
C
Cross Site Request Forgery (aka Session Riding)
Explore target website
Create a link that when clicked on, will execute the interesting functionality.
Convince user to click on link
Use web application debugging tool
Use network sniffing tool
View HTML source of web pages that contain links or buttons that perform actions of interest
Create a GET request containing all required parameters
Create a form that will submit a POST request
Execute a stored XSS attack
Execute a phishing attack
Include the malicious link on the attackers' own website where the user may have to click on the link

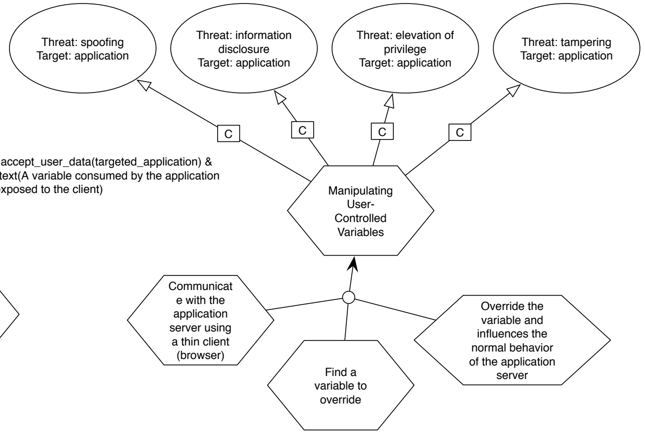
Context = other_context(The targeted application applies client-server) & other_context(The targeted application use J2EE or .NET framework)


Threat: tampering
Target: application
Threat: information disclosure
Target: application
Threat: spoofing
Target: application
Threat: elevation of privilege
Target: application
C
C
C
C
Exploitation of Session Variables, Resource IDs and other Trusted Credentials
Survey the application for Indicators of Susceptibility
Fetch samples
Impersonate
Spoofing
Spider all available pages
Attack known bad interfaces
Make many anonymous connections and records the session IDs assigned
Make authorized connections and records the session tokens or credentials issued
Gain access to (legitimately or illegitimately) a nearby system

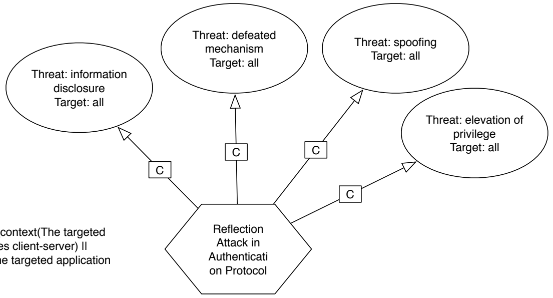
Context = other_context(The targeted application applies client-server) || other_context(The targeted application applies n-tier) || other_context(The targeted application applies SOA)


Threat: spoofing
Target: application
Threat: elevation of privilege
Target: application
C
C
Session Fixation
Setup the Attack
Attract a Victim
Abuse the Victim's Session
Choose a known predefined identifier
Create a trap session for the victim
Put links on web sites
email attack URLs to potential victims through spam and phishing techniques
Establish rogue proxy servers for network protocols that give out the session ID and then redirect the connection to the legitimate service

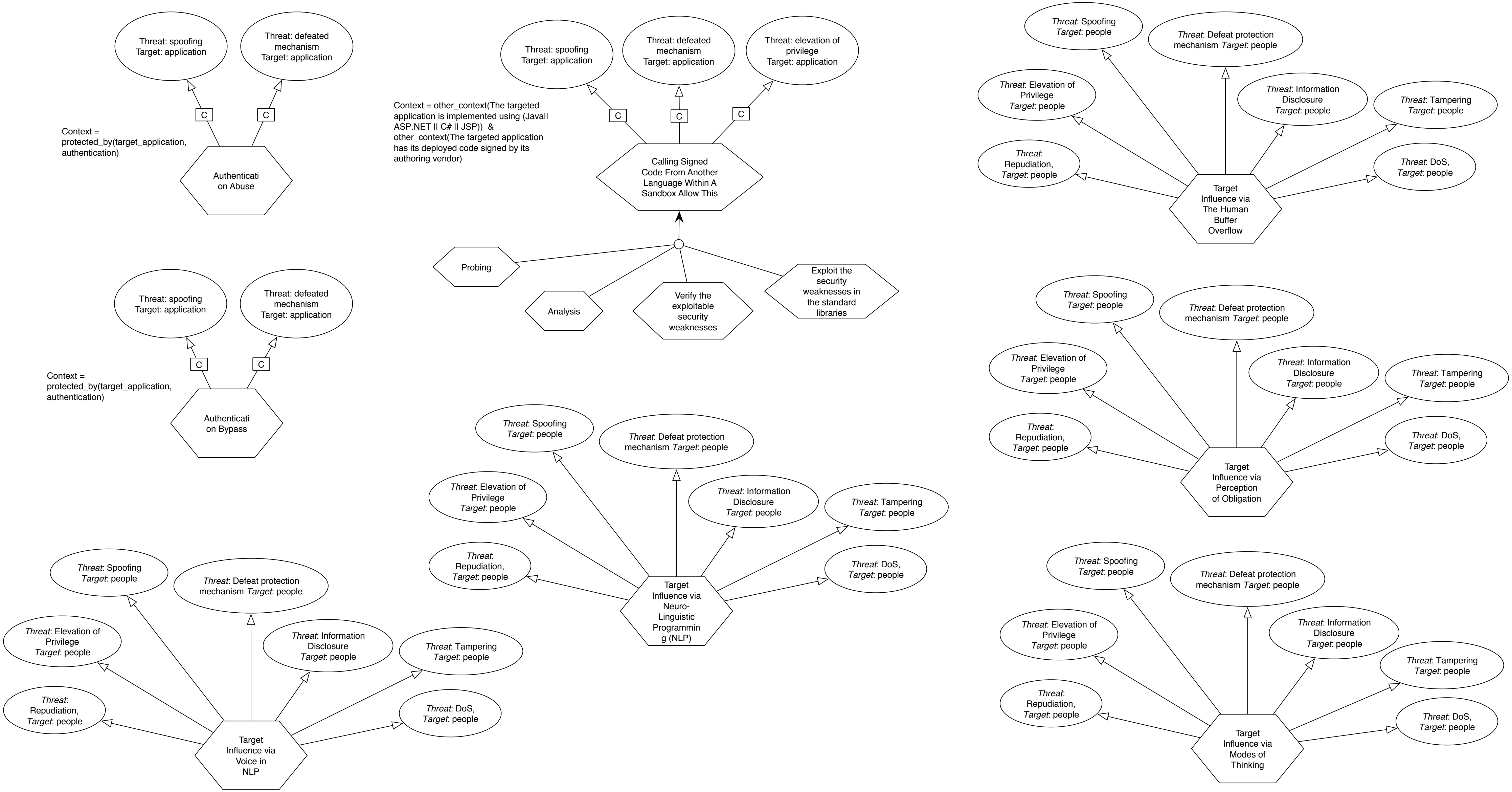
Context = other_context(The targeted application applies client-server) & other_context(The targeted application use J2EE or .NET framework) & other_context(The targeted application use session identifiers that remain unchanged when the privilege levels change) & other_context (The targeted application use permissive session management mechanism that accepts random user-generated session identifiers)


Threat: spoofing
Target: application
Threat: information disclosure
Target: application
Threat: elevation of privilege
Target: application
Threat: tampering
Target: application
C
C
C
C
Manipulating User-Controlled Variables
Communicate with the application server using a thin client (browser)
Find a variable to override
Override the variable and influences the normal behavior of the application server

Context = accept_user_data(targeted_application) & other_context(A variable consumed by the application server is exposed to the client)


Threat: information disclosure
Target: all
Threat: defeated mechanism
Target: all
Threat: spoofing
Target: all
Threat: elevation of privilege
Target: all
C
C
C
C
Reflection Attack in Authentication Protocol

Context = other_context(The targeted application applies client-server) || other_context(The targeted application applies SOA)

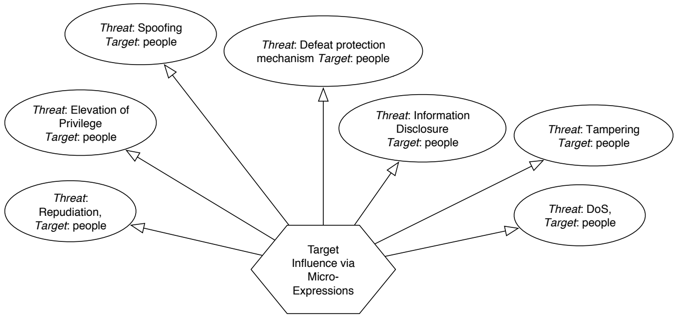


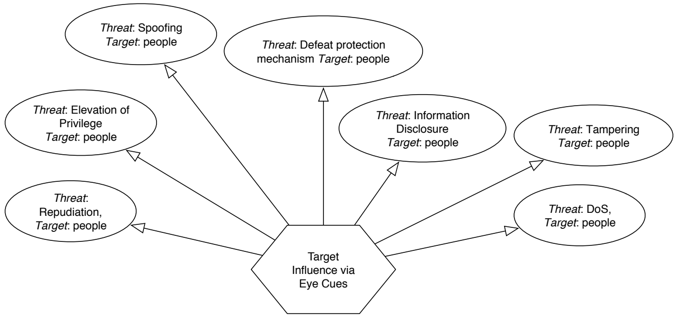
Threat: spoofing Target: application
Threat: defeated mechanism Target: application
C
C
Authentication Abuse
Threat: spoofing Target: application
Threat: defeated mechanism Target: application
C
C
Authentication Bypass
Threat: spoofing Target: application
Threat: defeated mechanism Target: application
Threat: elevation of privilege Target: application
C
C
C
Calling Signed Code From Another Language Within A Sandbox Allow This
Probing
Analysis
Verify the exploitable security weaknesses
Exploit the security weaknesses in the standard libraries
Threat: Spoofing *Target*: people
Threat: Defeat protection mechanism *Target*: people
Threat: Elevation of Privilege *Target*: people
Threat: Information Disclosure *Target*: people
Threat: Tampering *Target*: people
Threat: Repudiation, *Target*: people
Threat: DoS, *Target*: people
Target Influence via The Human Buffer Overflow
Threat: Spoofing *Target*: people
Threat: Defeat protection mechanism *Target*: people
Threat: Elevation of Privilege *Target*: people
Threat: Information Disclosure *Target*: people
Threat: Tampering *Target*: people
Threat: Repudiation, *Target*: people
Threat: DoS, *Target*: people
Target Influence via Perception of Obligation
Threat: Spoofing *Target*: people
Threat: Defeat protection mechanism *Target*: people
Threat: Elevation of Privilege *Target*: people
Threat: Information Disclosure *Target*: people
Threat: Tampering *Target*: people
Threat: Repudiation, *Target*: people
Threat: DoS, *Target*: people
Target Influence via Neuro-Linguistic Programming (NLP)
Threat: Spoofing *Target*: people
Threat: Defeat protection mechanism *Target*: people
Threat: Elevation of Privilege *Target*: people
Threat: Information Disclosure *Target*: people
Threat: Tampering *Target*: people
Threat: Repudiation, *Target*: people
Threat: DoS, *Target*: people
Target Influence via Voice in NLP
Threat: Spoofing *Target*: people
Threat: Defeat protection mechanism *Target*: people
Threat: Elevation of Privilege *Target*: people
Threat: Information Disclosure *Target*: people
Threat: Tampering *Target*: people
Threat: Repudiation, *Target*: people
Threat: DoS, *Target*: people
Target Influence via Modes of Thinking

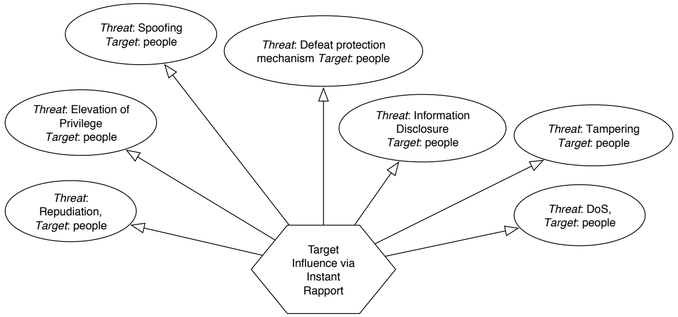
Context = protected_by(target_application, authentication)

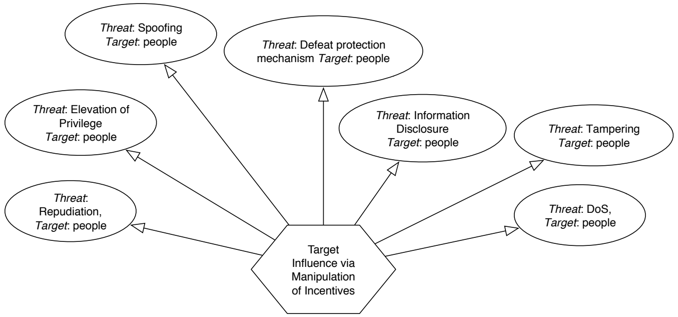
Context = protected_by(target_application, authentication)

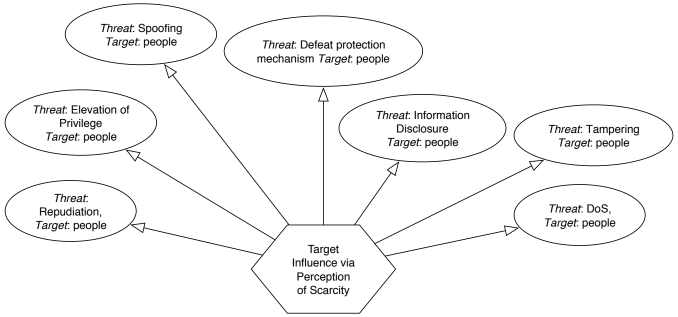
Context = other_context(The targeted application is implemented using (Java|| ASP.NET || C# || JSP)) & other_context(The targeted application has its deployed code signed by its authoring vendor)

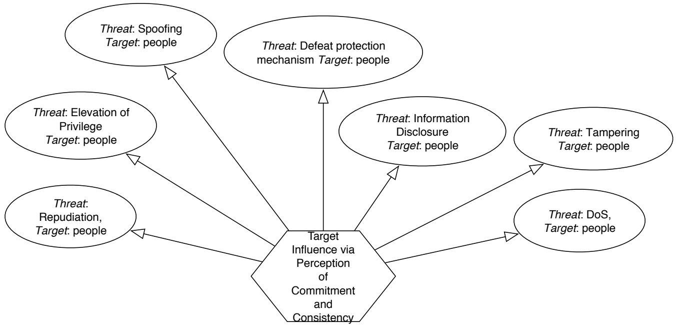

Target Influence via Micro-Expressions
Threat: Spoofing *Target*: people
Threat: Defeat protection mechanism *Target*: people
Threat: Elevation of Privilege *Target*: people
Threat: Information Disclosure *Target*: people
Threat: Tampering *Target*: people
Threat: Repudiation, *Target*: people
Threat: DoS, *Target*: people

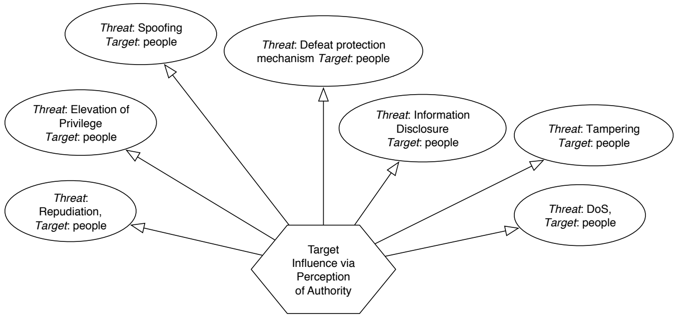

Target Influence via Eye Cues
Threat: Spoofing *Target*: people
Threat: Defeat protection mechanism *Target*: people
Threat: Elevation of Privilege *Target*: people
Threat: Information Disclosure *Target*: people
Threat: Tampering *Target*: people
Threat: Repudiation, *Target*: people
Threat: DoS, *Target*: people

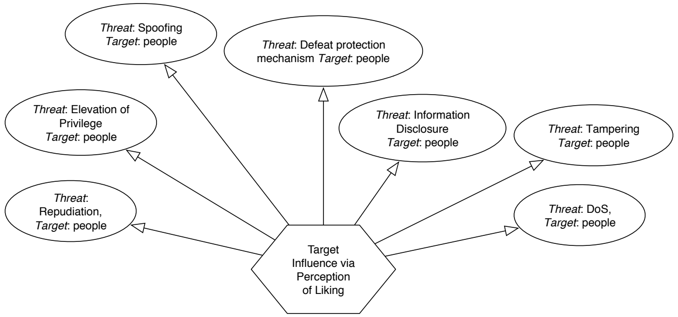

Target Influence via Instant Rapport
Threat: Spoofing *Target*: people
Threat: Defeat protection mechanism *Target*: people
Threat: Elevation of Privilege *Target*: people
Threat: Information Disclosure *Target*: people
Threat: Tampering *Target*: people
Threat: Repudiation, *Target*: people
Threat: DoS, *Target*: people

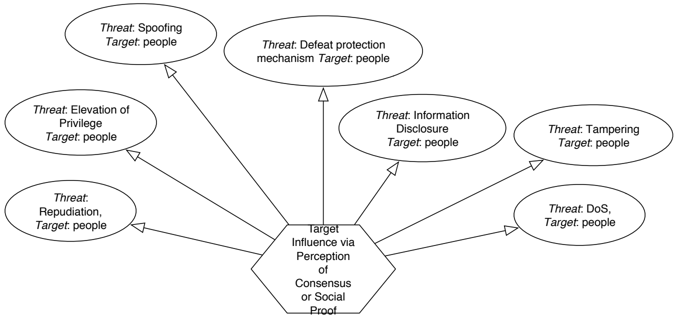

Target Influence via Manipulation of Incentives
Threat: Spoofing *Target*: people
Threat: Defeat protection mechanism *Target*: people
Threat: Elevation of Privilege *Target*: people
Threat: Information Disclosure *Target*: people
Threat: Tampering *Target*: people
Threat: Repudiation, *Target*: people
Threat: DoS, *Target*: people

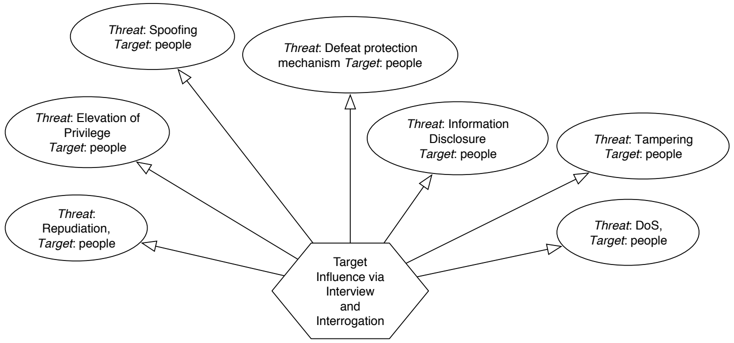

Target Influence via Perception of Scarcity
Threat: Spoofing *Target*: people
Threat: Defeat protection mechanism *Target*: people
Threat: Elevation of Privilege *Target*: people
Threat: Information Disclosure *Target*: people
Threat: Tampering *Target*: people
Threat: Repudiation, *Target*: people
Threat: DoS, *Target*: people

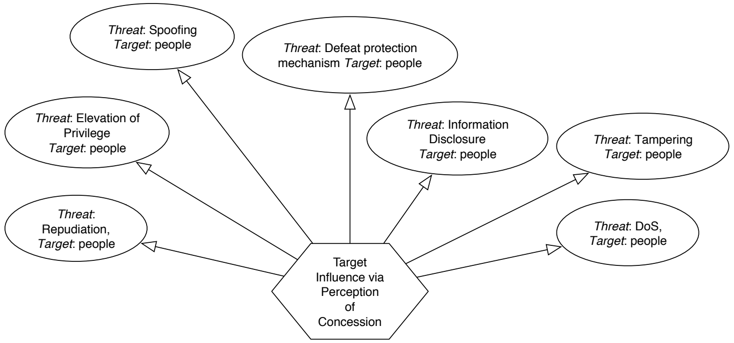

Target Influence via Perception of Commitment and Consistency
Threat: Spoofing *Target*: people
Threat: Defeat protection mechanism *Target*: people
Threat: Elevation of Privilege *Target*: people
Threat: Information Disclosure *Target*: people
Threat: Tampering *Target*: people
Threat: Repudiation, *Target*: people
Threat: DoS, *Target*: people

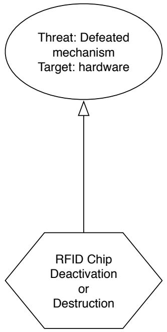

Target Influence via Perception of Authority
Threat: Spoofing *Target*: people
Threat: Defeat protection mechanism *Target*: people
Threat: Elevation of Privilege *Target*: people
Threat: Information Disclosure *Target*: people
Threat: Tampering *Target*: people
Threat: Repudiation, *Target*: people
Threat: DoS, *Target*: people

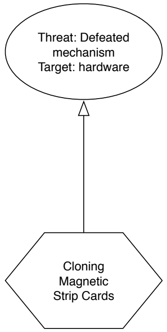

Target Influence via Perception of Liking
Threat: Spoofing *Target*: people
Threat: Defeat protection mechanism *Target*: people
Threat: Elevation of Privilege *Target*: people
Threat: Information Disclosure *Target*: people
Threat: Tampering *Target*: people
Threat: Repudiation, *Target*: people
Threat: DoS, *Target*: people

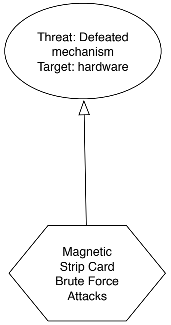

Target Influence via Perception of Consensus or Social Proof
Threat: Spoofing *Target*: people
Threat: Defeat protection mechanism *Target*: people
Threat: Elevation of Privilege *Target*: people
Threat: Information Disclosure *Target*: people
Threat: Tampering *Target*: people
Threat: Repudiation, *Target*: people
Threat: DoS, *Target*: people

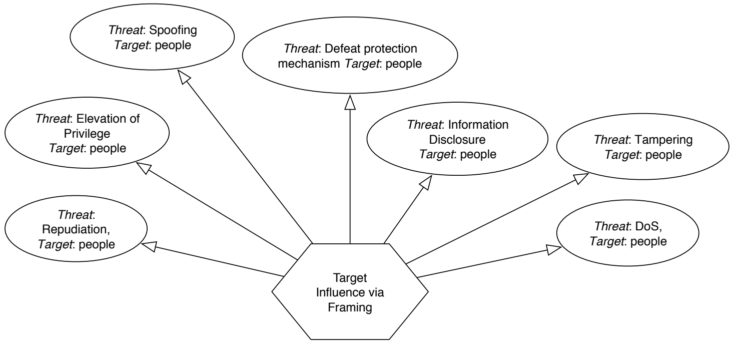

Threat: Spoofing Target: people
Threat: Defeat protection mechanism Target: people
Threat: Elevation of Privilege Target: people
Threat: Information Disclosure Target: people
Threat: Tampering Target: people
Threat: Repudiation, Target: people
Threat: DoS, Target: people
Target Influence via Interview and Interrogation

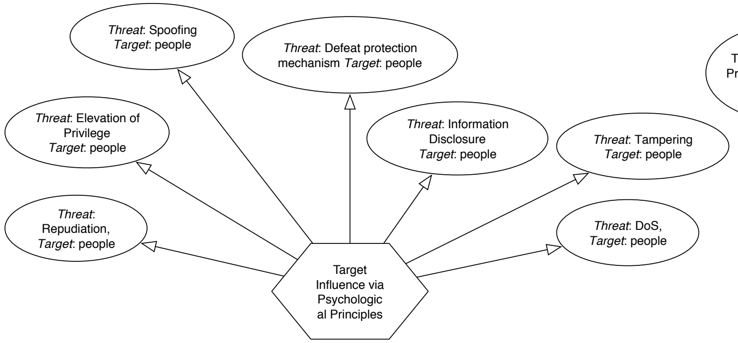

Threat: Spoofing Target: people
Threat: Defeat protection mechanism Target: people
Threat: Elevation of Privilege Target: people
Threat: Information Disclosure Target: people
Threat: Tampering Target: people
Threat: Repudiation, Target: people
Threat: DoS, Target: people
Target Influence via Perception of Concession

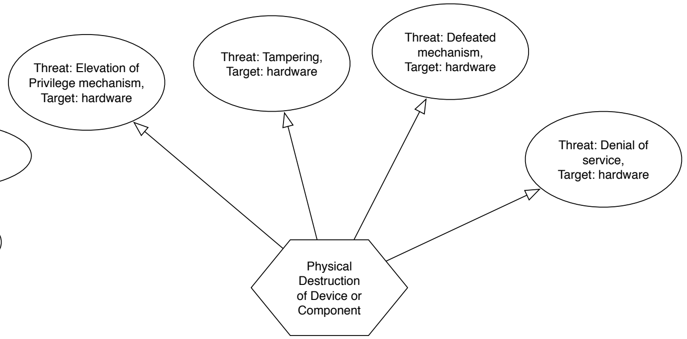

Threat: Defeated mechanism Target: hardware
RFID Chip Deactivation or Destruction

