Software Updates Strategies: a Quantitative Evaluation against Advanced Persistent Threats

Giorgio Di Tizio *Student Member, IEEE*, Michele Armellini, Fabio Massacci *Member, IEEE*

Abstract—Software updates reduce the opportunity for exploitation. However, since updates can also introduce breaking changes, enterprises face the problem of balancing the need to secure software with updates with the need to support operations. We propose a methodology to quantitatively investigate the effectiveness of software updates strategies against attacks of Advanced Persistent Threats (APTs). We consider strategies where the vendor updates are the only limiting factors to cases in which enterprises delay updates from 1 to 7 months based on SANS data.

Our manually curated dataset of APT attacks covers 86 APTs and 350 campaigns from 2008 to 2020. It includes information about attack vectors, exploited vulnerabilities (e.g. 0-days vs public vulnerabilities), and affected software and versions. Contrary to common belief, most APT campaigns employed publicly known vulnerabilities.

If an enterprise could theoretically update as soon as an update is released, it would face lower odds of being compromised than those waiting one (4.9x) or three (9.1x) months. However, if attacked, it could still be compromised from 14% to 33% of the times.

As in practice enterprises must do regression testing before applying an update, our major finding is that one could perform 12% of all possible updates restricting oneself only to versions fixing publicly known vulnerabilities without significant changes to the odds of being compromised compared to a company that updates for all versions.

Index Terms—Advanced Persistent Threats, software vulnerabilities, software updates

1 Introduction

A recent study [1] shows that it takes more than 200 days for an enterprise to align 90% of their machines with the latest (not known to be vulnerable) software version given the need to perform regression testing [2].

Such behavior is rational because not all vulnerabilities are always exploited in the wild [3], and several authors have determined that the actual risk of slow updates against 'mass attackers' is limited [4], [5] and often due to specific types of vulnerabilities such as those traded in Black Markets [6] or with other predictable characteristics [7], [8]. Hence, risk analysis might be an effective approach when considering 'mass attackers' which might well be 'work averse' and stick to old exploits until they are no longer profitable [9]. However, many companies also face Advanced Persistent Threats (APTs). APTs are highly specialized professionals [10] that use a variety of customized strategies [11], often leveraging on spearphishing [12] and 0-days [13], [14] to maintain a stealthy profile [13]. In this scenario, slow updates do not seem appropriate.

Yet, not all the APTs are really *sophisticated* [15]. Some reports challenged some of these 'allegations' and observed that APTs often reuse tools, malware, and vulnerabilities [12], [16], [17]. These reports are based on threat intelligence data with few overlaps [18] thus capturing only partial information of the APT attacks [19], [15].

These conflicting claims may be due to the lack of a systematic study. Indeed, previous works on APTs analysis [20], [13], [10], [21] mostly reported a *qualitative* analysis of a handful of APTs. However, relying on qualitative estimations is known to produce risk miscategorization and wrong prioritization [22], [23] due to several factors like judgmental biases [23], agenda-setting, and framing [24]. Framing of individual reports can produce a distorted perception of the risk. We lack a broad view of the APT landscape that allows companies to correctly assess the advantages and disadvantages of current approaches to software updates. Data acquisition of APT campaigns, i.e. specific attacks conducted by APT groups, is currently a challenging task. Semi-automated approaches based on report parsing [25] proved to be too riddled with false positives because the associations between APTs and software vulnerabilities (identified by a CVE) are based on the presence of keywords and not on the semantics of the document. Here our research questions are as follows:

RQ1. What are the APTs characteristics that quantitatively describe the landscape of APT campaigns as observable from public reports?

RQ2. Given a quantitative description of both APT campaigns and software updates, how effective are different update strategies to protect against APT campaigns?

We thus make the following contributions:

- We build a structured, manually verified database in Neo4j of 86 APTs and more than 350 campaigns based on an exhaustive search of over 500 technical reports and blogs and up to 22 different resources for each APT. The database [26] is available on Zenodo.
- We present a methodology to quantify and compare the effectiveness and cost of software update strategies on historical data about campaigns.

- *G. Di Tizio (corresponding author) is with University of Trento, Italy. E-mail: giorgio.ditizio@unitn.it*
- *M. Armellini is with University of Trento, Italy.*
- *F. Massacci is with University of Trento, Italy and Vrije Universiteit Amsterdam, The Netherlands.*

- We quantitatively evaluated the effectiveness and cost of different software updates strategies, in terms of the *conditional* probability of being compromised and the number of updates required for 5 widely used software products (*Office*, *Acrobat Reader*, *Air*, *JRE*, and *Flash Player*) for the Windows O.S.

Scope of the work: We provide a quantitative analysis of the risk against APT to allow companies to make rational decisions on software updates. We do not propose new mechanisms to detect and mitigate APTs attacks.

2 The Software Update Problem

If a company could only update for new functionalities, the choice would be obvious: why fixing what is not broken? Yet, companies must update for security reasons too. However, it is not uncommon that vulnerability fixes are merged with features changes in a single update. Every time a new version of a software is published, one can

- *update* immediately;
- *wait* some time (e.g. for regression testing) and update;
- *skip* the update.