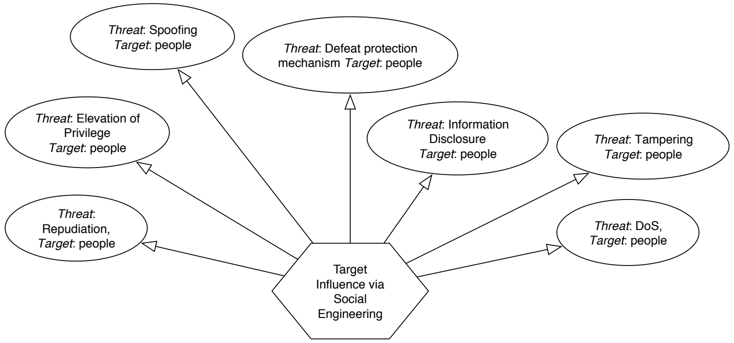

Threat: Defeated mechanism Target: hardware
Cloning Magnetic Strip Cards

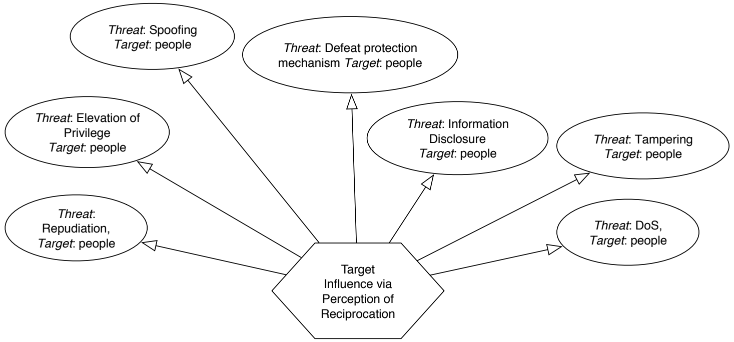

Threat: Defeated mechanism Target: hardware
Magnetic Strip Card Brute Force Attacks

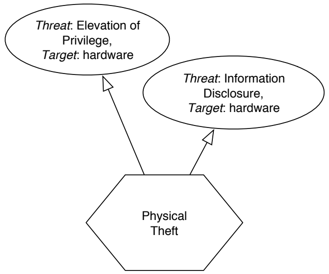

Threat: Spoofing Target: people
Threat: Defeat protection mechanism Target: people
Threat: Elevation of Privilege Target: people
Threat: Information Disclosure Target: people
Threat: Tampering Target: people
Threat: Repudiation, Target: people
Threat: DoS, Target: people
Target Influence via Framing

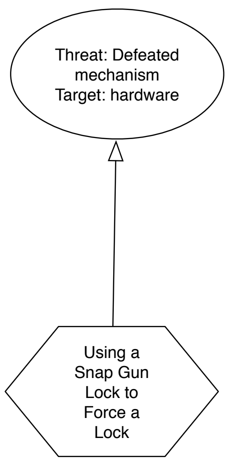

Threat: Spoofing Target: people
Threat: Defeat protection mechanism Target: people
Threat: Elevation of Privilege Target: people
Threat: Information Disclosure Target: people
Threat: Tampering Target: people
Threat: Repudiation, Target: people
Threat: DoS, Target: people
Target Influence via Psychological Principles

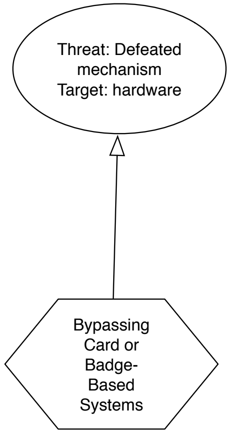

Threat: Elevation of Privilege mechanism, Target: hardware
Threat: Tampering, Target: hardware
Threat: Defeated mechanism, Target: hardware
Threat: Denial of service, Target: hardware
Physical Destruction of Device or Component

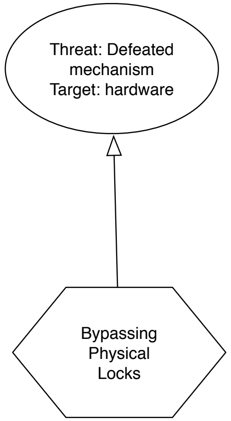

Threat: Spoofing Target: people
Threat: Defeat protection mechanism Target: people
Threat: Elevation of Privilege Target: people
Threat: Information Disclosure Target: people
Threat: Tampering Target: people
Threat: Repudiation, Target: people
Threat: DoS, Target: people
Target Influence via Social Engineering

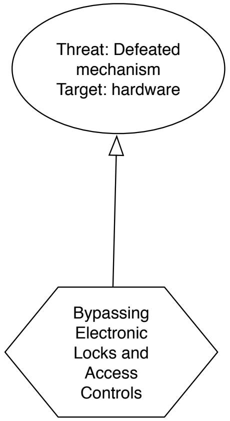

Threat: Spoofing Target: people
Threat: Defeat protection mechanism Target: people
Threat: Elevation of Privilege Target: people
Threat: Information Disclosure Target: people
Threat: Tampering Target: people
Threat: Repudiation, Target: people
Threat: DoS, Target: people
Target Influence via Perception of Reciprocation

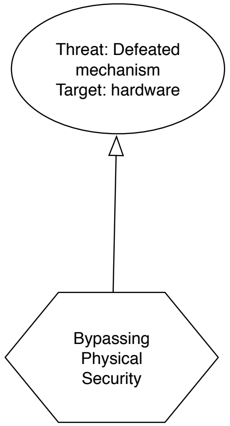

Threat: Elevation of Privilege, Target: hardware
Threat: Information Disclosure, Target: hardware
Physical Theft

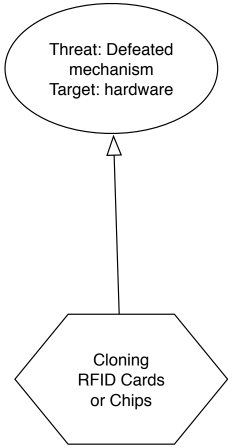

Threat: Defeated mechanism
Target: hardware
Using a Snap Gun Lock to Force a Lock

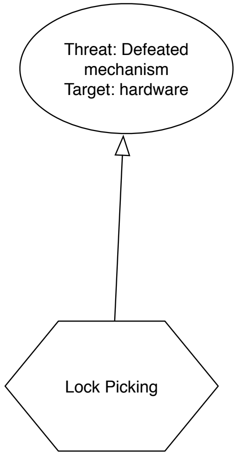

Threat: Defeated mechanism
Target: hardware
Bypassing Card or Badge-Based Systems

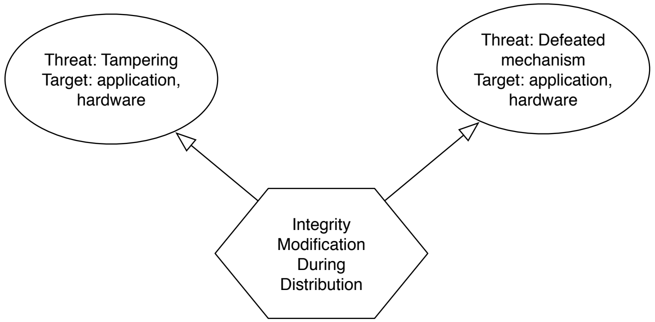


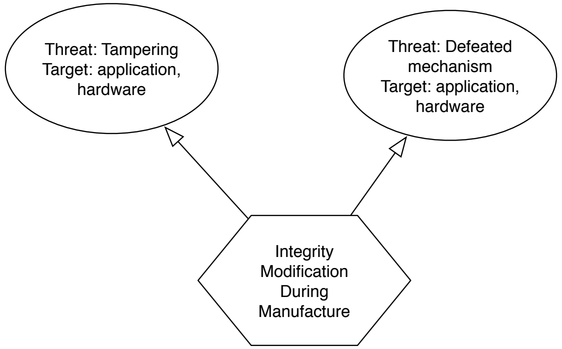

Threat: Defeated mechanism
Target: hardware
Bypassing Electronic Locks and Access Controls



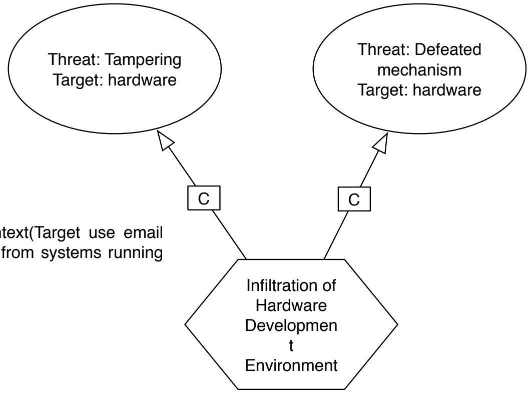




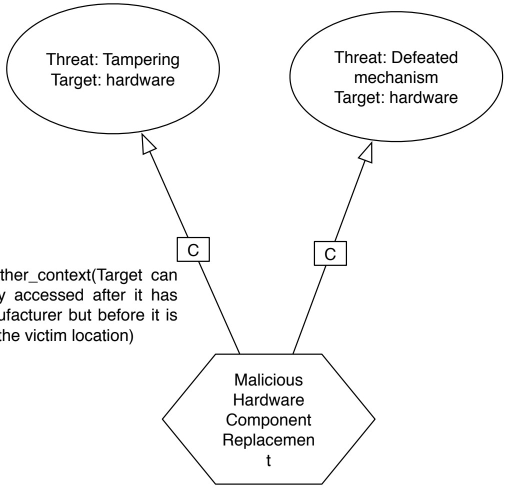

Threat: Tampering
Target: application, hardware
Threat: Defeated mechanism
Target: application, hardware
Integrity Modification During Distribution


Threat: Tampering
Target: application, hardware
Threat: Defeated mechanism
Target: application, hardware
Integrity Modification During Manufacture

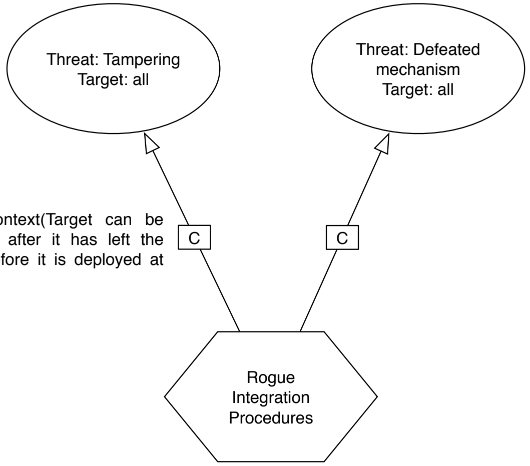
Context = other_context(Target use email or removable media from systems running the IDE)


Threat: Tampering
Target: hardware
Threat: Defeated mechanism
Target: hardware
C
C
Infiltration of Hardware Development Environment

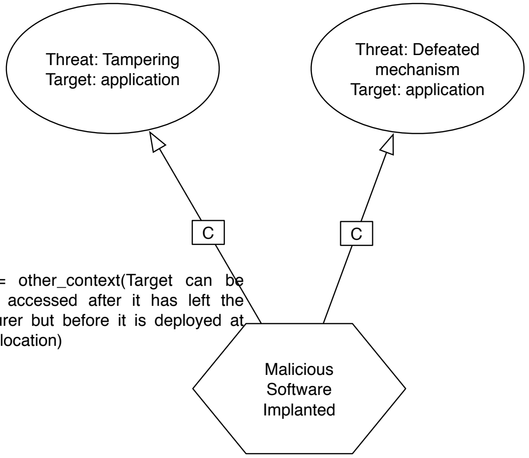
Context = other_context(Target can be physically accessed after it has left the manufacturer but before it is deployed at the victim location)

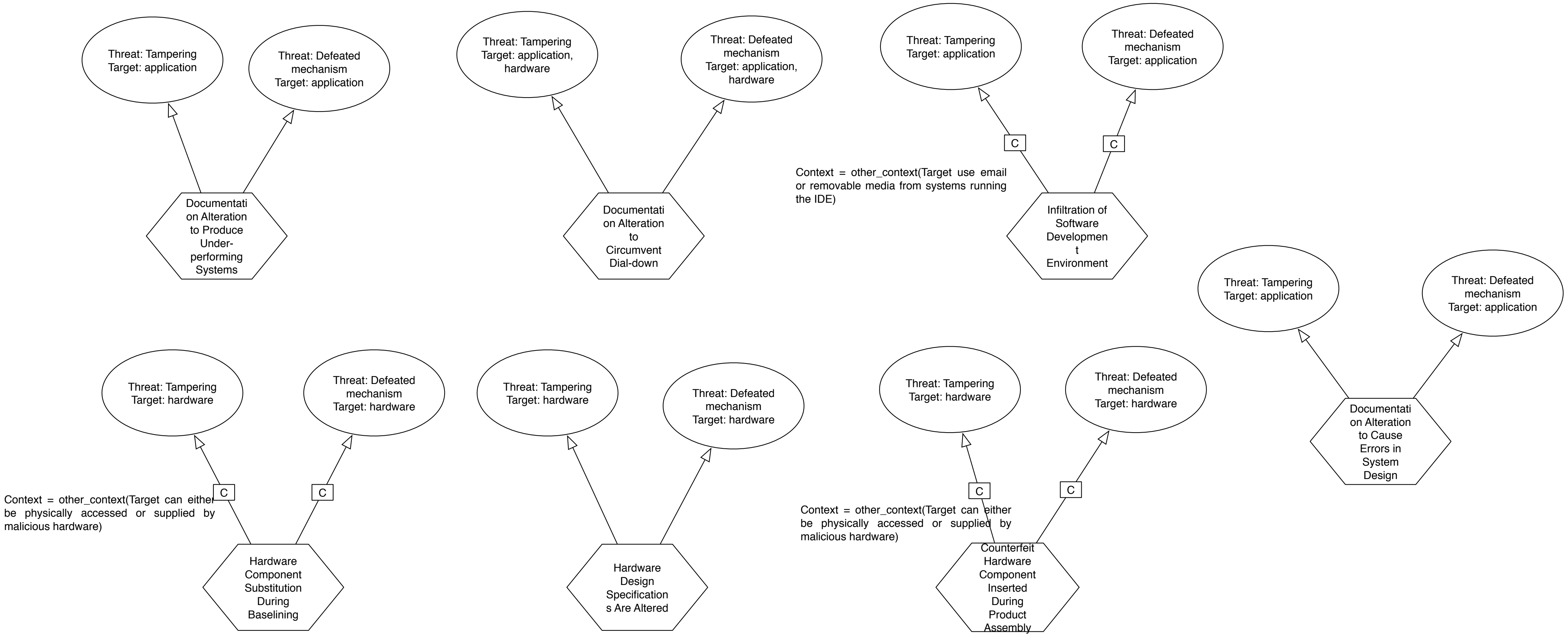

Threat: Tampering
Target: hardware
Threat: Defeated mechanism
Target: hardware
C
C
Malicious Hardware Component Replacement

Context = other_context(Target can be physically accessed after it has left the manufacturer but before it is deployed at the victim location)



Context = other_context(Target can be physically accessed after it has left the manufacturer but before it is deployed at the victim location)




Threat: Tampering
Target: application
Threat: Defeated mechanism
Target: application
Documentation Alteration to Produce Under-performing Systems
Threat: Tampering
Target: application, hardware
Threat: Defeated mechanism
Target: application, hardware
Documentation Alteration to Circumvent Dial-down
Threat: Tampering
Target: application
Threat: Defeated mechanism
Target: application
C
C
Context = other_context(Target use email or removable media from systems running the IDE)
Infiltration of Software Development Environment
Threat: Tampering
Target: application
Threat: Defeated mechanism
Target: application
Documentation Alteration to Cause Errors in System Design
Threat: Tampering
Target: hardware
Threat: Defeated mechanism
Target: hardware
C
C
Context = other_context(Target can either be physically accessed or supplied by malicious hardware)
Hardware Component Substitution During Baselining
Threat: Tampering
Target: hardware
Threat: Defeated mechanism
Target: hardware
Hardware Design Specifications Are Altered
Threat: Tampering
Target: hardware
Threat: Defeated mechanism
Target: hardware
C
C
Context = other_context(Target can either be physically accessed or supplied by malicious hardware)
Counterfeit Hardware Component Inserted During Product Assembly

Appendix D

Context Inference Rules for Attack Patterns

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
% Attack Pattern Context %
%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%
% CAPEC-20 Encryption Brute Forcing
%%%%%%%%%%
applicable_to(20,AG) :- relevant_to(20,AG), has_target(AG,TA), protected_by(TA,
encryption).



%%%%%%%%%%
% CAPEC-97 Cryptanalysis
%%%%%%%%%%
applicable_to(97,AG) :- relevant_to(97,AG), has_target(AG,TA), use_technique(TA,
cryptographic_algorithm).



%%%%%%%%%%
% CAPEC-49 Password Brute Forcing
%%%%%%%%%%
applicable_to(49,AG) :- relevant_to(49,AG), has_target(AG,TA), protected_by(TA,
password_based_authentication), use_detailed_technique(TA,
password_throttling_mechanism).
%%%%
% question uncheckable context
%%%%
question(use_detailed_technique, TA, password_throttling_mechanism) :- relevant_to(49,
AG), has_target(AG,TA), not use_detailed_technique(TA, password_throttling_mechanism),
not no_use_detailed_technique(TA, password_throttling_mechanism).
%%% new generated facts can be generated based on the answer to the question
% use_detailed_technique(TA, password_throttling_mechanism) :- question(use_technique,
TA, password_throttling_mechanism), yes
% no_use_detailed_technique(TA, password_throttling_mechanism) :-
question(use_technique, TA, password_throttling_mechanism), no



%%%%%%%%%%
% CAPEC-55 Rainbow Table Password Cracking
%%%%%%%%%%
applicable_to(55,AG) :- relevant_to(55,AG), has_target(AG,TA), protected_by(TA,
password_based_authentication), use_detailed_technique(TA, salt).
%%%%
% question uncheckable context
%%%%
question(use_detailed_technique, TA, salt) :- relevant_to(49, AG), has_target(AG,TA),
not use_detailed_technique(TA, salt), not no_use_detailed_technique(TA, salt).



%%%%%%%%%%
% CAPEC-16 Dictionary-based Password Attack
%%%%%%%%%%
applicable_to(16,AG) :- relevant_to(16,AG), has_target(AG,TA), protected_by(TA,
password_based_authentication), no_use_detailed_technique(TA, sound_password_policy),
no_use_detailed_technique(TA, password_throttling_mechanism).
%%
```

```
% question uncheckable context
%%
question(use_detailed_technique, TA, password_throttling_mechanism) :- relevant_to(16,
AG), has_target(AG,TA), not use_detailed_technique(TA, password_throttling_mechanism),
not no_use_detailed_technique(TA, password_throttling_mechanism).
question(use_detailed_technique, TA, sound_password_policy) :- relevant_to(16, AG),
has_target(AG,TA), not use_detailed_technique(TA, sound_password_policy), not
no_use_detailed_technique(TA, sound_password_policy).



%%%%%%%%%%
% CAPEC-70 Try Common(default) Usernames and Passwords
%%%%%%%%%%
applicable_to(70,AG) :- relevant_to(70,AG), has_target(AG,TA), protected_by(TA,
password_based_authentication).



%%%%%%%%%%
% CAPEC-56 Removing/short-circuiting 'guard logic'
%%%%%%%%%%
applicable_to(56,AG) :- relevant_to(56,AG), has_target(AG,TA), use_paradigm(TA,
client_server).
%%%%
% question uncheckable context
%%%%
question(use_paradigm, TA, client_server) :- relevant_to(56, AG), has_target(AG,TA),
not use_paradigm(TA, client_server), not no_use_paradigm(TA, client_server).



%%%%%%%%%%
% CAPEC-66 SQL Injection
%%%%%%%%%%
applicable_to(66,AG) :- relevant_to(66,AG), has_target(AG,TA), use_technique(TA,
sql_query).



%%%%%%%%%%
% CAPEC-7 Blind SQL Injection
%%%%%%%%%%
applicable_to(7,AG) :- relevant_to(7,AG), has_target(AG,TA), use_technique(TA,
sql_query).



%%%%%%%%%%
% CAPEC-110 SQL Injection through SOAP Parameter Tampering
%%%%%%%%%%
applicable_to(110,AG) :- relevant_to(110,AG), has_target(AG,TA),
use_technique(TA,soap), use_paradigm(TA,soa).
%%%%
% question uncheckable context
%%%%
question(use_paradigm,TA,soa) :- relevant_to(110,AG), has_target(AG,TA), not
use_paradigm(TA,soa), not no_use_paradigm(TA,soa).
```

```
%%%%%%%%%%
% CAPEC-109 SQL Injection through SOAP Parameter Tampering
%%%%%%%%%%
applicable_to(109, AG) :- relevant_to(109, AG), has_target(AG, TA), use_technique(TA,
soap), accept_user_data(TA), use_paradigm(TA, client_server),
use_detailed_technique(TA, data_access_layer_by_orm_tool).
%%%%
% question uncheckable context
%%%%
question(use_paradigm, TA, client_server) :- relevant_to(109, AG), has_target(AG, TA),
not use_paradigm(TA, client_server), not no_use_paradigm(TA, client_server).
question(use_detailed_technique, TA, data_access_layer_by_orm_tool) :- relevant_to(109,
AG), has_target(AG, TA), not use_detailed_technique(TA, data_access_layer_by_orm_tool),
not no_use_detailed_technique(TA, data_access_layer_by_orm_tool).
```

```
%%%%%%%%%%
% CAPEC-108 Command Line Execution through SQL Injection
%%%%%%%%%%
applicable_to(108, AG) :- relevant_to(108, AG), has_target(AG, TA), trust(TA,
data_in_the_database).
%%%%
% question uncheckable context
%%%%
question(trust, TA, data_in_the_database) :- relevant_to(108, AG), has_target(AG, TA),
not trust(TA, data_in_the_database), not no_trust(TA, data_in_the_database).
```

```
%%%%%%%%%%
% CAPEC-69 Target Programs with Elevated Privileges
%%%%%%%%%%
applicable_to(69, AG) :- relevant_to(69, AG), has_target(AG, TA), use_data_from(TA, _),
technical_context(TA, run_with_elevated_OS_privileges), technical_context(TA,
give_away_information_about_itself).
%%%%
% question uncheckable context
%%%%
question(technical_context, TA, run_with_elevated_OS_privileges) :- relevant_to(69,
AG), has_target(AG, TA), not technical_context(TA, run_with_elevated_OS_privileges),
not no_technical_context(TA, run_with_elevated_OS_privileges).
question(technical_context, TA, give_away_information_about_itself) :- relevant_to(69,
AG), has_target(AG, TA), not technical_context(TA, give_away_information_about_itself),
not no_technical_context(TA, give_away_information_about_itself).
```

```
%%%%%%%%%%
% CAPEC-14 Client-side Injection-induced Buffer Overflow
%%%%%%%%%%
applicable_to(14, AG) :- relevant_to(14, AG), has_target(AG, TA), use_paradigm(TA,
client_server).
%%%%
% question uncheckable context
%%%%
question(use_paradigm, TA, client_server) :- relevant_to(14, AG), has_target(AG, TA),
not use_paradigm(TA, client_server), not no_use_paradigm(TA, client_server).
```

```
%%%%%%%%%%
% CAPEC-42 MIME Conversion
%%%%%%%%%%
applicable_to(42, AG) :- relevant_to(42, AG), has_target(AG, TA), technical_context(TA,
use_a_mail_server).
%%%%
% question uncheckable context
%%%%
question(technical_context, TA, use_a_mail_server) :- relevant_to(42, AG),
has_target(AG, TA), not technical_context(TA, use_a_mail_server), not
no_technical_context(TA, use_a_mail_server).



%%%%%%%%%%
% CAPEC-10 Buffer Overflow via Environment Variables
%%%%%%%%%%
applicable_to(10, AG) :- relevant_to(10, AG), has_target(AG, TA),
use_detailed_technique(TA, environment_variables).
%%%%
% question uncheckable context
%%%%
question(use_detailed_technique, TA, environment_variables) :- relevant_to(10, AG),
has_target(AG, TA), not use_detailed_technique(TA, environment_variables), not
no_use_detailed_technique(TA, environment_variables).



%%%%%%%%%%
% CAPEC-24 Filter Failure through Buffer Overflow
%%%%%%%%%%
applicable_to(24, AG) :- relevant_to(24, AG), has_target(AG, TA), use_language(TA,
c_or_c_plus_plus).
%%%%
% question uncheckable context
%%%%
question(use_language, TA, c_or_c_plus_plus) :- relevant_to(24, AG), has_target(AG,
TA), not use_language(TA, c_or_c_plus_plus), not no_use_language(TA, c_or_c_plus_plus).



%%%%%%%%%%
% CAPEC-67 String Format Overflow in syslog()
%%%%%%%%%%
applicable_to(67, AG) :- relevant_to(67, AG), has_target(AG, TA), accept_user_data(TA),
use_language(TA, c_or_c_plus_plus).
%%%%
% question uncheckable context
%%%%
question(use_language, TA, c_or_c_plus_plus) :- relevant_to(67, AG), has_target(AG,
TA), not use_language(TA, c_or_c_plus_plus), not no_use_language(TA, c_or_c_plus_plus).



%%%%%%%%%%
% CAPEC-9 String Format Overflow in syslog()
%%%%%%%%%%
applicable_to(9, AG) :- relevant_to(9, AG), has_target(AG, TA), technical_context(TA,
expose_a_command_line_utility_to_users).
%%%%
```

```
% question uncheckable context
%%%%
question(technical_context, TA, expose_a_command_line_utility_to_users) :-
relevant_to(9, AG), has_target(AG, TA), not technical_context(TA,
expose_a_command_line_utility_to_users), not no_technical_context(TA,
expose_a_command_line_utility_to_users).



%%%%%%%%%%
% CAPEC-8 Buffer Overflow in an API Call
%%%%%%%%%%
applicable_to(8, AG) :- relevant_to(8, AG), has_target(AG, TA), technical_context(TA,
expose_an_api_to_users).
%%%%
% question uncheckable context
%%%%
question(technical_context, TA, expose_an_api_to_users) :- relevant_to(8, AG),
has_target(AG, TA), not technical_context(TA, expose_an_api_to_users), not
no_technical_context(TA, expose_an_api_to_users).



%%%%%%%%%%
% CAPEC-45 Buffer Overflow via Symbolic Links
%%%%%%%%%%
applicable_to(45, AG) :- relevant_to(45, AG), has_target(AG, TA), use_language(TA,
c_or_c_plus_plus).
%%%%
% question uncheckable context
%%%%
question(use_language, TA, c_or_c_plus_plus) :- relevant_to(45, AG), has_target(AG,
TA), not use_language(TA, c_or_c_plus_plus), not no_use_language(TA, c_or_c_plus_plus).



%%%%%%%%%%
% CAPEC-44 Overflow Binary Resource File
%%%%%%%%%%
applicable_to(44, AG) :- relevant_to(44, AG), has_target(AG, TA), use_language(TA,
c_or_c_plus_plus), technical_context(TA, processes_binary_resource_files).
%%%%
% question uncheckable context
%%%%
question(use_language, TA, c_or_c_plus_plus) :- relevant_to(44, AG), has_target(AG,
TA), not use_language(TA, c_or_c_plus_plus), not no_use_language(TA, c_or_c_plus_plus).
question(technical_context, TA, processes_binary_resource_files) :- relevant_to(44,
AG), has_target(AG, TA), not technical_context(TA, processes_binary_resource_files),
not no_technical_context(TA, processes_binary_resource_files).



%%%%%%%%%%
% CAPEC-46 Overflow Variables and Tags
%%%%%%%%%%
applicable_to(46, AG) :- relevant_to(46, AG), has_target(AG, TA),
accept_user_input(TA), use_language(TA, c_or_c_plus_plus).
%%%%
% question uncheckable context
%%%%
question(use_language, TA, c_or_c_plus_plus) :- relevant_to(46, AG), has_target(AG,
TA), not use_language(TA, c_or_c_plus_plus), not no_use_language(TA, c_or_c_plus_plus).



%%%%%%%%%%
% CAPEC-47 Buffer Overflow via Parameter Expansion
%%%%%%%%%%
applicable_to(47, AG) :- relevant_to(47, AG), has_target(AG, TA),
accept_user_input(TA), use_language(TA, c_or_c_plus_plus).
%%%%
% question uncheckable context
%%%%
question(use_language, TA, c_or_c_plus_plus) :- relevant_to(47, AG), has_target(AG,
TA), not use_language(TA, c_or_c_plus_plus), not no_use_language(TA, c_or_c_plus_plus).



%%%%%%%%%%
% CAPEC-100 Overflow Buffers
%%%%%%%%%%
applicable_to(100, AG) :- relevant_to(100, AG), has_target(AG, TA),
technical_context(TA, perform_buffer_operation), use_language(TA,
c_or_c_plus_plus_or_ajax_or_perl_or_php_or_vb_or_ruby).
%%%%
% question uncheckable context
%%%%
question(use_language, TA, c_or_c_plus_plus_or_ajax_or_perl_or_php_or_vb_or_ruby) :-
relevant_to(100, AG), has_target(AG, TA), not use_language(TA,
c_or_c_plus_plus_or_ajax_or_perl_or_php_or_vb_or_ruby), not no_use_language(TA,
c_or_c_plus_plus_or_ajax_or_perl_or_php_or_vb_or_ruby).
%
question(technical_context, TA, perform_buffer_operation) :- relevant_to(100, AG),
has_target(AG, TA), not technical_context(TA, perform_buffer_operation), not
no_technical_context(TA, perform_buffer_operation).



%%%%%%%%%%
% CAPEC-22 Exploiting Trust in Client (aka Make the Client Invisible)
%%%%%%%%%%
applicable_to(22, AG) :- relevant_to(22, AG), has_target(AG, TA),
accept_user_input(TA), use_paradigm(TA, client_server_or_ntier).
%%%%
% question uncheckable context
%%%%
question(use_paradigm, TA, client_server_or_ntier) :- relevant_to(22, AG),
has_target(AG, TA), not use_paradigm(TA, client_server_or_ntier), not
no_use_paradigm(TA, client_server_or_ntier).



%%%%%%%%%%
% CAPEC-77 Manipulating User-Controlled Variables
%%%%%%%%%%
applicable_to(77, AG) :- relevant_to(77, AG), has_target(AG, TA),
accept_user_input(TA), technical_context(TA, a_variable_is_exposed_to_client).
%%%%
% question uncheckable context
%%%%
question(technical_context, TA, a_variable_is_exposed_to_client) :- relevant_to(77,
AG), has_target(AG, TA), not technical_context(TA, a_variable_is_exposed_to_client),
not no_technical_context(TA, a_variable_is_exposed_to_client).



%%%%%%%%%%
% CAPEC-39 Manipulating Opaque Client-based Data Tokens
%%%%%%%%%%
applicable_to(39, AG) :- relevant_to(39, AG), has_target(AG, TA),
accept_user_input(TA).



%%%%%%%%%%
% CAPEC-94 Man in the Middle Attack
%%%%%%%%%%
applicable_to(94, AG) :- relevant_to(94, AG), has_target(AG, TA), communicate(TA, _),
not protected_by(TA, encryption).



%%%%%%%%%%
% CAPEC-13 Subverting Environment Variable Values
%%%%%%%%%%
applicable_to(13, AG) :- relevant_to(13, AG), has_target(AG, TA), accept_user_data(TA),
technical_context(TA, an_environment_variable_is_accessible_to_the_user).
%%%%
% question uncheckable context
%%%%
question(technical_context, TA, an_environment_variable_is_accessible_to_the_user) :-
relevant_to(13, AG), has_target(AG, TA), not technical_context(TA,
an_environment_variable_is_accessible_to_the_user), not no_technical_context(TA,
an_environment_variable_is_accessible_to_the_user).




%%%%%%%%%%
% CAPEC-31 Accessing/Intercepting/Modifying HTTP Cookies
%%%%%%%%%%
applicable_to(31, AG) :- relevant_to(31, AG), has_target(AG, TA), use_paradigm(TA,
client_server_or_ntier), technical_context(TA, a_http_daemon_that_relies_on_cookies).
%%%%
% question uncheckable context
%%%%
question(use_paradigm, TA, client_server_or_ntier) :- relevant_to(31, AG),
has_target(AG, TA), not use_paradigm(TA, client_server_or_ntier), not
no_use_paradigm(TA, client_server_or_ntier).
%
question(technical_context, TA, a_http_daemon_that_relies_on_cookies) :-
relevant_to(31, AG), has_target(AG, TA), not technical_context(TA,
a_http_daemon_that_relies_on_cookies), not no_technical_context(TA,
a_http_daemon_that_relies_on_cookies).



%%%%%%%%%%
% CAPEC-57 Utilizing REST's Trust in the System Resource to Register Man in the Middle
%%%%%%%%%%
applicable_to(57, AG) :- relevant_to(57, AG), has_target(AG, TA), use_paradigm(TA,
soa).
%%%%
```

```
% question uncheckable context
%%%%
question(use_paradigm, TA, soa) :- relevant_to(57, AG), has_target(AG, TA), not
use_paradigm(TA, soa), not no_use_paradigm(TA, soa).



%%%%%%%%%%
% CAPEC-219 XML Routing Detour Attacks
%%%%%%%%%%
applicable_to(219, AG) :- relevant_to(219, AG), has_target(AG, TA), use_paradigm(TA,
client_server), technical_context(TA, have_multiple_stages_processing_of_XML_content).
%%%%
% question uncheckable context
%%%%
question(use_paradigm, TA, client_server) :- relevant_to(219, AG), has_target(AG, TA),
not use_paradigm(TA, client_server), not no_use_paradigm(TA, client_server).
%
question(technical_context, TA, have_multiple_stages_processing_of_XML_content) :-
relevant_to(219, AG), has_target(AG, TA), not technical_context(TA,
have_multiple_stages_processing_of_XML_content), not no_technical_context(TA,
have_multiple_stages_processing_of_XML_content).



%%%%%%%%%%
% CAPEC-76 Manipulating Input to File System Calls
%%%%%%%%%%
applicable_to(76, AG) :- relevant_to(76, AG), has_target(AG, TA), accept_user_data(TA),
technical_context(TA, user_controlled_variables_is_applied_directly_to_the_filesystem).
%%%%
% question uncheckable context
%%%%
question(technical_context, TA,
user_controlled_variables_is_applied_directly_to_the_filesystem) :- relevant_to(76,
AG), has_target(AG, TA), not technical_context(TA,
user_controlled_variables_is_applied_directly_to_the_filesystem), not
no_technical_context(TA,
user_controlled_variables_is_applied_directly_to_the_filesystem).



%%%%%%%%%%
% CAPEC-122 Privilege Abuse
%%%%%%%%%%
applicable_to(122, AG) :- relevant_to(122, AG), has_target(AG, TA), protected_by(TA,
access_control).



%%%%%%%%%%
% CAPEC-180 Exploiting Incorrectly Configured Access Control Security Levels
%%%%%%%%%%
applicable_to(180, AG) :- relevant_to(180, AG), has_target(AG, TA), protected_by(TA,
access_control).



%%%%%%%%%%
% CAPEC-1 Accessing Functionality Not Properly Constrained by ACLs
%%%%%%%%%%
```

```
applicable_to(1, AG) :- relevant_to(1, AG), has_target(AG, TA), use_technique(TA, acl).



%%%%%%%%%%
% CAPEC-58 Restful Privilege Elevation
%%%%%%%%%%
applicable_to(58, AG) :- relevant_to(58, AG), has_target(AG, TA), use_paradigm(TA,
soa).
%%%%
% question uncheckable context
%%%%
question(use_paradigm, TA, soa) :- relevant_to(58, AG), has_target(AG, TA), not
use_paradigm(TA, soa), not no_use_paradigm(TA, soa).



%%%%%%%%%%
% CAPEC-17 Accessing, Modifying or Executing Executable Files
%%%%%%%%%%
applicable_to(17, AG) :- relevant_to(17, AG), has_target(AG, TA), technical_context(TA,
user_can_directly_access_executable_files_or_upload_files_to_execute).
%%%%
% question uncheckable context
%%%%
question(technical_context, TA,
user_can_directly_access_executable_files_or_upload_files_to_execute) :-
relevant_to(17, AG), has_target(AG, TA), not technical_context(TA,
user_can_directly_access_executable_files_or_upload_files_to_execute), not
no_technical_context(TA,
user_can_directly_access_executable_files_or_upload_files_to_execute).



%%%%%%%%%%
% CAPEC-115 Authentication Bypass
%%%%%%%%%%
applicable_to(115, AG) :- relevant_to(115, AG), has_target(AG, TA), protected_by(TA,
authentication).



%%%%%%%%%%
% CAPEC-237 Calling Signed Code From Another Language Within A Sandbox Allow This
%%%%%%%%%%
applicable_to(237, AG) :- relevant_to(237, AG), has_target(AG, TA), use_language(TA,
java_or_asp_net_or_c_sharp_or_jsp), technical_context(TA,
deployed_code_signed_by_its_authoring_vendor).
%%%%
% question uncheckable context
%%%%
question(use_language, TA, java_or_asp_net_or_c_sharp_or_jsp) :- relevant_to(237, AG),
has_target(AG, TA), not use_language(TA, java_or_asp_net_or_c_sharp_or_jsp), not
no_use_language(TA, java_or_asp_net_or_c_sharp_or_jsp).
%
question(technical_context, TA, deployed_code_signed_by_its_authoring_vendor) :-
relevant_to(237, AG), has_target(AG, TA), not technical_context(TA,
deployed_code_signed_by_its_authoring_vendor), not no_technical_context(TA,
deployed_code_signed_by_its_authoring_vendor).
```

```
%%%%%%%%%%
% CAPEC-114 Authentication Abuse
%%%%%%%%%%
applicable_to(114, AG) :- relevant_to(114, AG), has_target(AG, TA), protected_by(TA,
authentication).



%%%%%%%%%%
% CAPEC-90 Reflection Attack in Authentication Protocol
%%%%%%%%%%
applicable_to(90, AG) :- relevant_to(90, AG), has_target(AG, TA), use_paradigm(TA,
client_server_or_soa).
%%%%
% question uncheckable context
%%%%
question(use_paradigm, TA, client_server_or_soa) :- relevant_to(90, AG), has_target(AG,
TA), not use_paradigm(TA, client_server_or_soa), not no_use_paradigm(TA,
client_server_or_soa).
%



%%%%%%%%%%
% CAPEC-21 Exploitation of Session Variables, Resource IDs and other Trusted
Credentials
%%%%%%%%%%
applicable_to(21, AG) :- relevant_to(21, AG), has_target(AG, TA), use_paradigm(TA,
client_server_or_soa_or_ntier).
%%%%
% question uncheckable context
%%%%
question(use_paradigm, TA, client_server_or_soa_or_ntier) :- relevant_to(21, AG),
has_target(AG, TA), not use_paradigm(TA, client_server_or_soa_or_ntier), not
no_use_paradigm(TA, client_server_or_soa_or_ntier).



%%%%%%%%%%
% CAPEC-62 Cross Site Request Forgery (aka Session Riding)
%%%%%%%%%%
applicable_to(62, AG) :- relevant_to(62, AG), has_target(AG, TA), use_paradigm(TA,
client_server), use_framework(TA, j2ee_or_dot_net).
%%%%
% question uncheckable context
%%%%
question(use_paradigm, TA, client_server) :- relevant_to(62, AG), has_target(AG, TA),
not use_paradigm(TA, client_server), not no_use_paradigm(TA, client_server).
%
question(use_framework, TA, j2ee_or_dot_net) :- relevant_to(62, AG), has_target(AG,
TA), not use_framework(TA, j2ee_or_dot_net), not no_use_framework(TA, j2ee_or_dot_net).



%%%%%%%%%%
% CAPEC-102 Session Sidejacking
%%%%%%%%%%
applicable_to(102, AG) :- relevant_to(102, AG), has_target(AG, TA), not
protected_by(TA, secure_communication_mechanism), use_paradigm(TA, client_server),
use_language(TA, ajax), technical_context(TA,
has_an_active_session_with_a_target_system).
%%%%
% question uncheckable context
%%%%
question(use_paradigm, TA, client_server) :- relevant_to(102, AG), has_target(AG, TA),
not use_paradigm(TA, client_server), not no_use_paradigm(TA, client_server).
%
question(technical_context, TA, has_an_active_session_with_a_target_system) :-
relevant_to(102, AG), has_target(AG, TA), not technical_context(TA,
has_an_active_session_with_a_target_system), not no_technical_context(TA,
has_an_active_session_with_a_target_system).



%%%%%%%%%%
% CAPEC-61 Session Fixation
%%%%%%%%%%
applicable_to(61, AG) :- relevant_to(61, AG), has_target(AG, TA), use_paradigm(TA,
client_server), use_framework(TA, j2ee_or_dot_net), technical_context(TA,
use_session_identifiers_that_remain_unchanged_when_the_privilege_levels_change),
technical_context(TA,
use_permissive_session_management_mechanism_that_accepts_random_user_generated_session_
identifiers).
%%%%
% question uncheckable context
%%%%
question(use_paradigm, TA, client_server) :- relevant_to(61, AG), has_target(AG, TA),
not use_paradigm(TA, client_server), not no_use_paradigm(TA, client_server).
%
question(use_framework, TA, j2ee_or_dot_net) :- relevant_to(61, AG), has_target(AG,
TA), not use_framework(TA, j2ee_or_dot_net), not no_use_framework(TA, j2ee_or_dot_net).
%
question(technical_context, TA,
use_session_identifiers_that_remain_unchanged_when_the_privilege_levels_change) :-
relevant_to(61, AG), has_target(AG, TA), not technical_context(TA,
use_session_identifiers_that_remain_unchanged_when_the_privilege_levels_change), not
no_technical_context(TA,
use_session_identifiers_that_remain_unchanged_when_the_privilege_levels_change).
%
question(technical_context, TA,
use_permissive_session_management_mechanism_that_accepts_random_user_generated_session_
identifiers) :- relevant_to(61, AG), has_target(AG, TA), not technical_context(TA,
use_permissive_session_management_mechanism_that_accepts_random_user_generated_session_
identifiers), not no_technical_context(TA,
use_permissive_session_management_mechanism_that_accepts_random_user_generated_session_
identifiers).



%%%%%%%%%%
% CAPEC-60 Reusing Session IDs (aka Session Replay)
%%%%%%%%%%
applicable_to(60, AG) :- relevant_to(60, AG), has_target(AG, TA), use_paradigm(TA,
client_server), use_framework(TA, j2ee_or_dot_net), technical_context(TA,
use_session_id_to_track_users), technical_context(TA,
use_session_id_to_control_access_to_resources).
%%%%
% question uncheckable context
%%%%
question(use_paradigm, TA, client_server) :- relevant_to(60, AG), has_target(AG, TA),
not use_paradigm(TA, client_server), not no_use_paradigm(TA, client_server).
```

```
%
question(use_framework, TA, j2ee_or_dot_net) :- relevant_to(60, AG), has_target(AG,
TA), not use_framework(TA, j2ee_or_dot_net), not no_use_framework(TA, j2ee_or_dot_net).
%
question(technical_context, TA, use_session_id_to_track_users) :- relevant_to(60, AG),
has_target(AG, TA), not technical_context(TA, use_session_id_to_track_users), not
no_technical_context(TA, use_session_id_to_track_users).
%
question(technical_context, TA, use_session_id_to_control_access_to_resources) :-
relevant_to(60, AG), has_target(AG, TA), not technical_context(TA,
use_session_id_to_control_access_to_resources), not no_technical_context(TA,
use_session_id_to_control_access_to_resources).



%%%%%%%%%%
% CAPEC-196 Session Credential Falsification through Forging
%%%%%%%%%%
applicable_to(196, AG) :- relevant_to(196, AG), has_target(AG, TA), use_paradigm(TA,
client_server_or_soa), technical_context(TA,
use_session_credentials_to_identify_legitimate_users).
%%%%
% question uncheckable context
%%%%
question(use_paradigm, TA, client_server_or_soa) :- relevant_to(196, AG),
has_target(AG, TA), not use_paradigm(TA, client_server_or_soa), not no_use_paradigm(TA,
client_server_or_soa).
%
question(technical_context, TA, use_session_credentials_to_identify_legitimate_users)
:- relevant_to(196, AG), has_target(AG, TA), not technical_context(TA,
use_session_credentials_to_identify_legitimate_users), not no_technical_context(TA,
use_session_credentials_to_identify_legitimate_users).



%%%%%%%%%%
% CAPEC-59 Session Credential Falsification through Prediction
%%%%%%%%%%
applicable_to(59, AG) :- relevant_to(59, AG), has_target(AG, TA), use_paradigm(TA,
client_server), use_framework(TA, j2ee_or_dot_net), technical_context(TA,
use_session_id_to_track_users), technical_context(TA,
use_session_id_to_control_access_to_resources), technical_context(TA,
use_session_ids_that_are_predictable).
%%%%
% question uncheckable context
%%%%
question(use_paradigm, TA, client_server) :- relevant_to(59, AG), has_target(AG, TA),
not use_paradigm(TA, client_server), not no_use_paradigm(TA, client_server).
%
question(use_framework, TA, j2ee_or_dot_net) :- relevant_to(59, AG), has_target(AG,
TA), not use_framework(TA, j2ee_or_dot_net), not no_use_framework(TA, j2ee_or_dot_net).
%
question(technical_context, TA, use_session_id_to_track_users) :- relevant_to(59, AG),
has_target(AG, TA), not technical_context(TA, use_session_id_to_track_users), not
no_technical_context(TA, use_session_id_to_track_users).
%
question(technical_context, TA, use_session_id_to_control_access_to_resources) :-
relevant_to(59, AG), has_target(AG, TA), not technical_context(TA,
use_session_id_to_control_access_to_resources), not no_technical_context(TA,
use_session_id_to_control_access_to_resources).
%
```

```
question(technical_context, TA, use_session_ids_that_are_predictable) :-
relevant_to(59, AG), has_target(AG, TA), not technical_context(TA,
use_session_ids_that_are_predictable), not no_technical_context(TA,
use_session_ids_that_are_predictable).



%%%%%%%%%%
% CAPEC-522 Malicious Hardware Component Replacement
%%%%%%%%%%
applicable_to(522, AG) :- relevant_to(522, AG), has_target(AG, TA), other_context(TA,
can_be_physically_accessed_after_it_has_left_the_manufacturer_but_before_it_is_deployed
_at_the_victim_location).
%%%%
% question uncheckable context
%%%%
question(other_context, TA,
can_be_physically_accessed_after_it_has_left_the_manufacturer_but_before_it_is_deployed
_at_the_victim_location) :- relevant_to(522, AG), has_target(AG, TA), not
other_context(TA,
can_be_physically_accessed_after_it_has_left_the_manufacturer_but_before_it_is_deployed
_at_the_victim_location), not no_other_context(TA,
can_be_physically_accessed_after_it_has_left_the_manufacturer_but_before_it_is_deployed
_at_the_victim_location).



%%%%%%%%%%
% CAPEC-524 Rogue Integration Procedures
%%%%%%%%%%
applicable_to(524, AG) :- relevant_to(524, AG), has_target(AG, TA), other_context(TA,
can_be_physically_accessed_after_it_has_left_the_manufacturer_but_before_it_is_deployed
_at_the_victim_location).
%%%%
% question uncheckable context
%%%%
question(other_context, TA,
can_be_physically_accessed_after_it_has_left_the_manufacturer_but_before_it_is_deployed
_at_the_victim_location) :- relevant_to(524, AG), has_target(AG, TA), not
other_context(TA,
can_be_physically_accessed_after_it_has_left_the_manufacturer_but_before_it_is_deployed
_at_the_victim_location), not no_other_context(TA,
can_be_physically_accessed_after_it_has_left_the_manufacturer_but_before_it_is_deployed
_at_the_victim_location).



%%%%%%%%%%
% CAPEC-523 Malicious Software Implanted
%%%%%%%%%%
applicable_to(523, AG) :- relevant_to(523, AG), has_target(AG, TA), other_context(TA,
can_be_physically_accessed_after_it_has_left_the_manufacturer_but_before_it_is_deployed
_at_the_victim_location).
%%%%
% question uncheckable context
%%%%
question(other_context, TA,
can_be_physically_accessed_after_it_has_left_the_manufacturer_but_before_it_is_deployed
_at_the_victim_location) :- relevant_to(523, AG), has_target(AG, TA), not
other_context(TA,
can_be_physically_accessed_after_it_has_left_the_manufacturer_but_before_it_is_deployed
_at_the_victim_location), not no_other_context(TA,
can_be_physically_accessed_after_it_has_left_the_manufacturer_but_before_it_is_deployed
_at_the_victim_location).



%%%%%%%%%%
% CAPEC-537 Infiltration of Hardware Development Environment
%%%%%%%%%%
applicable_to(537, AG) :- relevant_to(537, AG), has_target(AG, TA), other_context(TA,
use_email_or_removable_media_from_systems_running_the_ide).
%%%%
% question uncheckable context
%%%%
question(other_context, TA, use_email_or_removable_media_from_systems_running_the_ide)
:- relevant_to(537, AG), has_target(AG, TA), not other_context(TA,
use_email_or_removable_media_from_systems_running_the_ide), not no_other_context(TA,
use_email_or_removable_media_from_systems_running_the_ide).



%%%%%%%%%%
% CAPEC-511 Infiltration of Hardware Development Environment
%%%%%%%%%%
applicable_to(511, AG) :- relevant_to(511, AG), has_target(AG, TA), other_context(TA,
use_email_or_removable_media_from_systems_running_the_ide).
%%%%
% question uncheckable context
%%%%
question(other_context, TA, use_email_or_removable_media_from_systems_running_the_ide)
:- relevant_to(511, AG), has_target(AG, TA), not other_context(TA,
use_email_or_removable_media_from_systems_running_the_ide), not no_other_context(TA,
use_email_or_removable_media_from_systems_running_the_ide).



%%%%%%%%%%
% CAPEC-520 Counterfeit Hardware Component Inserted During Product Assembly
%%%%%%%%%%
applicable_to(520, AG) :- relevant_to(520, AG), has_target(AG, TA), other_context(TA,
can_either_be_physically_accessed_or_supplied_by_malicious_hardware).
%%%%
% question uncheckable context
%%%%
question(other_context, TA,
can_either_be_physically_accessed_or_supplied_by_malicious_hardware) :-
relevant_to(520, AG), has_target(AG, TA), not other_context(TA,
can_either_be_physically_accessed_or_supplied_by_malicious_hardware), not
no_other_context(TA,
can_either_be_physically_accessed_or_supplied_by_malicious_hardware).



%%%%%%%%%%
% CAPEC-516 Hardware Component Substitution During Baselining
%%%%%%%%%%
applicable_to(516, AG) :- relevant_to(516, AG), has_target(AG, TA), other_context(TA,
can_either_be_physically_accessed_or_supplied_by_malicious_hardware).
%%%%
% question uncheckable context
```

```
%%%%
question(other_context, TA,
can_either_be_physically_accessed_or_supplied_by_malicious_hardware) :-
relevant_to(516, AG), has_target(AG, TA), not other_context(TA,
can_either_be_physically_accessed_or_supplied_by_malicious_hardware), not
no_other_context(TA,
can_either_be_physically_accessed_or_supplied_by_malicious_hardware).
```

Appendix E

Attack Pattern Validation Results

Figure E.1: Attack strategy model

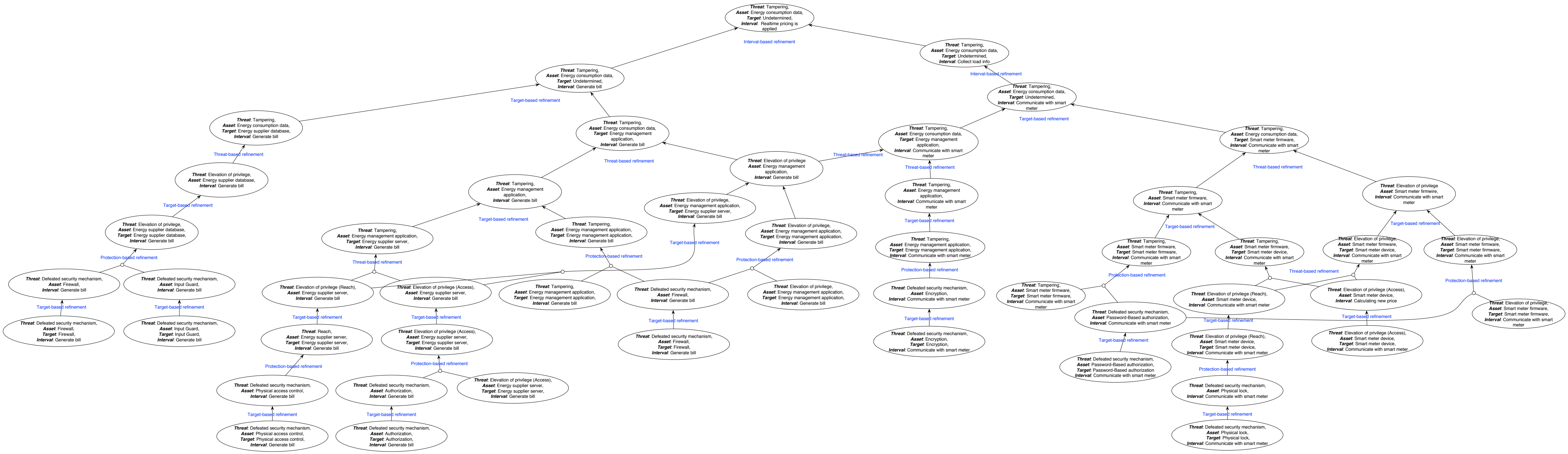

Threat: Tampering, Asset: Energy consumption data, Target: Undetermined, Interval: Realtime pricing is applied
Interval-based refinement
Threat: Tampering, Asset: Energy consumption data, Target: Undetermined, Interval: Collect load info
Interval-based refinement
Threat: Tampering, Asset: Energy consumption data, Target: Undetermined, Interval: Generate bill
Target-based refinement
Threat: Tampering, Asset: Energy consumption data, Target: Undetermined, Interval: Communicate with smart meter
Target-based refinement
Threat: Tampering, Asset: Energy consumption data, Target: Energy supplier database, Interval: Generate bill
Threat-based refinement
Threat: Elevation of privilege, Asset: Energy supplier database, Interval: Generate bill
Target-based refinement
Threat: Elevation of privilege, Asset: Energy supplier database, Target: Energy supplier database, Interval: Generate bill
Protection-based refinement
Threat: Defeated security mechanism, Asset: Firewall, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Input Guard, Interval: Generate bill
Target-based refinement
Target-based refinement
Threat: Defeated security mechanism, Asset: Firewall, Target: Firewall, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Input Guard, Target: Input Guard, Interval: Generate bill
Threat: Tampering, Asset: Energy consumption data, Target: Energy management application, Interval: Generate bill
Threat-based refinement
Threat: Tampering, Asset: Energy management application, Interval: Generate bill
Target-based refinement
Threat: Tampering, Asset: Energy management application, Target: Energy supplier server, Interval: Generate bill
Threat-based refinement
Threat: Elevation of privilege (Reach), Asset: Energy supplier server, Interval: Generate bill
Threat: Elevation of privilege (Access), Asset: Energy supplier server, Interval: Generate bill
Target-based refinement
Target-based refinement
Threat: Reach, Asset: Energy supplier server, Target: Energy supplier server, Interval: Generate bill
Threat: Elevation of privilege (Access), Asset: Energy supplier server, Target: Energy supplier server, Interval: Generate bill
Protection-based refinement
Protection-based refinement
Threat: Defeated security mechanism, Asset: Physical access control, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Authorization, Interval: Generate bill
Threat: Elevation of privilege (Access), Asset: Energy supplier server, Target: Energy supplier server, Interval: Generate bill
Target-based refinement
Target-based refinement
Threat: Defeated security mechanism, Asset: Physical access control, Target: Physical access control, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Authorization, Target: Authorization, Interval: Generate bill
Threat: Tampering, Asset: Energy management application, Target: Energy management application, Interval: Generate bill
Protection-based refinement
Threat: Tampering, Asset: Energy management application, Target: Energy management application, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Firewall, Interval: Generate bill
Target-based refinement
Threat: Defeated security mechanism, Asset: Firewall, Target: Firewall, Interval: Generate bill
Threat: Elevation of privilege Asset: Energy management application, Interval: Generate bill
Threat: Elevation of privilege, Asset: Energy management application, Target: Energy supplier server, Interval: Generate bill
Target-based refinement
Threat: Elevation of privilege, Asset: Energy management application, Target: Energy management application, Interval: Generate bill
Protection-based refinement
Threat: Elevation of privilege, Asset: Energy management application, Target: Energy management application, Interval: Generate bill
Threat-based refinement
Threat: Tampering, Asset: Energy consumption data, Target: Energy management application, Interval: Communicate with smart meter
Threat-based refinement
Threat: Tampering, Asset: Energy management application, Interval: Communicate with smart meter
Target-based refinement
Threat: Tampering, Asset: Energy management application, Target: Energy management application, Interval: Communicate with smart meter
Protection-based refinement
Threat: Defeated security mechanism, Asset: Encryption, Interval: Communicate with smart meter
Target-based refinement
Threat: Defeated security mechanism, Asset: Encryption, Target: Encryption, Interval: Communicate with smart meter
Threat: Tampering, Asset: Energy consumption data, Target: Smart meter firmware, Interval: Communicate with smart meter
Threat-based refinement
Threat: Tampering, Asset: Smart meter firmware, Interval: Communicate with smart meter
Target-based refinement
Threat: Tampering, Asset: Smart meter firmware, Target: Smart meter firmware, Interval: Communicate with smart meter
Threat: Tampering, Asset: Smart meter firmware, Target: Smart meter device, Interval: Communicate with smart meter
Protection-based refinement
Threat: Tampering, Asset: Smart meter firmware, Target: Smart meter firmware, Interval: Communicate with smart meter
Threat: Defeated security mechanism, Asset: Password-Based authorization, Interval: Communicate with smart meter
Target-based refinement
Threat: Defeated security mechanism, Asset: Password-Based authorization, Target: Password-Based authorization Interval: Communicate with smart meter
Threat-based refinement
Threat: Elevation of privilege (Reach), Asset: Smart meter device, Interval: Communicate with smart meter
Threat: Elevation of privilege (Access), Asset: Smart meter device, Interval: Calculating new price
Target-based refinement
Target-based refinement
Threat: Elevation of privilege (Reach), Asset: Smart meter device, Target: Smart meter device, Interval: Communicate with smart meter
Threat: Elevation of privilege (Access), Asset: Smart meter device, Target: Smart meter device, Interval: Communicate with smart meter
Protection-based refinement
Threat: Defeated security mechanism, Asset: Physical lock, Interval: Communicate with smart meter
Target-based refinement
Threat: Defeated security mechanism, Asset: Physical lock, Target: Physical lock, Interval: Communicate with smart meter
Threat: Elevation of privilege Asset: Smart meter firmwire, Interval: Communicate with smart meter
Target-based refinement
Threat: Elevation of privilege, Asset: Smart meter firmware, Target: Smart meter device, Interval: Communicate with smart meter
Threat: Elevation of privilege, Asset: Smart meter firmware, Target: Smart meter firmware, Interval: Communicate with smart meter
Protection-based refinement
Threat: Elevation of privilege, Asset: Smart meter firmware, Target: Smart meter firmware, Interval: Communicate with smart meter

Figure E.2: Operationalize attack strategies into relevant attack patterns

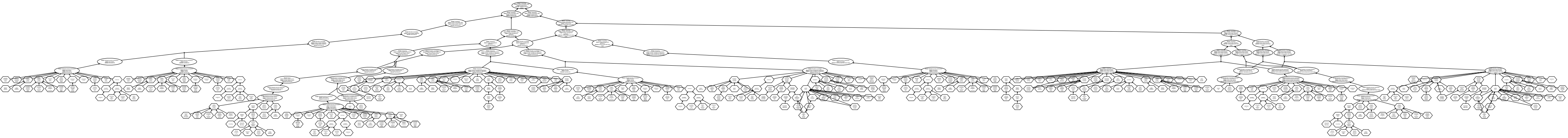
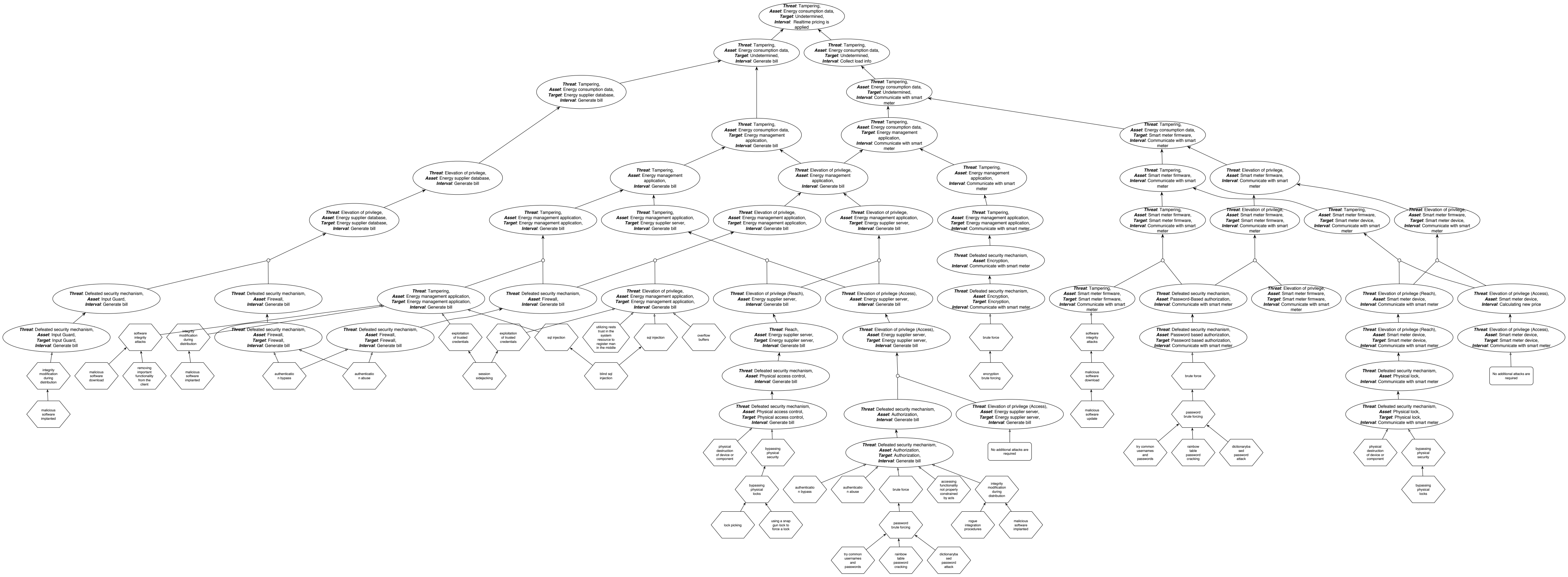


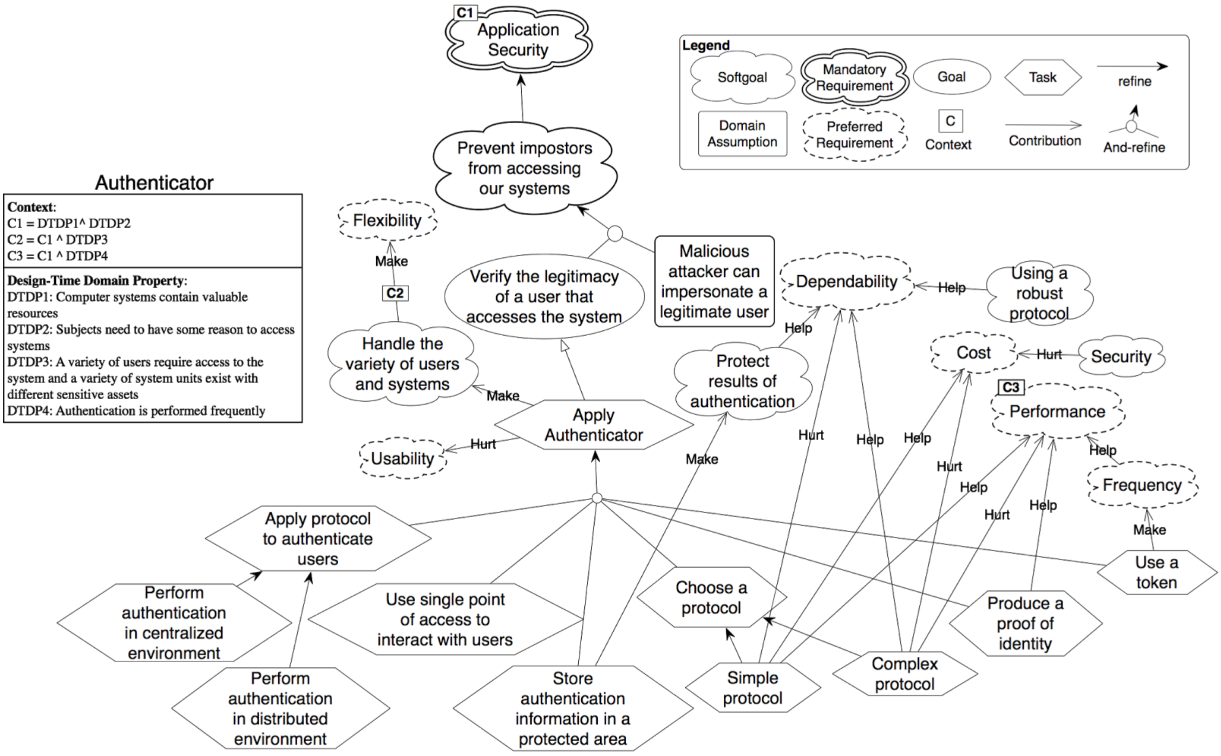
Figure E.3: Final attack model



Threat: Tampering, Asset: Energy consumption data, Target: Undetermined, Interval: Realtime pricing is applied
Threat: Tampering, Asset: Energy consumption data, Target: Undetermined, Interval: Generate bill
Threat: Tampering, Asset: Energy consumption data, Target: Undetermined, Interval: Collect load info
Threat: Tampering, Asset: Energy consumption data, Target: Energy supplier database, Interval: Generate bill
Threat: Tampering, Asset: Energy consumption data, Target: Undetermined, Interval: Communicate with smart meter
Threat: Tampering, Asset: Energy consumption data, Target: Energy management application, Interval: Generate bill
Threat: Tampering, Asset: Energy consumption data, Target: Energy management application, Interval: Communicate with smart meter
Threat: Tampering, Asset: Energy consumption data, Target: Smart meter firmware, Interval: Communicate with smart meter
Threat: Elevation of privilege, Asset: Energy supplier database, Interval: Generate bill
Threat: Tampering, Asset: Energy management application, Interval: Generate bill
Threat: Elevation of privilege, Asset: Energy management application, Interval: Generate bill
Threat: Tampering, Asset: Energy management application, Interval: Communicate with smart meter
Threat: Tampering, Asset: Smart meter firmware, Interval: Communicate with smart meter
Threat: Elevation of privilege, Asset: Smart meter firmware, Interval: Communicate with smart meter
Threat: Elevation of privilege, Asset: Energy supplier database, Target: Energy supplier database, Interval: Generate bill
Threat: Tampering, Asset: Energy management application, Target: Energy management application, Interval: Generate bill
Threat: Tampering, Asset: Energy management application, Target: Energy supplier server, Interval: Generate bill
Threat: Elevation of privilege, Asset: Energy management application, Target: Energy management application, Interval: Generate bill
Threat: Elevation of privilege, Asset: Energy management application, Target: Energy supplier server, Interval: Generate bill
Threat: Tampering, Asset: Energy management application, Target: Energy management application, Interval: Communicate with smart meter
Threat: Tampering, Asset: Smart meter firmware, Target: Smart meter firmware, Interval: Communicate with smart meter
Threat: Elevation of privilege, Asset: Smart meter firmware, Target: Smart meter firmware, Interval: Communicate with smart meter
Threat: Tampering, Asset: Smart meter firmware, Target: Smart meter device, Interval: Communicate with smart meter
Threat: Elevation of privilege, Asset: Smart meter firmware, Target: Smart meter device, Interval: Communicate with smart meter
Threat: Defeated security mechanism, Asset: Encryption, Interval: Communicate with smart meter
Threat: Defeated security mechanism, Asset: Input Guard, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Firewall, Interval: Generate bill
Threat: Tampering, Asset: Energy management application, Target: Energy management application, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Firewall, Interval: Generate bill
Threat: Elevation of privilege, Asset: Energy management application, Target: Energy management application, Interval: Generate bill
Threat: Elevation of privilege (Reach), Asset: Energy supplier server, Interval: Generate bill
Threat: Elevation of privilege (Access), Asset: Energy supplier server, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Encryption, Target: Encryption, Interval: Communicate with smart meter
Threat: Tampering, Asset: Smart meter firmware, Target: Smart meter firmware, Interval: Communicate with smart meter
Threat: Defeated security mechanism, Asset: Password-Based authorization, Interval: Communicate with smart meter
Threat: Elevation of privilege, Asset: Smart meter firmware, Target: Smart meter firmware, Interval: Communicate with smart meter
Threat: Elevation of privilege (Reach), Asset: Smart meter device, Interval: Communicate with smart meter
Threat: Elevation of privilege (Access), Asset: Smart meter device, Interval: Calculating new price
Threat: Defeated security mechanism, Asset: Input Guard, Target: Input Guard, Interval: Generate bill
software integrity attacks
integrity modification during distribution
Threat: Defeated security mechanism, Asset: Firewall, Target: Firewall, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Firewall, Target: Firewall, Interval: Generate bill
exploitation of trusted credentials
exploitation of trusted credentials
sql injection
utilizing rests trust in the system resource to register man in the middle
sql injection
overflow buffers
Threat: Reach, Asset: Energy supplier server, Target: Energy supplier server, Interval: Generate bill
Threat: Elevation of privilege (Access), Asset: Energy supplier server, Target: Energy supplier server, Interval: Generate bill
brute force
software integrity attacks
Threat: Defeated security mechanism, Asset: Password based authorization, Target: Password based authorization, Interval: Communicate with smart meter
Threat: Elevation of privilege (Reach), Asset: Smart meter device, Target: Smart meter device, Interval: Communicate with smart meter
Threat: Elevation of privilege (Access), Asset: Smart meter device, Target: Smart meter device, Interval: Communicate with smart meter
integrity modification during distribution
malicious software download
removing important functionality from the client
malicious software implanted
authentication bypass
authentication abuse
session sidejacking
blind sql injection
Threat: Defeated security mechanism, Asset: Physical access control, Interval: Generate bill
encryption brute forcing
malicious software download
brute force
Threat: Defeated security mechanism, Asset: Physical lock, Interval: Communicate with smart meter
No additional attacks are required
malicious software implanted
Threat: Defeated security mechanism, Asset: Physical access control, Target: Physical access control, Interval: Generate bill
Threat: Defeated security mechanism, Asset: Authorization, Interval: Generate bill
Threat: Elevation of privilege (Access), Asset: Energy supplier server, Target: Energy supplier server, Interval: Generate bill
malicious software update
password brute forcing
Threat: Defeated security mechanism, Asset: Physical lock, Target: Physical lock, Interval: Communicate with smart meter
physical destruction of device or component
bypassing physical security
Threat: Defeated security mechanism, Asset: Authorization, Target: Authorization, Interval: Generate bill
No additional attacks are required
try common usernames and passwords
rainbow table password cracking
dictionarybased password attack
physical destruction of device or component
bypassing physical security
bypassing physical locks
authentication bypass
authentication abuse
brute force
accessing functionality not properly constrained by acls
integrity modification during distribution
bypassing physical locks
lock picking
using a snap gun lock to force a lock
password brute forcing
rogue integration procedures
malicious software implanted
try common usernames and passwords
rainbow table password cracking
dictionarybased password attack

Appendix F

Goal Models Built from Security Patterns


C1
Application Security
Legend
Softgoal
Mandatory Requirement
Goal
Task
refine
Domain Assumption
Preferred Requirement
C
Context
Contribution
And-refine
Prevent impostors from accessing our systems
Flexibility
Make
C2
Verify the legitimacy of a user that accesses the system
Malicious attacker can impersonate a legitimate user
Dependability
Help
Using a robust protocol
Handle the variety of users and systems
Make
Help
Protect results of authentication
Cost
Hurt
Security
C3
Performance
Apply Authenticator
Usability
Hurt
Make
Hurt
Help
Help
Hurt
Help
Hurt
Help
Help
Frequency
Make
Apply protocol to authenticate users
Use a token
Perform authentication in centralized environment
Use single point of access to interact with users
Choose a protocol
Produce a proof of identity
Simple protocol
Complex protocol
Perform authentication in distributed environment
Store authentication information in a protected area

Authenticator

Context:
C1 = DTDP1^ DTDP2
C2 = C1 ^ DTDP3
C3 = C1 ^ DTDP4

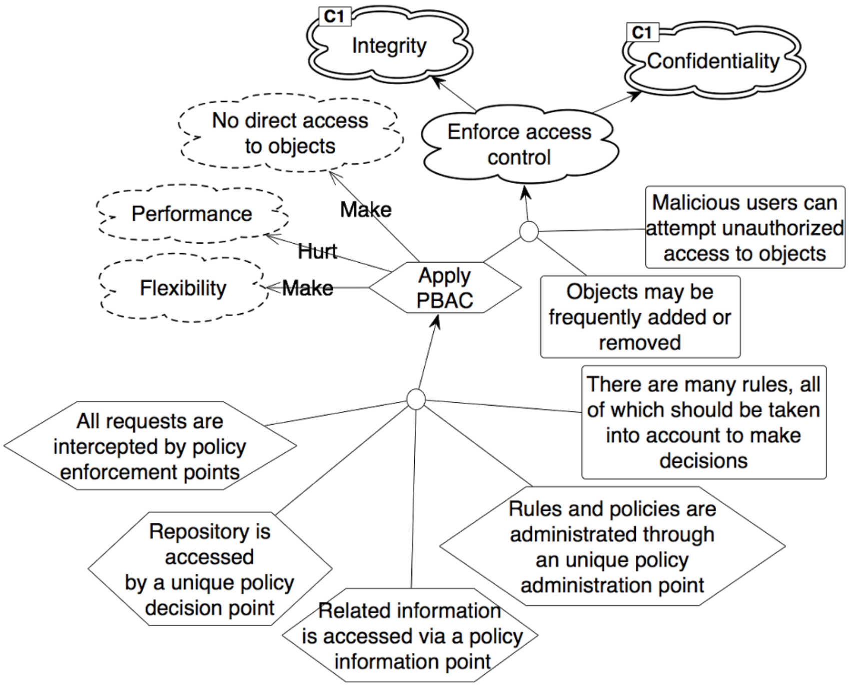
Design-Time Domain Property:
DTDP1: Computer systems contain valuable resources
DTDP2: Subjects need to have some reason to access systems
DTDP3: A variety of users require access to the system and a variety of system units exist with different sensitive assets
DTDP4: Authentication is performed frequently

14

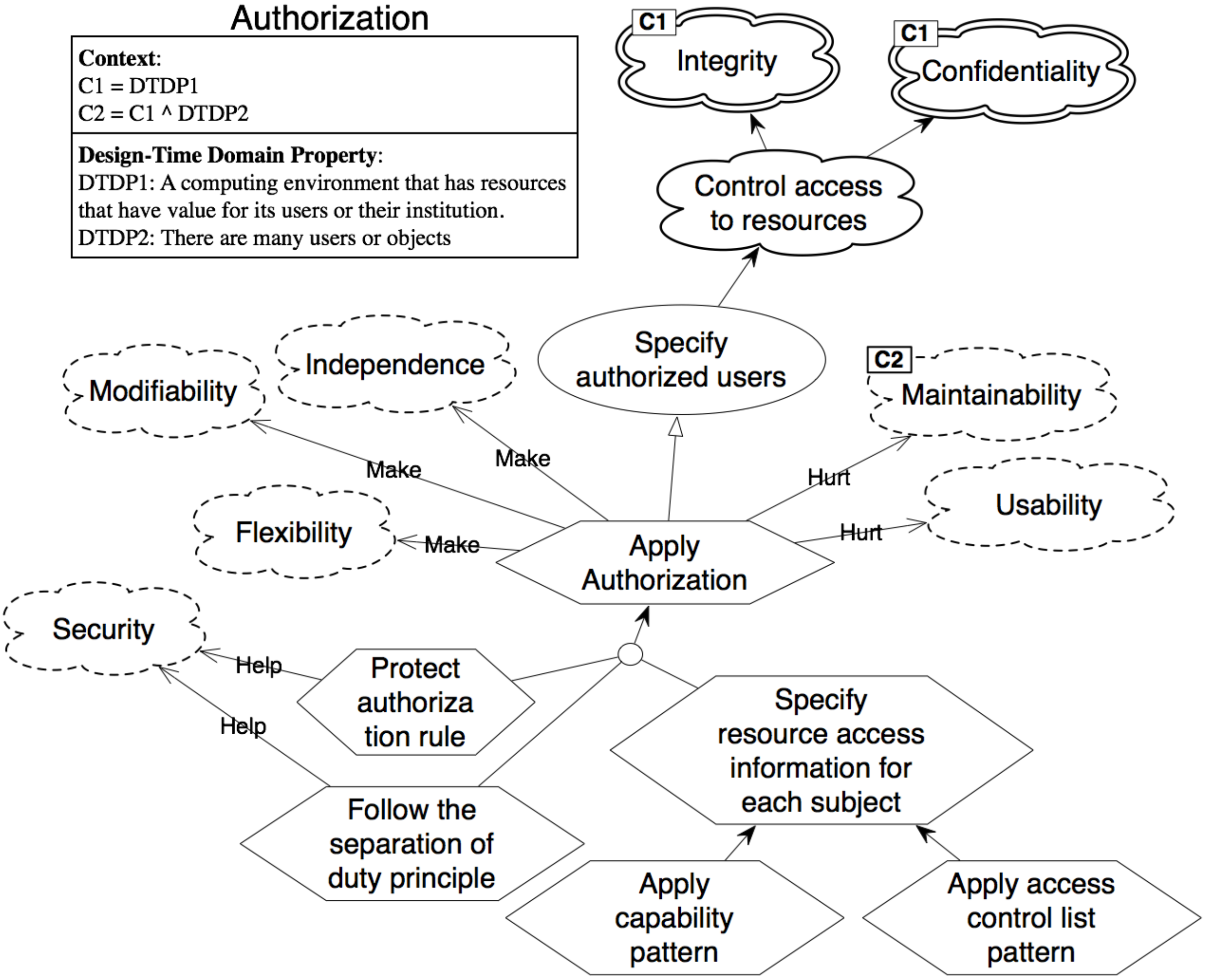
PBAC

Context:
C1 = DTDP1 ^ DTDP2 ^ DTDP3 ^ DTDP4
C2 = C1 ^ DTDP5 ^ DTDP6

Design-Time Domain Property:
DTDP1: Consider centralized or distributed systems with a large number of resources (objects)
DTDP2: A large number of subjects may access those objects
DTDP3: Rules are defined to control access to objects, which are typically designed by different actors.
DTDP4: Each set of rules designed by a specific policy designer can concern overlapping sets


C1
Integrity
C1
Confidentiality
No direct access to objects
Enforce access control
Performance
Make
Hurt
Malicious users can attempt unauthorized access to objects
Flexibility
Make
Apply PBAC
Objects may be frequently added or removed
There are many rules, all of which should be taken into account to make decisions
All requests are intercepted by policy enforcement points
Repository is accessed by a unique policy decision point
Rules and policies are administrated through an unique policy administration point
Related information is accessed via a policy information point

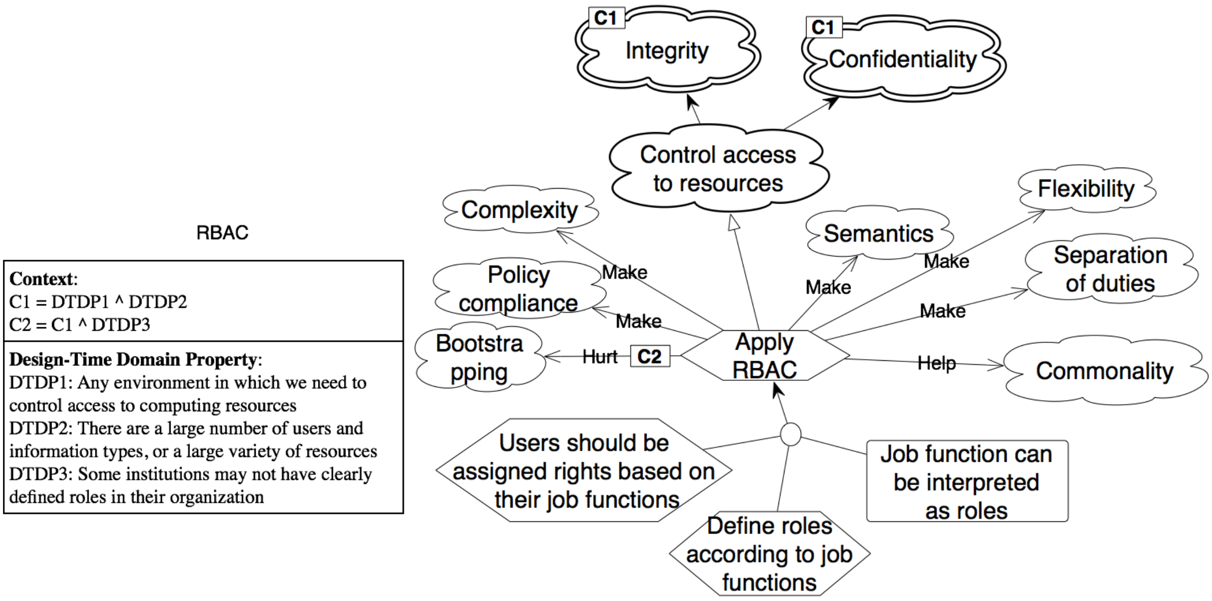
Authorization



Context:
C1 = DTDP1
C2 = C1 ^ DTDP2

Design-Time Domain Property:
DTDP1: A computing environment that has resources that have value for its users or their institution.
DTDP2: There are many users or objects

RBAC



Context:
C1 = DTDP1 ^ DTDP2
C2 = C1 ^ DTDP3

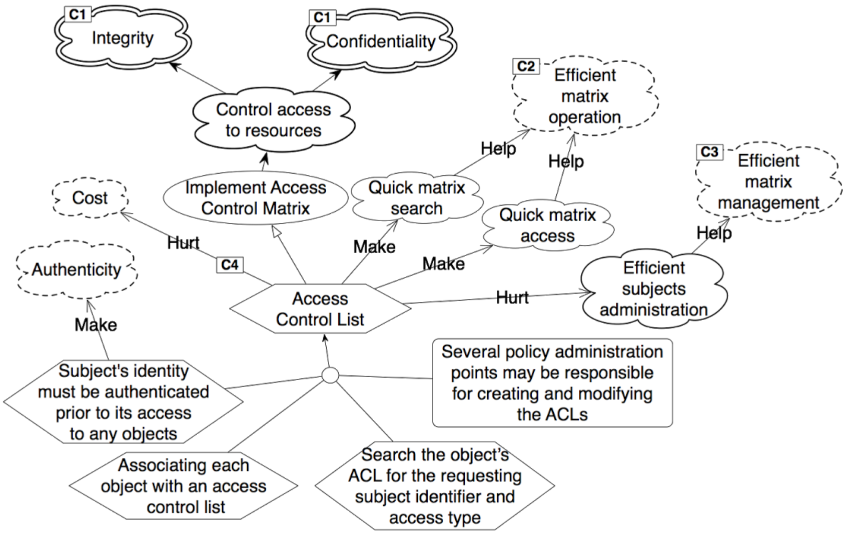
Design-Time Domain Property:
DTDP1: Any environment in which we need to control access to computing resources
DTDP2: There are a large number of users and information types, or a large variety of resources
DTDP3: Some institutions may not have clearly defined roles in their organization

C1
Integrity
C1
Confidentiality
Control access to resources
Complexity
Flexibility
Semantics
Policy compliance
Make
Make
Make
Separation of duties
Make
Make
Bootstra pping
Hurt
C2
Apply RBAC
Help
Commonality
Users should be assigned rights based on their job functions
Job function can be interpreted as roles
Define roles according to job functions

Access Control List

Context:
C1 = DTDP1 ^ DTDP2 ^ DTDP3 ^ DTDP4
C2 = C1 ^ DTDP5 ^ DTDP6
C3 = C1 ^ DTDP7
C4 = C1 ^ DTDP8

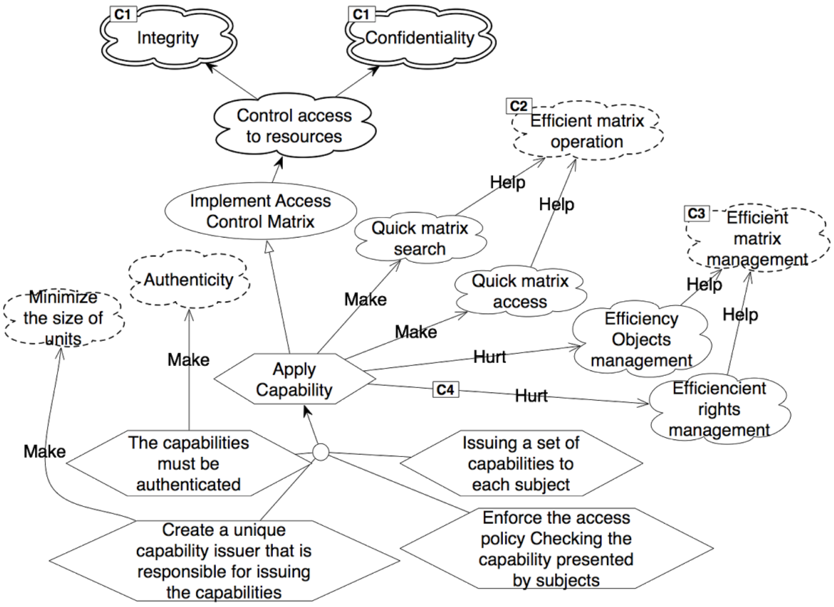
Design-Time Domain Property:
DTDP1: Centralized or distributed systems in which access to resources must be controlled
DTDP2: The systems comprise a policy decision point and policy enforcement points that enforce the access policy
DTDP3: A system has subjects that need to access resources to perform tasks.
DTDP4: Access rights are defined and can be modeled as an access matrix
DTDP5: The number of subjects and/or objects can be large
DTDP6: The access control matrix is sparse
DTDP7: Subjects and objects may be frequently added or removed
DTDP8: The environment is heterogeneous


C1
Integrity
C1
Confidentiality
Control access to resources
C2
Efficient matrix operation
C3
Efficient matrix management
Implement Access Control Matrix
Quick matrix search
Quick matrix access
Help
Help
Help
Cost
Hurt
C4
Make
Make
Authenticity
Access Control List
Hurt
Efficient subjects administration
Make
Several policy administration points may be responsible for creating and modifying the ACLs
Subject's identity must be authenticated prior to its access to any objects
Associating each object with an access control list
Search the object's ACL for the requesting subject identifier and access type

Capability

Context:
C1 = DTDP1 ^ DTDP2 ^ DTDP3 ^ DTDP4 ^ DTDP5
C2 = C1 ^ DTDP6 ^ DTDP7
C3 = C1 ^ DTDP8
C4 = C1 ^ DTDP9

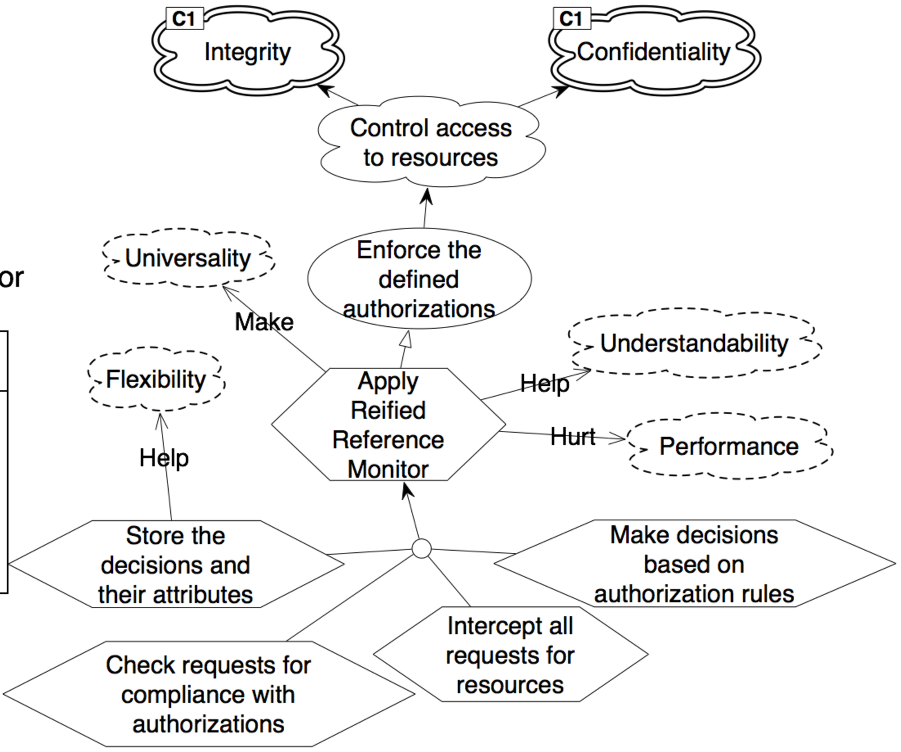
Domain Property:
DTDP1: Centralized or distributed systems in which access to resources must be controlled
DTDP2: The systems comprise a policy decision point and policy enforcement points that enforce the access policy
DTDP3: A system has subjects that need to access resources to perform tasks.
DTDP4: Access rights are defined and can be modeled as an access matrix
DTDP5: The system's implementation is vulnerable to threats from attackers that may compromise its components.
DTDP6: The number of subjects and/or objects can be large
DTDP7: The access control matrix is sparse
DTDP8: Subjects and objects may be frequently added or removed
DTDP9: The environment is heterogeneous


C1
Integrity
C1
Confidentiality
Control access to resources
C2
Efficient matrix operation
Implement Access Control Matrix
Help
Quick matrix search
Help
C3
Efficient matrix management
Authenticity
Quick matrix access
Minimize the size of units
Make
Help
Make
Efficiency Objects management
Help
Make
Hurt
Apply Capability
C4
Hurt
Efficiencient rights management
Make
The capabilities must be authenticated
Issuing a set of capabilities to each subject
Create a unique capability issuer that is responsible for issuing the capabilities
Enforce the access policy Checking the capability presented by subjects

Reified Reference Monitor

Context:
C1 = DTDP1 ^ DTDP2

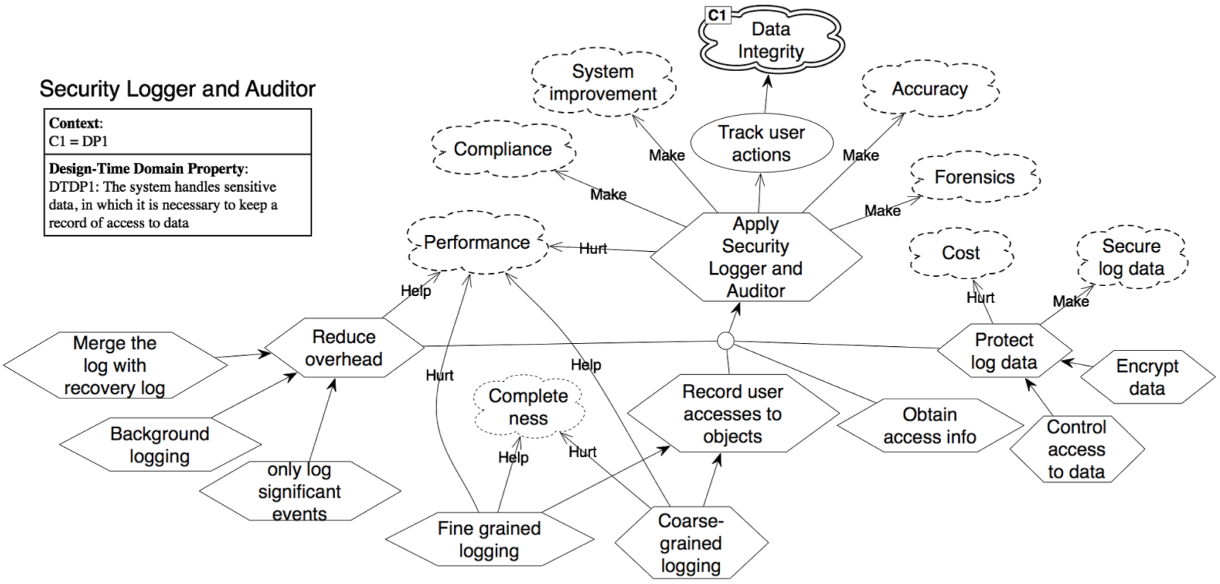
Design-Time Domain Property:
DTDP1: A multiprocessing environment in which subjects request protection objects to perform their functions
DTDP2: Subjects access resources based on a decision made by a reference monitor

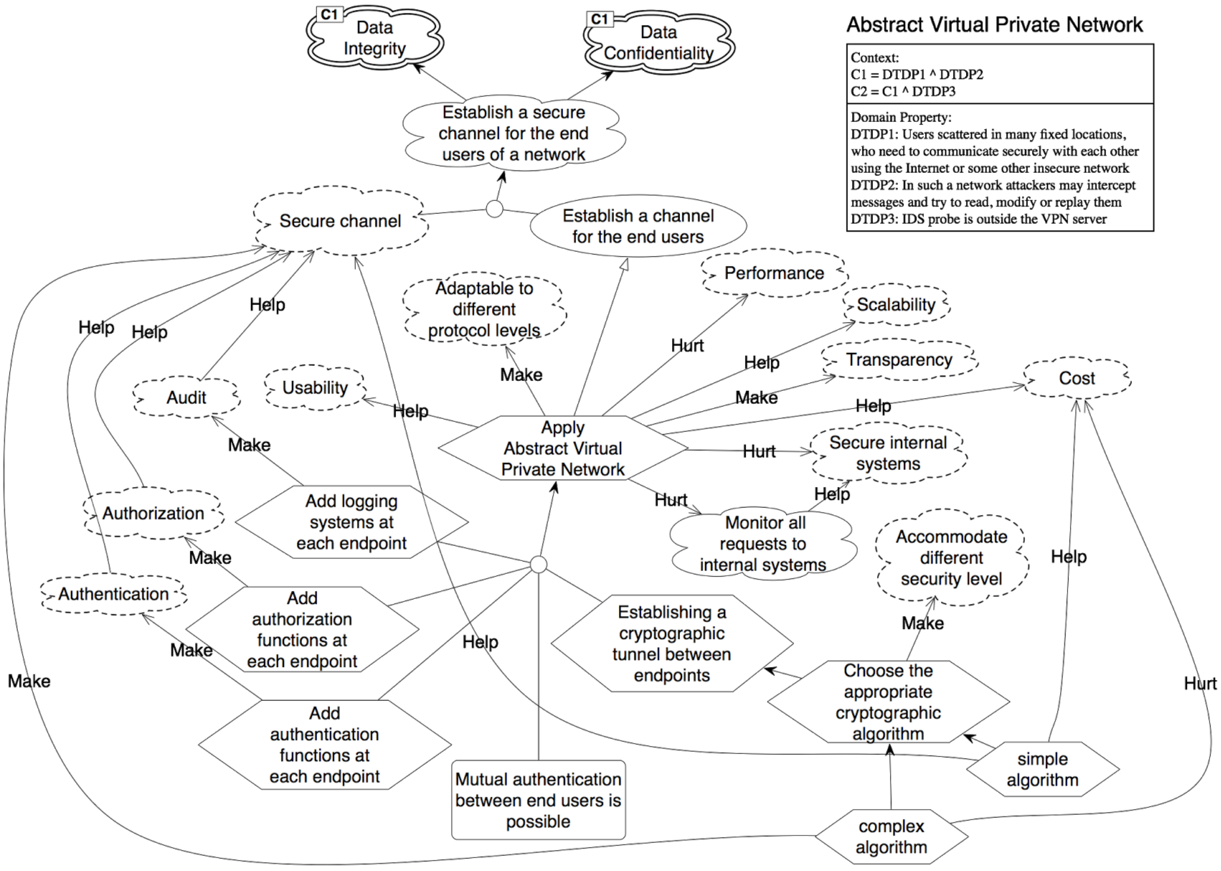

C1
Integrity
C1
Confidentiality
Control access to resources
Universality
Enforce the defined authorizations
Make
Understandability
Flexibility
Apply Reified Reference Monitor
Help
Hurt
Performance
Help
Store the decisions and their attributes
Make decisions based on authorization rules
Intercept all requests for resources
Check requests for compliance with authorizations

Security Logger and Auditor

Context:
C1 = DP1

Design-Time Domain Property:
DTDP1: The system handles sensitive data, in which it is necessary to keep a record of access to data




C1
Data Integrity
C1
Data Confidentiality
Establish a secure channel for the end users of a network
Secure channel
Establish a channel for the end users
Adaptable to different protocol levels
Performance
Scalability
Transparency
Cost
Help
Help
Help
Audit
Usability
Help
Make
Hurt
Help
Make
Help
Apply Abstract Virtual Private Network
Hurt
Secure internal systems
Make
Hurt
Help
Monitor all requests to internal systems
Authorization
Add logging systems at each endpoint
Make
Accommodate different security level
Help
Authentication
Add authorization functions at each endpoint
Establishing a cryptographic tunnel between endpoints
Make
Make
Help
Make
Choose the appropriate cryptographic algorithm
Hurt
Add authentication functions at each endpoint
simple algorithm
Mutual authentication between end users is possible
complex algorithm

Abstract Virtual Private Network

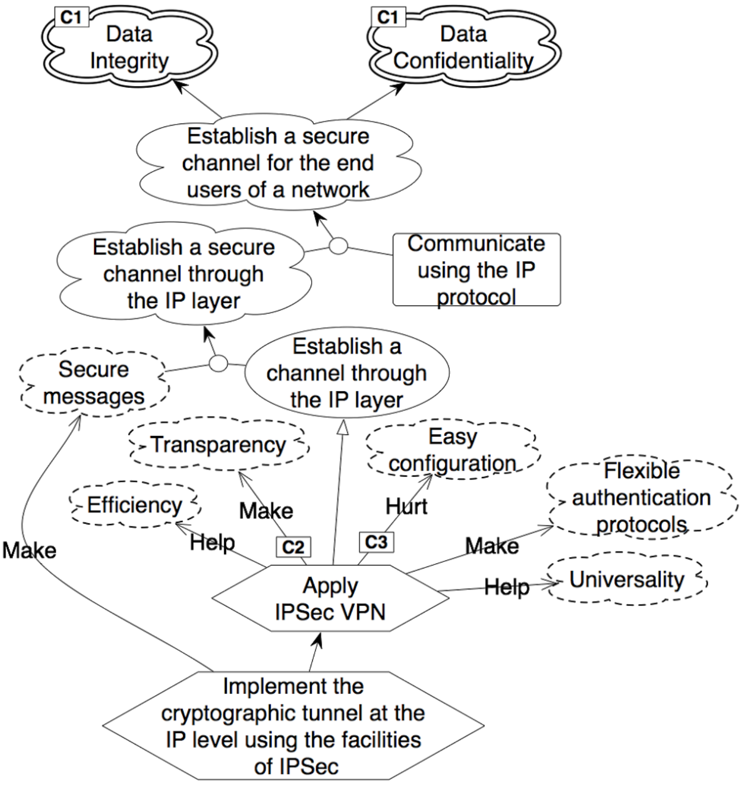
Context:
C1 = DTDP1 ^ DTDP2
C2 = C1 ^ DTDP3

Domain Property:
DTDP1: Users scattered in many fixed locations, who need to communicate securely with each other using the Internet or some other insecure network
DTDP2: In such a network attackers may intercept messages and try to read, modify or replay them
DTDP3: IDS probe is outside the VPN server

IPSec VPN

Context:
C1 = DTDP1 ^ DTDP2
C2 = C1 ^ DTDP3
C3 = C1 ^ DTDP4

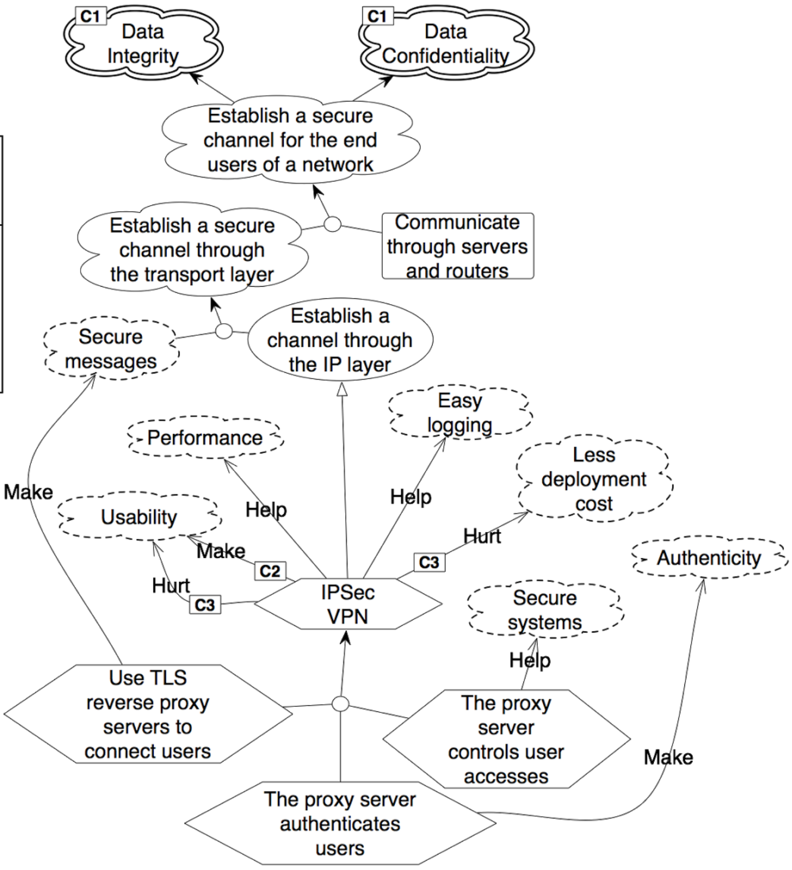
Design-Time Domain Property:
DTDP1: Users scattered in many fixed locations, who need to communicate securely with each other using the Internet or some other insecure network.
DTDP2: In such a network attackers may intercept messages and try to read, modify or replay them.
DTDP3: The system is in gateway-to-gateway architectures
DTDP4: The system is in host-to-gateway or host-to-host architecture.

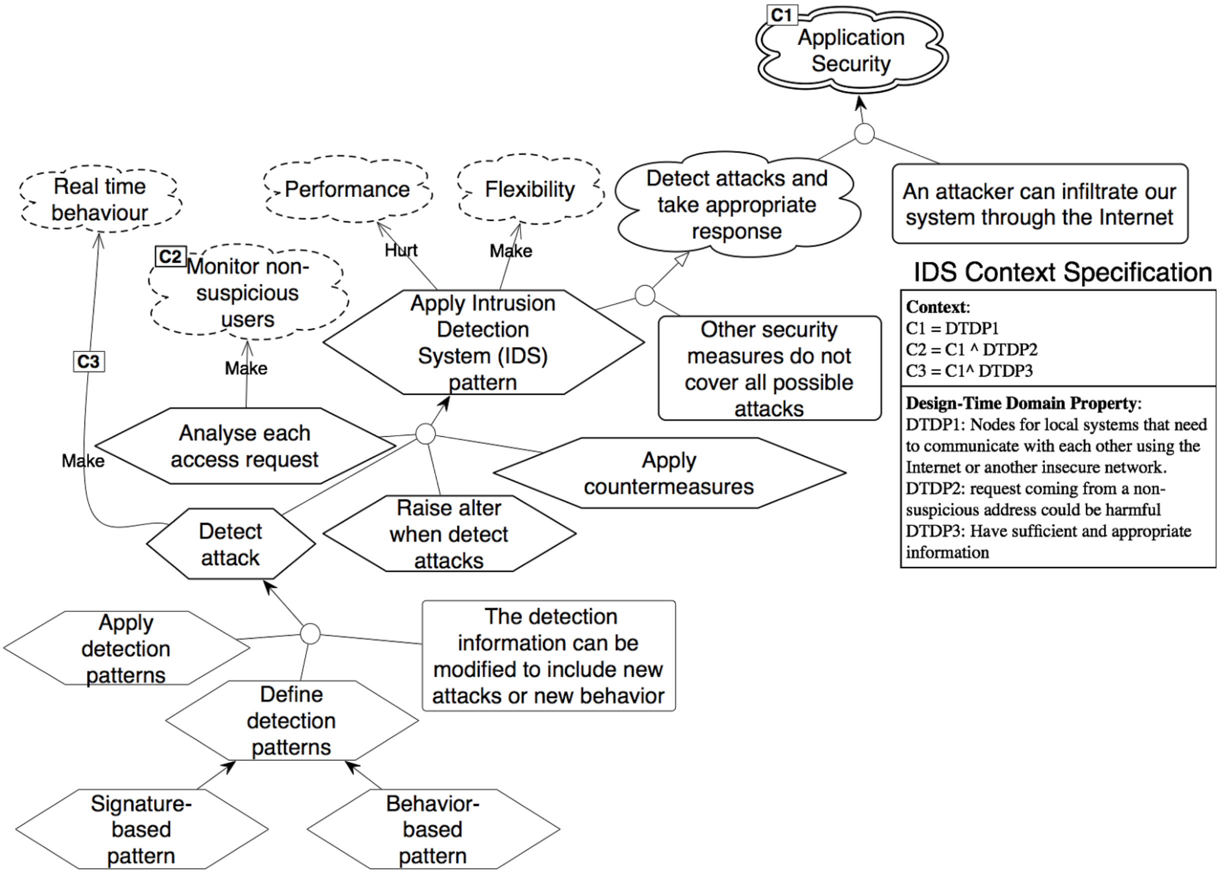

C1
Data Integrity
C1
Data Confidentiality
Establish a secure channel for the end users of a network
Establish a secure channel through the IP layer
Communicate using the IP protocol
Secure messages
Establish a channel through the IP layer
Transparency
Easy configuration
Efficiency
Make
Hurt
Flexible authentication protocols
Make
Help
C2
C3
Make
Apply IPSec VPN
Help
Universality
Implement the cryptographic tunnel at the IP level using the facilities of IPSec

TLS VPN

Context:
C1 = DTDP1 ^ DTDP2
C2 = C1 ^ DTDP3
C3 = C1 ^ DTDP4

Design-Time Domain Property:
DTDP1: Users scattered in many fixed locations, who need to communicate securely with each other using the Internet or some other insecure network.
DTDP2: In such a network attackers may intercept messages and try to read, modify or replay them.
DTDP3: Access is needed only for web-based applications.
DTDP4: Need to access non-web-based applications.


C1
Data Integrity
C1
Data Confidentiality
Establish a secure channel for the end users of a network
Establish a secure channel through the transport layer
Communicate through servers and routers
Secure messages
Establish a channel through the IP layer
Easy logging
Performance
Less deployment cost
Make
Usability
Help
Help
Hurt
Make
C2
C3
Authenticity
Hurt
C3
IPSec VPN
Secure systems
Help
Use TLS reverse proxy servers to connect users
The proxy server controls user accesses
Make
The proxy server authenticates users


C1
Application Security
An attacker can infiltrate our system through the Internet
Detect attacks and take appropriate response
Real time behaviour
Performance
Flexibility
Hurt
Make
C2
Monitor non-suspicious users
Make
C3
Apply Intrusion Detection System (IDS) pattern
Other security measures do not cover all possible attacks
Analyse each access request
Make
Apply countermeasures
Detect attack
Raise alter when detect attacks
Apply detection patterns
The detection information can be modified to include new attacks or new behavior
Define detection patterns
Signature-based pattern
Behavior-based pattern

IDS Context Specification

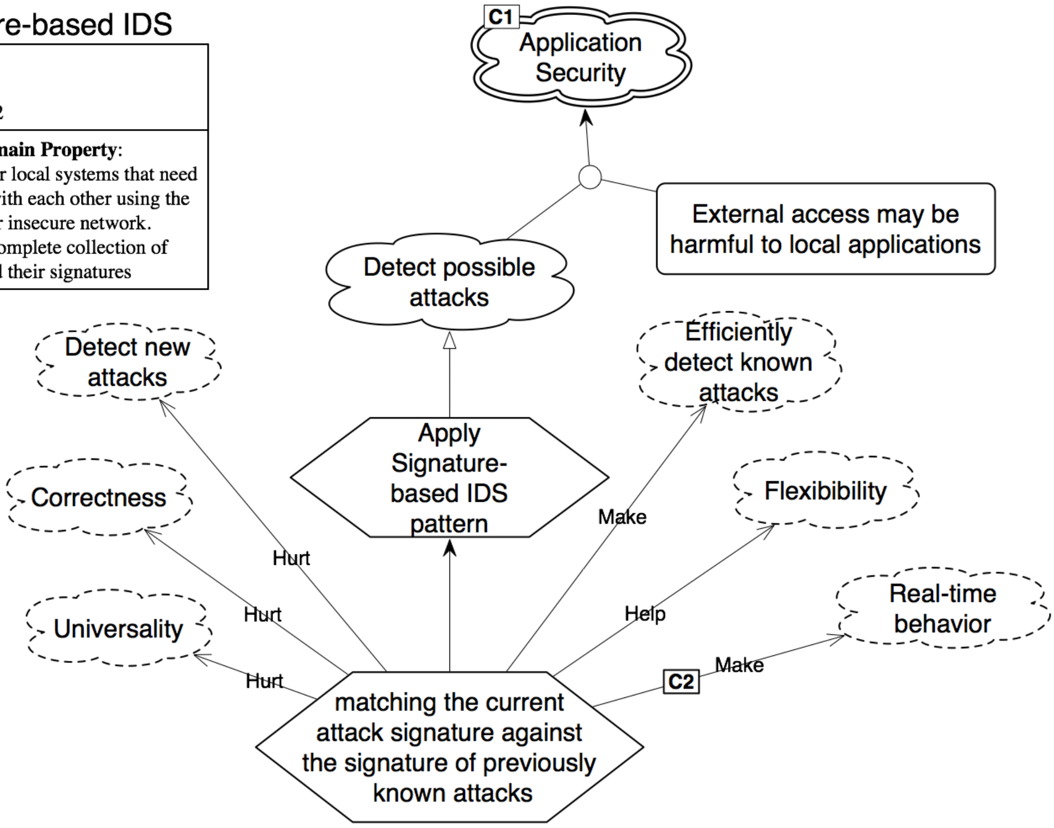
Context:
C1 = DTDP1
C2 = C1 ^ DTDP2
C3 = C1^ DTDP3

Design-Time Domain Property:
DTDP1: Nodes for local systems that need to communicate with each other using the Internet or another insecure network.
DTDP2: request coming from a non-suspicious address could be harmful
DTDP3: Have sufficient and appropriate information

Signature-based IDS

Context:
C1 = DTDP1
C2 = C1 ^ DTDP2

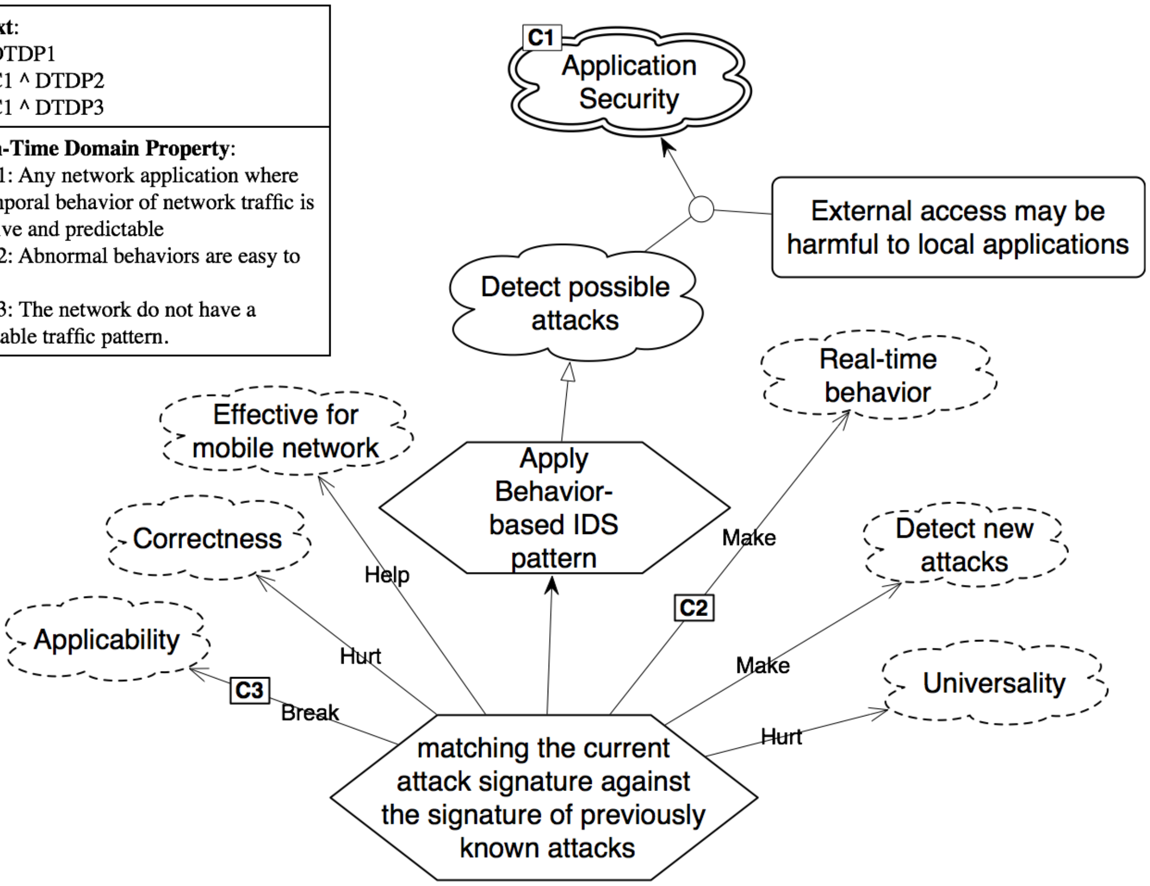
Design-Time Domain Property:
DTDP1: Nodes for local systems that need to communicate with each other using the Internet or another insecure network.
DTDP2: Have a complete collection of known attacks and their signatures


C1
Application Security
External access may be harmful to local applications
Detect possible attacks
Detect new attacks
Efficiently detect known attacks
Apply Signature-based IDS pattern
Correctness
Flexibility
Universality
Real-time behavior
Hurt
Hurt
Hurt
Make
Help
Make
C2
matching the current attack signature against the signature of previously known attacks

Behavior-based IDS

Context:
C1 = DTDP1
C2 = C1 ^ DTDP2
C3 = C1 ^ DTDP3

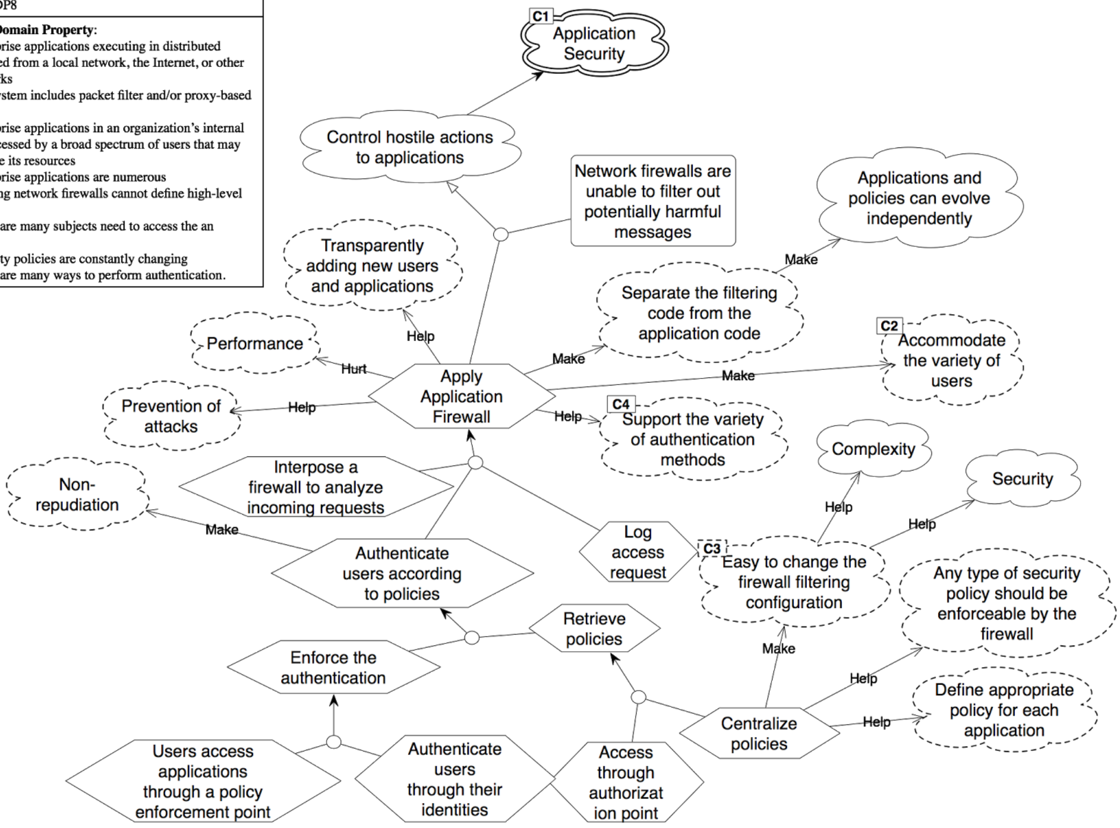
Design-Time Domain Property:
DTDP1: Any network application where the temporal behavior of network traffic is repetitive and predictable
DTDP2: Abnormal behaviors are easy to detect
DTDP3: The network do not have a predictable traffic pattern.


C1
Application Security
External access may be harmful to local applications
Detect possible attacks
Real-time behavior
Effective for mobile network
Apply Behavior-based IDS pattern
Correctness
Make
Help
Detect new attacks
C2
Applicability
Hurt
Make
Universality
C3
Break
Hurt
matching the current attack signature against the signature of previously known attacks

Application Firewall

Context:
C1 = DTDP1 ^ DTDP2 ^ DTDP3 ^ DTDP4 ^ DTDP5
C2 = C1 ^ DTDP6
C3 = C1 ^ DTDP7
C4 = C1 ^ DTDP8

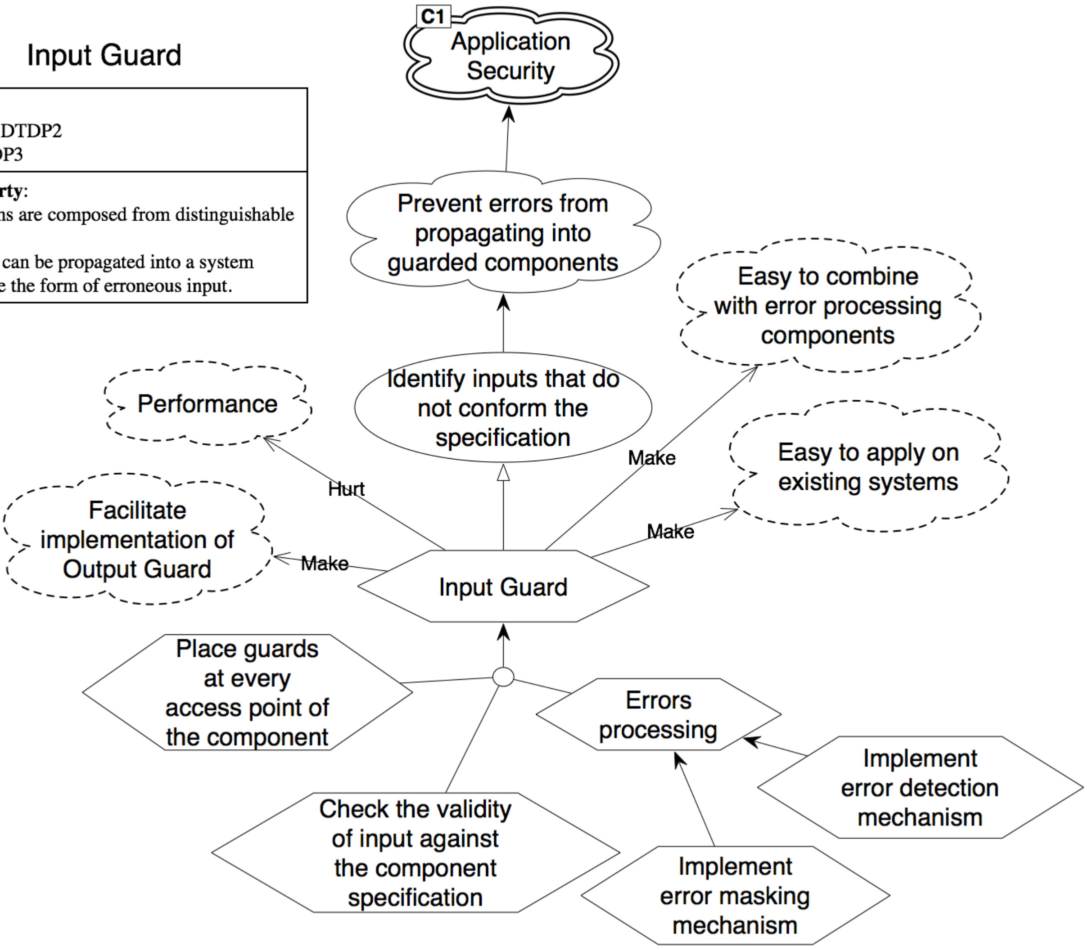
Design-Time Domain Property:
DTDP1: Enterprise applications executing in distributed systems accessed from a local network, the Internet, or other external networks
DTDP2: The system includes packet filter and/or proxy-based firewalls
DTDP3: Enterprise applications in an organization's internal network are accessed by a broad spectrum of users that may attempt to abuse its resources
DTDP4: Enterprise applications are numerous
DTDP5: Existing network firewalls cannot define high-level rules.
DTDP6: There are many subjects need to access the an application.
DTDP7: Security policies are constantly changing
DTDP8: There are many ways to perform authentication.


C1
Application Security
Control hostile actions to applications
Network firewalls are unable to filter out potentially harmful messages
Applications and policies can evolve independently
Transparently adding new users and applications
Make
Separate the filtering code from the application code
Performance
Help
Hurt
Make
C2
Accommodate the variety of users
Make
Apply Application Firewall
Prevention of attacks
Help
Help
C4
Support the variety of authentication methods
Complexity
Security
Interpose a firewall to analyze incoming requests
Non-repudiation
Help
Help
Make
Log access request
C3
Easy to change the firewall filtering configuration
Authenticate users according to policies
Any type of security policy should be enforceable by the firewall
Retrieve policies
Make
Enforce the authentication
Help
Define appropriate policy for each application
Centralize policies
Help
Users access applications through a policy enforcement point
Authenticate users through their identities
Access through authorizat ion point

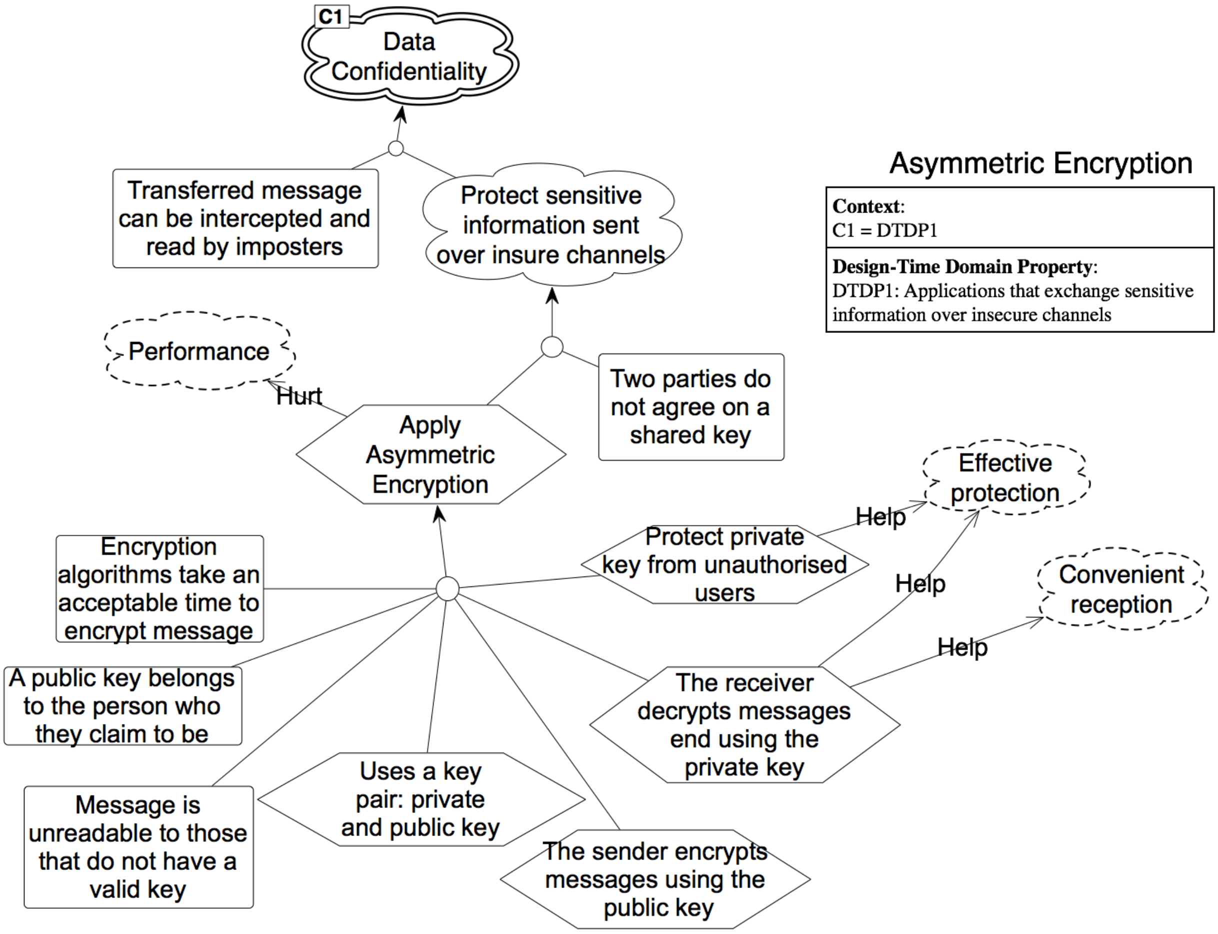
Input Guard

Context:
C1 = DTDP1 ^ DTDP2
C2 = C1 ^ DTDP3

Domain Property:
DTDP1: Systems are composed from distinguishable components.
DTDP2: Errors can be propagated into a system component have the form of erroneous input.


C1
Application Security
Prevent errors from propagating into guarded components
Easy to combine with error processing components
Performance
Identify inputs that do not conform the specification
Make
Easy to apply on existing systems
Hurt
Facilitate implementation of Output Guard
Make
Make
Input Guard
Place guards at every access point of the component
Errors processing
Implement error detection mechanism
Check the validity of input against the component specification
Implement error masking mechanism

Asymmetric Encryption


C1
Data Confidentiality
Transferred message can be intercepted and read by imposters
Protect sensitive information sent over insure channels
Performance
Hurt
Apply Asymmetric Encryption
Two parties do not agree on a shared key
Effective protection
Help
Protect private key from unauthorised users
Encryption algorithms take an acceptable time to encrypt message
Help
Convenient reception
Help
A public key belongs to the person who they claim to be
The receiver decrypts messages end using the private key
Message is unreadable to those that do not have a valid key
Uses a key pair: private and public key
The sender encrypts messages using the public key

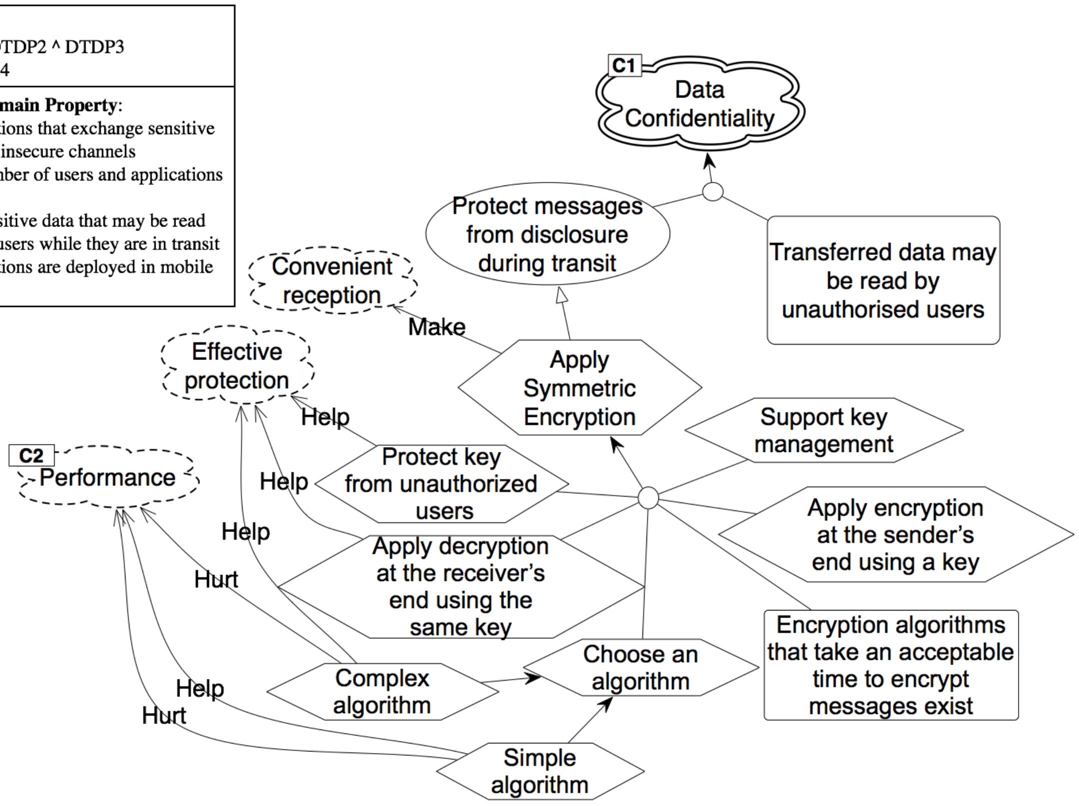
Context:
C1 = DTDP1

Design-Time Domain Property:
DTDP1: Applications that exchange sensitive information over insecure channels

Symmetric Encryption

Context:
C1 = DTDP1 ^ DTDP2 ^ DTDP3
C2 = C1 ^ DTDP4

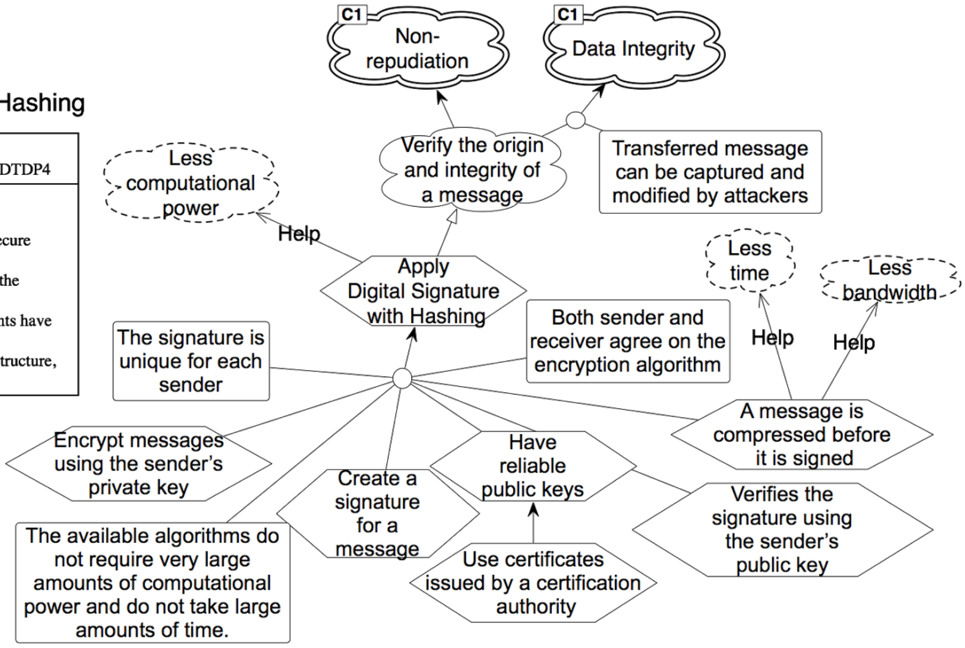
Design-Time Domain Property:
DTDP1: Applications that exchange sensitive information over insecure channels
DTDP2: The number of users and applications is not very large
DTDP3: The sensitive data that may be read by unauthorized users while they are in transit
DTDP4: Applications are deployed in mobile devices.


C1
Data Confidentiality
Protect messages from disclosure during transit
Transferred data may be read by unauthorised users
Convenient reception
Make
Apply Symmetric Encryption
Effective protection
Help
C2
Performance
Help
Protect key from unauthorized users
Support key management
Apply encryption at the sender's end using a key
Help
Hurt
Apply decryption at the receiver's end using the same key
Encryption algorithms that take an acceptable time to encrypt messages exist
Choose an algorithm
Complex algorithm
Help
Hurt
Simple algorithm

Digital Signature with Hashing

Context:
C1 = DTDP1 ^ DTDP2 ^ DTDP3 ^ DTDP4

Design-Time Domain Property:
DTDP1: Applications that exchange documents or messages through insecure channels
DTDP2: The origin and integrity of the information need to be preserved
DTDP3: Those exchanging documents have access to a public key system
DTDP4: There is a public key infrastructure, which handle the public keys.


C1
Non-repudiation
C1
Data Integrity
Verify the origin and integrity of a message
Transferred message can be captured and modified by attackers
Less computational power
Help
Apply Digital Signature with Hashing
Less time
Less bandwidth
Both sender and receiver agree on the encryption algorithm
Help
Help
The signature is unique for each sender
A message is compressed before it is signed
Encrypt messages using the sender's private key
Have reliable public keys
Create a signature for a message
Verifies the signature using the sender's public key
The available algorithms do not require very large amounts of computational power and do not take large amounts of time.
Use certificates issued by a certification authority

Bibliography

Iso/iec 27000:2012 information technology – security techniques – information security management systems – overview and vocabulary. *Online: http://www.27000.org/*, 2012.

Alberts, Christopher J and Dorofee, Audrey. *Managing information security risks: the OCTAVE approach.* Addison-Wesley Longman Publishing Co., Inc., 2002.

Alexander, Christopher; Ishikawa, Sara, and Silverstein, Murray. *A pattern language: towns, buildings, construction*, volume 2. Oxford University Press, 1977.

Ali, Raian; Dalpiaz, Fabiano, and Giorgini, Paolo. A goal-based framework for contextual requirements modeling and analysis. *Requirements Engineering*, 15(4):439–458, 2010.

Alrajeh, Dalal; Kramer, Jeff; Russo, Alessandra, and Uchitel, Sebastin. Learning operational requirements from goal models. In *Proceedings of the 31st International Conference on Software Engineering*, pages 265–275, 2009.

Altuhhova, Olga; Matulevičius, Raimundas, and Ahmed, Naved. Towards definition of secure business processes. In *Advanced Information Systems Engineering Workshops*, pages 1–15. Springer, 2012.

Araujo, Ivan and Weiss, Michael. Linking patterns and non-functional requirements. In *Proceedings of the Ninth Conference on Pattern Language of Programs (PLOP 2002), September 8-12, 2002*, 2002.

Asnar, Yudis; Massacci, Fabio; Saidane, Ayda; Riccucci, Carlo; Felici, Massimo; Tedeschi, Alessandra; El-Khoury, Paul; Li, Keqin; Séguran, Magali, and Zannone, Nicola. Organizational patterns for security and dependability: From design to application. *Int. J. Secur. Softw. Eng.*, 2(3):1–22, 2011a.

Asnar, Yudistira; Li, Tong; Massacci, Fabio, and Paci, Federica. Computer aided threat identification. In *Commerce and Enterprise Computing (CEC), 2011 IEEE 13th Conference on*, pages 145–152. IEEE, 2011b.

Atzeni, Andrea; Cameroni, Cesare; Faily, Shamal; Lyle, John, and Fléchais, Ivan. Here's johnny: A methodology for developing attacker personas. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 722–727. IEEE, 2011.

Barnum, Sean and Sethi, Amit. Attack patterns as a knowledge resource for building secure software. In *OMG Software Assurance Workshop: Cigital*, 2007.

Beckers, Kristian; Krautsevich, Leanid, and Yautsiukhin, Artsiom. Analysis of social engineering threats with attack graphs. In *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*, pages 216–232. Springer, 2015.

Bozic, Josip and Wotawa, Franz. Security testing based on attack patterns. In *Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on*, pages 4–11. IEEE, 2014.

Bresciani, Paolo; Perini, Anna; Giorgini, Paolo; Giunchiglia, Fausto, and Mylopoulos, John. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

Buschmann, Frank; Meunier, Regine; Rohnert, Hans; Sommerlad, Peter, and Stal, Michael. *A system of patterns: Pattern-oriented software architecture.* Wiley New York, 1996.

Buschmann, Frank; Henney, Kelvin, and Schimdt, Douglas. *Pattern-oriented Software Architecture: On Patterns and Pattern Language*, volume 5. John Wiley & Sons, 2007.

Carpenter, M; Goodspeed, T; Singletary, B; Skoudis, E, and Wright, J. Advanced metering infrastructure attack methodology. *InGuardians white paper*, 2009.

Casagrande, Erik; Woldeamlak, Selamawit; Woon, Wei Lee; Zeineldin, Hatem H, and Svetinovic, Davor. Nlp-kaos for systems goal elicitation: smart metering system case study. *Software Engineering, IEEE Transactions on*, 40(10):941–956, 2014.

Castro, Jaelson; Kolp, Manuel, and Mylopoulos, John. A requirements-driven development methodology. In *Advanced Information Systems Engineering*, pages 108–123. Springer, 2001.

Castro, Jaelson; Kolp, Manuel, and Mylopoulos, John. Towards requirements-driven information systems engineering: the tropos project. *Information systems*, 27(6):365–389, 2002.

Chen, Peter; Dean, Marjon; Ojoko-Adams, Don; Osman, Hassan, and Lopez, Lillian. Systems quality requirements engineering (square) methodology: Case study on asset management system. Technical report, DTIC Document, 2004.

Chung, Lawrence. Representation and utilization of non-functional requirements for information system design. In *Advanced Information Systems Engineering*, pages 5–30. Springer, 1991.

Chung, Lawrence. Dealing with security requirements during the development of information systems. In *Advanced Information Systems Engineering*, volume 685 of *LNCS*, pages 234–251. Springer Berlin Heidelberg, 1993.

Chung, Lawrence and Supakkul, Sam. Representing nfrs and frs: A goal-oriented and use case driven approach. In Dosch, Walter; Lee, RogerY., and Wu, Chisu, editors, *Software Engineering Research and Applications*, volume 3647 of *LNCS*, pages 29–41. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-32133-0.

Committee, IEEE Computer Society. Software Engineering Standards and Board, IEEE-SA Standards. Ieee recommended practice for software requirements specifications. Institute of Electrical and Electronics Engineers, 1993.

Coplien, James O. *Software patterns.* SIGS, 1996.

Coplien, James O and Harrison, Neil B. Organizational patterns of agile software development. 2004.

Crook, Robert; Ince, Darrel; Lin, Luncheng, and Nuseibeh, Bashar. Security requirements engineering: When anti-requirements hit the fan. In *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*, pages 203–205. IEEE, 2002.

Cuellar, Jorge and Suppan, Santiago. A smart metering scenario. Network of Excellence on Engineering Secure Future Internet Software Services and Systems, eRISE 2013, 2013.

Cui, Xiaofeng and Paige, R. An integrated framework for system/software requirements development aligning with business motivations. In *Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on*, pages 547–552, May 2012. doi: 10.1109/ICIS.2012.32.

Dalpiaz, Fabiano; Paja, Elda, and Giorgini, Paolo. Security requirements engineering via commitments. *2011 1st Workshop on Socio-Technical Aspects in Security and Trust (STAST)*, pages 1–8, September 2011.

Dalpiaz, Fabiano; Souza, Vítor E Silva, and Mylopoulos, John. The many faces of operationalization in goal-oriented requirements engineering. In *Proceedings of the Tenth Asia-Pacific Conference on Conceptual Modelling-Volume 154*, pages 3–7, 2014.

Dardenne, Anne; Van Lamsweerde, Axel, and Fickas, Stephen. Goal-directed requirements acquisition. *Science of computer programming*, 20(1):3–50, 1993.

Darimont, Robert and Van Lamsweerde, Axel. Formal refinement patterns for goal-driven requirements elaboration. *ACM SIGSOFT Software Engineering Notes*, 21(6):179–190, 1996.

De Landtsheer, Renaud and Van Lamsweerde, Axel. Reasoning about confidentiality at requirements engineering time. In *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 41–49. ACM, 2005.

de Sousa, Geórgia Maria C; da Silva, Ismênia GL, and de Castro, Jaelson Brelaz. Adapting the nfr framework to aspect-oriented requirements engineering. In *Proceeding of XVII Brazilian Symposium on Software Engineering*, pages 83–98, 2003.

Dubois, Éric; Heymans, Patrick; Mayer, Nicolas, and Matulevičius, Raimundas. A systematic approach to define the domain of information system security risk management. In *Intentional Perspectives on Information Systems Engineering*, pages 289–306. Springer, 2010.

Easterbrook, Steve; Singer, Janice; Storey, Margaret-Anne, and Damian, Daniela. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008.

Eiter, Thomas; Gottlob, Georg, and Mannila, Heikki. Disjunctive datalog. *ACM Transactions on Database Systems (TODS)*, 22(3):364–418, 1997.

Elahi, Golnaz and Yu, Eric. A goal oriented approach for modeling and analyzing security trade-offs. In *Conceptual Modeling-ER 2007*, pages 375–390. Springer, 2007.

Elahi, Golnaz; Yu, Eric, and Zannone, Nicola. A modeling ontology for integrating vulnerabilities into security requirements conceptual foundations. In *Conceptual Modeling-ER 2009*, pages 99–114. Springer, 2009.

Elahi, Golnaz; Yu, Eric, and Zannone, Nicola. A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities. *Requirements Engineering*, 15(1):41–62, 2010.

Elahi, Golnaz; Yu, Eric; Li, Tong, and Liu, Lin. Security requirements engineering in the wild: A survey of common practices. In *Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual*, pages 314–319. IEEE, 2011.

Engebretson, Patrick Henry and Pauli, Joshua J. Leveraging parent mitigations and threats for capec-driven hierarchies. In *Sixth International Conference on Information Technology: New Generations, 2009. ITNG'09.*, pages 344–349, 2009.

Estrada, Hugo; Rebollar, Alicia Martínez; Pastor, Oscar, and Mylopoulos, John. An empirical evaluation of the i* framework in a model-based software generation environment. In *Advanced Information Systems Engineering*, pages 513–527. Springer, 2006.

Fabian, Benjamin; Gürses, Seda; Heisel, Maritta; Santen, Thomas, and Schmidt, Holger. A comparison of security requirements engineering methods. *Requirements engineering*, 15(1):7–40, 2010.

Fernandez, Eduardo; Pelaez, Juan, and Larrondo-Petrie, Maria. Attack patterns: A new forensic and design tool. In *Advances in digital forensics III*, pages 345–357. Springer, 2007a.

Fernandez, Eduardo B; Ballesteros, Jose; Desouza-Doucet, Ana C, and Larrondo-Petrie, Maria M. Security patterns for physical access control systems. In *Data and applications security XXI*, pages 259–274. Springer, 2007b.

Fernandez, Eduardo B; Washizaki, Hironori, and Yoshioka, Nobukazu. Abstract security patterns. In *Proceedings of the 15th Conference on Pattern Languages of Programs*, pages 41–42. ACM, 2008.

Fernandez, Eduardo B; Yoshioka, Nobukazu, and Washizaki, Hironori. Modeling misuse patterns. In *2009 International Conference on Availability, Reliability and Security*, pages 566–571. IEEE, 2009.

Fernandez-Buglioni, Eduardo. *Security patterns in practice: designing secure architectures using software patterns.* John Wiley & Sons, 2013.

Firesmith, Donald. Engineering security requirements. *Journal of Object Technology*, 2(1):53–68, 2003a.

Firesmith, Donald. Specifying reusable security requirements. *Journal of Object Technology*, 3(1):61–75, 2004.

Firesmith, Donald G. Security use cases. *Journal of object technology*, 2(3), 2003b.

Flick, Tony and Morehouse, Justin. *Securing the smart grid: next generation power grid security.* Elsevier, 2010.

Gamma, Erich; Helm, Richard; Johnson, Ralph, and Vlissides, John. Design patterns: Elements of reusable object-oriented software, 1994.

Gegick, Michael and Williams, Laurie. Matching attack patterns to security vulnerabilities in software-intensive system designs. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–7, 2005.

Giorgini, Paolo; Massacci, Fabio; Mylopoulos, John, and Zannone, Nicola. Modeling security requirements through ownership, permission and delegation. In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, pages 167–176. IEEE, 2005a.

Giorgini, Paolo; Massacci, Fabio, and Zannone, Nicola. Security and trust requirements engineering. In *Foundations of Security Analysis and Design III*, volume 3655 of *LNCS*, pages 237–272. Springer Berlin Heidelberg, 2005b.

Giorgini, Paolo; Mouratidis, Haralambos, and Zannone, Nicola. Modelling security and trust with secure tropos. *Integrating Security and Software Engineering: Advances and Future Vision*, pages 160–189, 2006.

Gross, Daniel and Yu, Eric. From non-functional requirements to design through patterns. *Requirements Engineering*, 6(1):18–36, 2001. ISSN 0947-3602.

Group, Object Management. Notation (bpmn) version 2.0. *OMG Specification, Object Management Group*, 2011.

Gunawan, Linda Ariani; Herrmann, Peter, and Kraemer, Frank Alexander. Towards the integration of security aspects into system development using collaboration-oriented models. In *Security Technology*, pages 72–85. Springer, 2009.

Hafiz, Munawar and Johnson, Ralph E. Security patterns and their classification schemes. *University of Illinois at Urbana-Champaign Department of Computer Science, Tech. Rep*, 2006.

Hafiz, Munawar; Adamczyk, Paul, and Johnson, Ralph E. Organizing security patterns. *IEEE Software*, 24(4): 52–60, 2007.

Hafiz, Munawar; Adamczyk, Paul, and Johnson, Ralph E. Growing a pattern language (for security). In *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software*, pages 139–158. ACM, 2012.

Haley, Charles B; Laney, Robin C, and Nuseibeh, Bashar. Deriving security requirements from crosscutting threat descriptions. In *Proceedings of the 3rd international conference on Aspect-oriented software development*, pages 112–121. ACM, 2004.

Haley, Charles B; Laney, Robin; Moffett, Jonathan D, and Nuseibeh, Bashar. Security requirements engineering: A framework for representation and analysis. *Software Engineering, IEEE Transactions on*, 34(1):133–153, 2008.

Halleux, Pierre; Mathieu, Ludovic, and Andersson, Birger. A method to support the alignment of business models and goal models. *Proceedings of BUSITAL*, 8:121, 2008.

Haren, Van. *TOGAF Version 9.1.* Van Haren Publishing, 2011.

Hatebur, Denis; Heisel, Maritta, and Schmidt, Holger. Security engineering using problem frames. In *Emerging Trends in Information and Communication Security*, pages 238–253. Springer, 2006.

Hatebur, Denis; Heisel, Maritta, and Schmidt, Holger. A security engineering process based on patterns. In *Database and Expert Systems Applications, 2007. DEXA'07. 18th International Workshop on*, pages 734–738. IEEE, 2007.

Hernan, Shawn; Lambert, Scott; Ostwald, Tomasz, and Shostack, Adam. Threat modeling-uncover security design flaws using the stride approach. *MSDN Magazine-Louisville*, pages 68–75, 2006.

Herrmann, Gaby and Pernul, Gunther. Towards security semantics in workflow management. In *System Sciences, 1998., Proceedings of the Thirty-First Hawaii International Conference on*, volume 7, pages 766–767. IEEE, 1998.

Herrmann, Peter and Herrmann, Gaby. Security requirement analysis of business processes. *Electronic Commerce Research*, 6(3-4):305–335, 2006. ISSN 1389-5753.

Heyman, Thomas; Yskout, Koen; Scandariato, Riccardo, and Joosen, Wouter. An analysis of the security patterns landscape. In *Proceedings of the Third International Workshop on Software Engineering for Secure Systems (SESS)*, pages 3–10. IEEE Computer Society, 2007.

Heyman, Thomas; Yskout, Koen; Scandariato, Riccardo; Schmidt, Holger, and Yu, Yijun. The security twin peaks. In *Engineering Secure Software and Systems*, pages 167–180. Springer, 2011.

Hoo, Kevin Soo; Sudbury, Andrew W, and Jaquith, Andrew R. Tangible roi through secure software engineering. *Secure Business Quarterly*, 1(2):Q4, 2001.

Horkoff, Jennifer and Yu, Eric. Finding solutions in goal models: an interactive backward reasoning approach. In *Conceptual Modeling–ER 2010*, pages 59–75. Springer, 2010.

Horkoff, Jennifer and Yu, Eric. Analyzing goal models: different approaches and how to choose among them. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 675–682. ACM, 2011.

Horkoff, Jennifer and Yu, Eric. Comparison and evaluation of goal-oriented satisfaction analysis techniques. *Requirements Engineering*, 18(3):199–222, 2013.

Horkoff, Jennifer; Aydemir, Fatma Başak; Li, Feng-Lin; Li, Tong, and Mylopoulos, John. Evaluating modeling languages: An example from the requirements domain. In *Conceptual Modeling (ER 2014)*, pages 260–274. Springer International Publishing, 2014a.

Horkoff, Jennifer; Li, Tong; Li, Feng-Lin; Salnitri, Mattia; Cardoso, Evellin; Giorgini, Paolo; Mylopoulos, John, and Pimentel, João. Taking goal models downstream: A systematic roadmap. In *Research Challenges in Information Science (RCIS), 2014 IEEE Eighth International Conference on*, pages 1–12. IEEE, 2014b.

Horkoff, Jennifer; Li, Tong; Li, Feng-Lin; Salnitri, Mattia; Cardoso, Evellin; Giorgini, Paolo, and Mylopoulos, John. Using goal models downstream: A systematic roadmap and literature review. *International Journal of Information System Modeling and Design (IJISMD)*, 6(2):1–42, 2015.

ISO, ISO and Std, IEC. Iso 27005: 2011. *Information technology–Security techniques–Information security risk management. ISO*, 2011.

Ito, Yurina; Washizaki, Hironori; Yoshizawa, Masatoshi; Fukazawa, Yoshiaki; Okubo, Takao; Kaiya, Haruhiko; Hazeyama, Atsuo; Yoshioka, Nobukazu, and Fernandez, Eduardo B. Systematic mapping of security patterns research. In *Proceedings of the 22nd Conference on Pattern Languages of Programs Conference 2015 (PoLP 2015)*, 2015.

Jackson, Michael. *Problem frames: analysing and structuring software development problems.* Addison-Wesley, 2001.

Jacobson, Ivar; Booch, Grady; Rumbaugh, James; Rumbaugh, James, and Booch, Grady. *The unified software development process*, volume 1. Addison-wesley Reading, 1999.

Jureta, I.J.; Borgida, A.; Ernst, N.A., and Mylopoulos, J. Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling. In *Proc. of RE'10*, pages 115–124, 2010.

Jureta, Ivan J; Mylopoulos, John, and Faulkner, Stephane. Revisiting the core ontology and problem in requirements engineering. In *International Requirements Engineering, 2008. RE'08. 16th IEEE*, pages 71–80. IEEE, 2008.

Jürjens, Jan. Umlsec: Extending uml for secure systems development. In *≪UML≫2002—The Unified Modeling Language*, pages 412–425. Springer, 2002.

Jürjens, Jan. *Secure systems development with UML.* Springer Science & Business Media, 2005.

Kaiya, Haruhiko; Kono, Sho; Ogata, Shinpei; Okubo, Takao; Yoshioka, Nobukazu; Washizaki, Hironori, and Kaijiri, Kenji. Security requirements analysis using knowledge in capec. In *Advanced Information Systems Engineering Workshops*, pages 343–348. Springer, 2014.

Kim, Ji-Yeon and Kim, Hyung-Jong. Defining security primitives for eliciting flexible attack scenarios through capec analysis. In *Information Security Applications*, pages 370–382. Springer, 2014.

Knuth, Donald Ervin. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1998.

Kruchten, Philippe. *The rational unified process: an introduction.* Addison-Wesley Professional, 2004.

Lamsweerde, Axel Van. Elaborating security requirements by construction of intentional anti-models. In *Proceedings of the 26th International Conference on Software Engineering*, pages 148–157. IEEE Computer Society, 2004.

Lankhorst, Marc M; Proper, Henderik Alex, and Jonkers, Henk. The architecture of the archimate language. In *Enterprise, Business-Process and Information Systems Modeling*, pages 367–380. Springer, 2009.

Lethbridge, Timothy C; Sim, Susan Elliott, and Singer, Janice. Studying software engineers: Data collection techniques for software field studies. *Empirical software engineering*, 10(3):311–341, 2005.

Letier, Emmanuel and van Lamsweerde, Axel. Deriving operational software specifications from system goals. In *Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 119–128, 2002. doi: 10.1145/587051.587070.

Li, Jin-Biao; Li, Tong, and Liu, Lin. Chinese requirements analysis based on class diagram semantics. *Acta Electronica Sinica*, pages 94–98, 2011.

Li, Tong and Horkoff, Jennifer. Dealing with security requirements for socio-technical systems: A holistic approach. In *Advanced Information Systems Engineering (CAiSE 2014)*, pages 185–200. Springer International Publishing, 2014.

Li, Tong and Mylopoulos, John. Modeling and applying security patterns using contextual goal models. In *The 7th International i* Workshop (iStar14)*, pages 208–223, 2014.

Li, Tong; Horkoff, Jennifer, and Mylopoulos, John. Integrating security patterns with security requirements analysis using contextual goal models. In *The Practice of Enterprise Modeling (PoEM 2014)*, pages 208–223. Springer Berlin Heidelberg, 2014a.

Li, Tong; Horkoff, Jennifer, and Mylopoulos, John. A prototype tool for modeling and analyzing security requirements from a holistic viewpoint. In *The CAiSE'14 Forum at the 26th International Conference on Advanced Information Systems Engineering*, pages 185–192, 2014b.

Li, Tong; Horkoff, Jennifer; Beckers, Kristian; Paja, Elda, and Mylopoulos, John. A holistic approach to security attack modeling and analysis. In *Proceedings of the Eighth International i* Workshop*, pages 49–54, 2015a.

Li, Tong; Horkoff, Jennifer, and Mylopoulos, John. Analyzing and enforcing security mechanisms on requirements specification. In *Requirements Engineering: Foundation for Software Quality (REFSQ 2015)*. Springer International Publishing, 2015b.

Li, Tong; Horkoff, Jennifer; Paja, Elda; Beckers, Kristian, and Mylopoulos, John. Analyzing attack strategies through anti-goal refinement. In *The Practice of Enterprise Modeling (PoEM 2015)*, pages 75–90. Springer International Publishing, 2015c.

Li, Tong; Paja, Elda; Mylopoulos, John; Horkoff, Jennifer, and Beckers, Kristian. Holistic security requirements analysis: An attacker's perspective. In *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*, pages 282–283. IEEE, 2015d.

Li, Tong; Horkoff, Jennifer; Paja, Elda; Beckers, Kristian, and Mylopoulos, John. Security attack analysis using attack patterns. In *The IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*. IEEE, 2016.

Lin, Lun-Cheng; Nuseibeh, Bashar; Ince, Daniel; Jackson, Michael, and Moffett, Jonathan. Analysing security threats and vulnerabilities using abuse frames. *ETAPS-04*, 2003a.

Lin, Luncheng; Nuseibeh, Bashar; Ince, Darrel; Jackson, Michael, and Moffett, Jonathan. Introducing abuse frames for analysing security requirements. In *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*, pages 371–372. IEEE, 2003b.

Liu, Lin; Yu, E, and Mylopoulos, John. Security and privacy requirements analysis within a social setting. In *Proc. of RE'03*, volume 3, pages 151–161, Monterey, California, 2003.

Liu, Lin; Yu, Eric S. K., and Mylopoulos, John. Secure-i*: Engineering secure software systems through social analysis. *Int. J. Software and Informatics*, 3(1):89–120, 2009.

Lodderstedt, Torsten; Basin, David, and Doser, Jürgen. Secureuml: A uml-based modeling language for model-driven security. In *≪UML≫2002—The Unified Modeling Language*, pages 426–441. Springer, 2002.

Lund, Mass Soldal; Solhaug, Bjørnar, and Stølen, Ketil. *Model-driven risk analysis: the CORAS approach.* Springer Science & Business Media, 2010.

Mansourov, Nikolai and Campara, Djenana. *System assurance: beyond detecting vulnerabilities.* Elsevier, 2010.

Massacci, Fabio and Paci, Federica. How to select a security requirements method? a comparative study with students and practitioners. In *Secure IT Systems*, pages 89–104. Springer, 2012.

Massacci, Fabio; Mylopoulos, John, and Zannone, Nicola. Security requirements engineering: the si* modeling language and the secure tropos methodology. In *Advances in Intelligent Information Systems*, pages 147–174. Springer, 2010.

Matulevičius, Raimundas; Mayer, Nicolas; Mouratidis, Haralambos; Dubois, Eric; Heymans, Patrick, and Genon, Nicolas. Adapting secure tropos for security risk management in the early phases of information systems development. In *Advanced Information Systems Engineering*, pages 541–555. Springer, 2008.

May, Jeffrey and Dhillon, Gaurpreet. A holistic approach for enriching information security analysis and security policy formation. In *European Conference on Information Systems (ECIS)*, 2010.

McDermott, J. and Fox, C. Using abuse case models for security requirements analysis. In *15th Annual Computer Security Applications Conference, (ACSAC'99) Proceedings.*, pages 55–64. IEEE, 1999.

Mead, Nancy R. Experiences in eliciting security requirements. Technical report, DTIC Document, 2006a.

Mead, Nancy R and Stehney, Ted. Security quality requirements engineering (square) methodology. *SIGSOFT Softw. Eng. Notes*, 30(4), 2005.

Mead, Nancy R; Viswanathan, Venkatesh, and Zhan, Justin. Incorporating security requirements engineering into the rational unified process. In *2008 International Conference on Information Security and Assurance*, pages 537–542. IEEE, 2008.

Mead, NR. Identifying security requirements using the security quality requirements engineering (square) method. *Integrating Security and Software Engineering*, pages 44–69, 2006b.

Meland, Per Håkon and Gjære, Erlend Andreas. Representing threats in bpmn 2.0. In *Availability, Reliability and Security (ARES), 2012 Seventh International Conference on*, pages 542–550. IEEE, 2012.

Mellado, Daniel; Fernández-Medina, Eduardo, and Piattini, Mario. Applying a security requirements engineering process. In *Computer Security–ESORICS 2006*, pages 192–206. Springer, 2006.

Mellado, Daniel; Fernández-Medina, Eduardo, and Piattini, Mario. A common criteria based security requirements engineering process for the development of secure information systems. *Computer standards & interfaces*, 29 (2):244–253, 2007.

Menzel, Michael; Thomas, Ivonne, and Meinel, Christoph. Security requirements specification in service-oriented business process management. In *Proceedings of International Conference on Availability, Reliability and Security, 2009. ARES'09*, pages 41–48. IEEE, 2009.

Mitnick, Kevin D and Simon, William L. *The Art of Intrusion: The real stories behind the exploits of hackers, intruders and deceivers*. John Wiley & Sons, 2005.

Mitnick, Kevin D and Simon, William L. *The art of deception: Controlling the human element of security*. John Wiley & Sons, 2011.

MITRE-CVE, . Common vulnerabilities and exposures. URL `https://cve.mitre.org/`.

MITRE-CWE, . Common weakness enumeration. URL `https://cwe.mitre.org/`.

Moore, Andrew P; Ellison, Robert J, and Linger, Richard C. Attack modeling for information security and survivability. Technical report, CMU-SEI-2001-TN-001., 2001.

Morais, Anderson; Hwang, Iksoon; Cavalli, Ana, and Martins, Eliane. Generating attack scenarios for the system security validation. *Networking science*, 2(3-4):69–80, 2013.

Mouratidis, Haralambos. Secure software systems engineering: the secure tropos approach. *Journal of Software*, 6(3):331–339, 2011.

Mouratidis, Haralambos and Giorgini, Paolo. A natural extension of tropos methodology for modelling security. In *Proc. of the Agent Oriented Methodologies Workshop (OOPSLA 2002)*. Citeseer, 2002.

Mouratidis, Haralambos and Giorgini, Paolo. Secure tropos: a security-oriented extension of the tropos methodology. *International Journal of Software Engineering and Knowledge Engineering*, 17(02):285–309, 2007a.

Mouratidis, Haralambos and Giorgini, Paolo. Security attack testing (sat) - testing the security of information systems at design time. *Information Systems*, 32(8):1166–1183, 2007b.

Mouratidis, Haralambos and Jurjens, Jan. From goal-driven security requirements engineering to secure design. *International Journal of Intelligent System*, 25(8):813–840, 2010.

Mouratidis, Haralambos; Giorgini, Paolo, and Manson, Gordon. Using security attack scenarios to analyse security during information systems. In *Proceedings of the International Conference on Enterprise Information Systems*, pages 10–17, 2004.

Mouratidis, Haralambos; Weiss, Michael, and Giorgini, Paolo. Modeling secure systems using an agent-oriented approach and security patterns. *International Journal of Software Engineering and Knowledge Engineering*, 16(3):471, 2006.

Mouratidis, Haralambos; Islam, Shareeful; Kalloniatis, Christos, and Gritzalis, Stefanos. A framework to support selection of cloud providers based on security and privacy requirements. *Journal of Systems and Software*, 86 (9):2276–2293, 2013.

Mowbray, Thomas J and Malveau, Raphael C. *CORBA design patterns*. John Wiley & Sons, Inc., 1997.

Mussbacher, Gunter; Weiss, Michael, and Amyot, Daniel. Formalizing architectural patterns with the goal-oriented requirement language. In *Nordic Pattern Languages of Programs Conference*, 2006.

Mylopoulos, John; Chung, Lawrence, and Nixon, Brian. Representing and using nonfunctional requirements: A process-oriented approach. *Software Engineering, IEEE Transactions on*, 18(6):483–497, 1992.

Nhlabatsi, Armstrong; Nuseibeh, Bashar, and Yu, Yijun. Security requirements engineering for evolving software systems: A survey. *IJSSE*, 1(1):54–73, 2010.

NIST, . Roadmap for smart grid interoperability standards, release 2.0. *NIST special publication*, 1108R2, 2012.

Niu, Nan and Easterbrook, Steve. So, you think you know others' goals? a repertory grid study. *Software, IEEE*, 24(2):53–61, 2007.

Nuseibeh, Bashar. Weaving together requirements and architectures. *Computer*, 34(3):115–119, 2001.

Okubo, Takao; Taguchi, Kenji, and Yoshioka, Nobukazu. Misuse cases+ assets+ security goals. In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, volume 3, pages 424–429. IEEE, 2009.

Okubo, Takao; Kaiya, Haruhiko, and Yoshioka, Nobukazu. Mutual refinement of security requirements and architecture using twin peaks model. In *Computer Software and Applications Conference Workshops (COMPSACW)*, pages 367–372. IEEE, 2012.

Oladimeji, Ebenezer; Supakkul, Sam, and Chung, Lawrence. Security threat modeling and analysis: A goal-oriented approach. In *Proc. of the 10th IASTED International Conference on Software Engineering and Applications (SEA 2006)*, pages 13–15. Citeseer, 2006a.

Oladimeji, Ebenezer A; Supakkul, Sam, and Chung, Lawrence. Representing security goals, policies, and objects. In *1st IEEE/ACIS International Workshop on Component-Based Software Engineering*, pages 160–167. IEEE, 2006b.

Ourston, Dirk; Matzner, Sara; Stump, William, and Hopkins, Bryan. Applications of hidden markov models to detecting multi-stage network attacks. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, pages 334–343. IEEE, 2003.

Paja, Elda; Giorgini, Paolo; Paul, Stéphane, and Meland, Per Håkon. Security requirements engineering for secure business processes. In *Workshops on Business Informatics Research*, pages 77–89. Springer, 2012.

Paja, Elda; Dalpiaz, Fabiano, and Giorgini, Paolo. Managing security requirements conflicts in socio-technical systems. In *Conceptual Modeling*, pages 270–283. Springer, 2013.

Paul, Stéphane. Towards automating the construction & maintenance of attack trees: a feasibility study. *arXiv preprint arXiv:1404.1986*, 2014.

Pavlidis, Michalis and Islam, Shareeful. Sectro: A case tool for modelling security in requirements engineering using secure tropos. In *CAiSE Forum*, pages 89–96, 2011.

Phillips, Cynthia and Swiler, Laura Painton. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM, 1998.

Pimentel, João; Lucena, Márcia; Castro, Jaelson; Silva, Carla; Santos, Emanuel, and Alencar, Fernanda. Deriving software architectural models from requirements models for adaptive systems: the stream-a approach. *Requirements Engineering*, 17(4):259–281, 2012.

Ponemon, L. Cost of data breach study: Global analysis. Technical report, Poneomon Institute sponsored by IBM, 2015.

Ranjan, Prabhat and Misra, Arun Kumar. Agent based system development: a domain-specific goal approach. *ACM SIGSOFT Software Engineering Notes*, 31(6):1–6, 2006.

Robin, Kazi Asad. Evaluation of a multilayer security requirements analysis framework. Master's thesis, University of Trento, 2015.

Robinson, William N; Pawlowski, Suzanne D, and Volkov, Vecheslav. Requirements interaction management. *ACM Computing Surveys (CSUR)*, 35(2):132–190, 2003.

Rodríguez, Alfonso; Fernández-Medina, Eduardo, and Piattini, Mario. A bpmn extension for the modeling of security requirements in business processes. *IEICE transactions on information and systems*, 90(4):745–752, 2007a.

Rodríguez, Alfonso; Fernández-Medina, Eduardo, and Piattini, Mario. M-bpsec: a method for security requirement elicitation from a uml 2.0 business process specification. In *Advances in Conceptual Modeling–Foundations and Applications*, pages 106–115. Springer, 2007b.

Rodríguez, Alfonso; de Guzmán, Ignacio García-Rodríguez; Fernández-Medina, Eduardo, and Piattini, Mario. Semi-formal transformation of secure business processes into analysis class and use case models: An mda approach. *Information and Software Technology*, 52(9):945 – 971, 2010.

Rodríguez, Alfonso; Fernández-Medina, Eduardo; Trujillo, Juan, and Piattini, Mario. Secure business process model specification through a uml 2.0 activity diagram profile. *Decision Support Systems*, 51(3):446–465, 2011.

Runeson, Per and Höst, Martin. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131–164, 2009.

Salini, P and Kanmani, S. Security requirements engineering process for web applications. *Procedia Engineering*, 38:2799–2807, 2012.

Salini, P and Kanmani, S. Effectiveness and performance analysis of model-oriented security requirements engineering to elicit security requirements: a systematic solution for developing secure software systems. *International Journal of Information Security*, pages 1–16, 2015.

Salnitri, Mattia; Dalpiaz, Fabiano, and Giorgini, Paolo. Modeling and verifying security policies in business processes. In *Enterprise, Business-Process and Information Systems Modeling*, pages 200–214. Springer, 2014a.

Salnitri, Mattia; Paja, Elda, and Giorgini, Paolo. Preserving compliance with security requirements in socio-technical systems. In *Cyber Security and Privacy*, pages 49–61. Springer, 2014b.

Salnitri, Mattia; Dalpiaz, Fabiano, and Giorgini, Paolo. Designing secure business processes with secbpmn. *Software & Systems Modeling*, pages 1–21, 2015.

Sánchez-Cid, Francisco and Maña, Antonio. Serenity pattern-based software development life-cycle. In *19th International Workshop on Database and Expert Systems Application, 2008*, pages 305–309, Sept 2008.

Sandhu, Ravi. Good-enough security: Toward a pragmatic business-driven discipline. *IEEE Internet Computing*, 7(1):66–68, 2003.

Scandariato, Riccardo; Wuyts, Kim, and Joosen, Wouter. A descriptive study of microsoft's threat modeling technique. *Requirements Engineering*, 20(2):163–180.

Scandariato, Riccardo; Yskout, Koen; Heyman, Thomas, and Joosen, Wouter. Architecting software with security patterns. Technical report, KU Leuven, 2008.

Schmidt, Holger. Threat-and risk-analysis during early security requirements engineering. In *ARES*, pages 188–195, 2010.

Schneier, B. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999.

Schumacher, Markus; Fernandez-Buglioni, Eduardo, and Hybertson, Duane. *Security Patterns: Integrating Security and Systems Engineering.* John Wiley & Sons, 2006.

Serenity-Consortium, . Smart items–medical emergency response. Technical report, March 2007.

Sheyner, Oleg; Haines, Joshua; Jha, Somesh; Lippmann, Richard, and Wing, Jeannette M. Automated generation and analysis of attack graphs. In *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 273–284. IEEE, 2002.

Shiroma, Y.; Washizaki, H.; Fukazawa, Y.; Kubo, A, and Yoshioka, N. Model-driven security patterns application based on dependences among patterns. In *International Conference on Availability, Reliability, and Security, 2010*, pages 555–559, Feb 2010. doi: 10.1109/ARES.2010.103.

Shostack, Adam. *Threat Modeling: Designing for Security.* John Wiley & Sons, 2014.

Sindre, Guttorm and Opdahl, Andreas L. Capturing security requirements through misuse cases. In *Norsk Informatikkonferanse, NIK 2001*, 2001.

Sindre, Guttorm and Opdahl, Andreas L. Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1):34–44, 2005.

Sindre, Guttorm; Firesmith, Donald G, and Opdahl, Andreas L. A reuse-based approach to determining security requirements. In *Proceedings of the 9th international workshop on requirements engineering: foundation for software quality (REFSQ'03), Klagenfurt, Austria.* Citeseer, 2003.

Skoudis, Ed and Liston, Tom. *Counter hack reloaded: a step-by-step guide to computer attacks and effective defenses.* Prentice Hall Press, 2005.

Souag, Amina; Mazo, Raúl; Salinesi, Camille, and Comyn-Wattiau, Isabelle. Reusable knowledge in security requirements engineering: a systematic mapping study. *Requirements Engineering*, pages 1–33, 2015.

Spears, Janine L. A holistic risk analysis method for identifying information security risks. In *Security Management, Integrity, and Internal Control in Information Systems*, pages 185–202. Springer, 2005.

Steele, Adam and Jia, Xiaoping. Adversary centered design: Threat modeling using anti-scenarios, anti-use cases and anti-personas. In *Proc. of Information and Knowledge Engineering, IKE'08*, pages 367–370, 2008.

Suleiman, Husam and Svetinovic, Davor. Evaluating the effectiveness of the security quality requirements engineering (square) method: a case study using smart grid advanced metering infrastructure. *Requirements Engineering*, 18(3):251–279, 2013.

Supaporn, Kawin; Prompoon, Nakornthip, and Rojkangsadan, Thongchai. An approach: Constructing the grammar from security pattern. In *Proc. 4th International Joint Conference on Computer Science and Software Engineering*, 2007.

TOG, . Mapping the togaf adm to the zachman framework, 2002. URL `http://www.opengroup.org/architecture/0210can/togaf8/doc-review/togaf8cr/c/p4/zf/zf_mapping.htm`.

Tzu, Sun. *The art of war*. Shambhala Publications, 2011.

Ustun, Volkan; Yilmaz, Levent, and Smith, Jeffrey S. A conceptual model for agent-based simulation of physical security systems. In *Proceedings of the 44th annual Southeast regional conference*, pages 365–370. ACM, 2006.

Uzunov, Anton V; Fernandez, Eduardo B, and Falkner, Katrina. Engineering security into distributed systems: A survey of methodologies. *J. UCS*, 18(20):2920–3006, 2012.

Uzunov, Anton V; Fernandez, Eduardo B, and Falkner, Katrina. Ase: a comprehensive pattern-driven security methodology for distributed systems. *Computer Standards & Interfaces*, 41:112–137, 2015.

Van Lamsweerde, A. and Letier, E. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*, 26(10):978–1005, Oct 2000.

Van Lamsweerde, Axel and others, . *Software System Reliability and Security*, volume 9, chapter Engineering requirements for system reliability and security, page 196. IOS PRESS, 2007.

VanHilst, Michael; Fernandez, Eduardo B, and Braz, Fabrício. A multi-dimensional classification for users of security patterns. *Journal of Research and Practice in Information Technology*, 41(2):87, 2009.

Wang, Hui; Jia, Zongpu, and Shen, Zihao. Research on security requirements engineering process. In *Industrial Engineering and Engineering Management, 2009. IE&EM'09. 16th International Conference on*, pages 1285–1288. IEEE, 2009.

Ware, Michael S; Bowles, John B, and Eastman, Caroline M. Using the common criteria to elicit security requirements with use cases. In *SoutheastCon, 2006. Proceedings of the IEEE*, pages 273–278. IEEE, 2005.

Wei, Chun. Misuse cases and abuse cases in eliciting security requirements, 2005.

Weingart, Steve H. Physical security devices for computer subsystems: A survey of attacks and defenses. In *Cryptographic Hardware and Embedded Systems — CHES 2000*, pages 302–317. Springer, 2000.

Whittle, Jon; Wijesekera, Duminda, and Hartong, Mark. Executable misuse cases for modeling security concerns. In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*, pages 121–130. IEEE, 2008.

Wieringa, RJ and Heerkens, JMG. The methodological soundness of requirements engineering papers: a conceptual framework and two case studies. *Requirements engineering*, 11(4):295–307, 2006.

Wimmer, Maria and Von Bredow, Bianca. A holistic approach for providing security solutions in e-government. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 1715–1724. IEEE, 2002.

Yoder, Joseph and Barcalow, Jeffrey. Architectural patterns for enabling application security. *Fourth Conference on Patterns Languages of Programs (PLoP'97)*, 1997.

Yskout, Koen; Heyman, Thomas; Scandariato, Riccardo, and Joosen, Wouter. A system of security patterns. Technical report, KU Leuven, 2006.

Yu, Eric. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the Third IEEE International Symposium on Requirement Engineering, 1997*, pages 226–235. IEEE Computer Soc. Press, 1997.

Yu, Yijun; Kaiya, Haruhiko; Washizaki, Hironori; Xiong, Yingfei; Hu, Zhenjiang, and Yoshioka, Nobukazu. Enforcing a security pattern in stakeholder goal models. In *Proceedings of the 4th ACM Workshop on Quality of Protection*, pages 9–14, 2008.

Yuan, Xiaohong; Nuakoh, Emmanuel Borkor; Beal, Jodria S, and Yu, Huiming. Retrieving relevant capec attack patterns for secure software development. In *Proceedings of the 9th Annual Cyber and Information Security Research Conference*, pages 33–36. ACM, 2014.

Zachman, John A. A framework for information systems architecture. *IBM systems journal*, 26(3):276–292, 1987.

Zave, Pamela and Jackson, Michael. Four dark corners of requirements engineering. *ACM Trans. Softw. Eng. Methodol.*, 6(1):1–30, 1997. ISSN 1049-331X.

Zuccato, Albin. Holistic security requirement engineering for electronic commerce. *Computers & Security*, 23(1): 63–76, 2004.