This choice may be influenced by asynchronous events related to the *reservation*, *disclosure*, and *exploitation* of software vulnerabilities in the *current* release.

Unfortunately, a company cannot fully decide *in advance* the configuration they will have when hit (or most frequently not hit) by an attacker as it depends on the attacker's choice. A company can only decide on the software updates strategy. To capture what can happen we introduce some terminology.

2.1 Terminology

For each vulnerability we identified five instants of time:

- *Vulnerability Reserved time* (t_{V_r}): when the CVE entry for the vulnerability is reserved by MITRE;
- *Vulnerability Published time* (t_{V_p}): when the CVE for the vulnerability is published in NVD;
- *Vulnerability Exploited time* (t_{V_e}): when the vulnerability is observed to be exploited in the wild;
- *Update release time* (t_{U_r}): when an update that addresses the vulnerability is released.
- *Update deployed time* (t_{U_d}): when an update that addresses the vulnerability is deployed.

Tab. 1 shows how we can classify attack scenarios based on the instant of time t_{V_e} and its relative position with the other events: t_{V_r}, t_{V_p}, and t_{U_r} Fig. 1 summarizes the possible combinations of the different events.

2.2 The Software Update Strategies

To answer *RQ2*, we describe the update strategies, summarized in Tab. 2, for an enterprise based on what was discussed previously: *update*, *wait* and then update, or *skip*. It is important to underline that disabling automated updates is not uncommon in enterprise networks [27], [28]. This is mainly due to compatibility issues between the updated software and internal projects [2], [29] that can produce disruption of the enterprise work. In this case delays are introduced to perform regression testing.

TABLE 1: Classification of Attack Scenarios

Scenario	Description
Unknown-Unknown/ Unpreventable (UU/U)	The vulnerability is exploited before a CVE was reserved, before its public disclosure, and before an update for the vulnerability was released.
Unknown-Unknown/ Preventable (UU/P)	The vulnerability is exploited before a CVE was reserved, before its public disclosure, but after an update for the vulnerability was released.
Known-Unknown/ Unpreventable (KU/U)	The vulnerability is exploited after a CVE was reserved, before its public disclosure, and before an update for the vulnerability was released.
Known-Unknown/ Preventable (KU/P)	The vulnerability is exploited after a CVE was reserved, before its public disclosure, and after an update for the vulnerability was released.
Known-Known/ Unpreventable (KK/U)	The vulnerability is exploited after a CVE was reserved, after its public disclosure, and before an update for the vulnerability was released.
Known-Known/ Preventable (KK/P)	The vulnerability is exploited after a CVE was reserved, after its public disclosure, and after an update for the vulnerability was released.

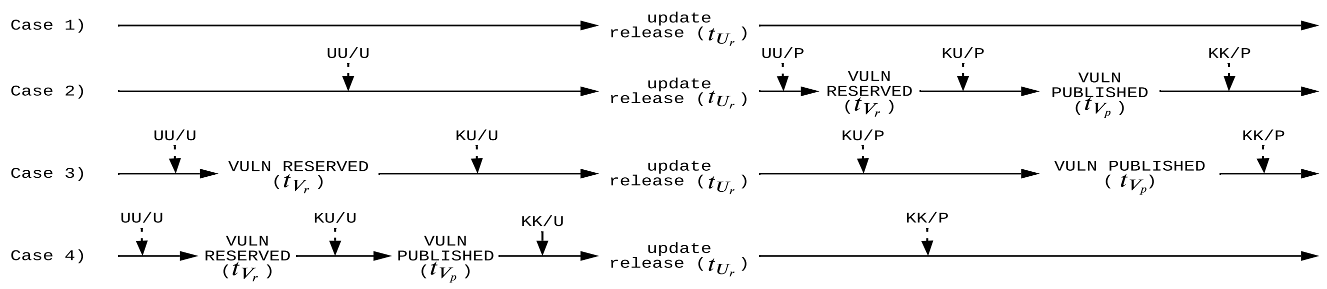
TABLE 2: Update strategies

Each strategy represents an approach for updating the software. The *Immediate* strategy represents the upper bound achievable by an enterprise. The *Planned*, *Reactive*, and *Informed Reactive* are evaluated with different update intervals that represent different level of responsiveness.

Strategy	Update Interval	Description
Immediate	/	Update to each newest version as soon as it is available without any delay
Planned	1, 3, 7 months	Update to each newest version but wait a delay before the deployment
Reactive	1, 3, 7 months	Update to the first new (non vulnerable) version only after the publication in NVD of a CVE, wait a delay before the deployment
Informed Reactive	1, 3, 7 months	Update to the first new (non vulnerable) version only after the reservation by MITRE of a CVE entry, wait a delay before the deployment

We considered the application of a software update with a certain delay, starting from the date on which the strategy bases its decision. We considered different update intervals to determine how the probabilities change if a more responsive approach is employed. We leverage update intervals data from SANS [30] based on a variety of enterprises (Government, Financial Services, Healthcare, and Consulting) and from Kotzias et al. [1] based on 28k enterprises. Tab. 3 shows the update intervals from SANS and maps them to our update strategies.

Immediate strategy: The enterprise updates its software as soon as a new version is available (t_{U_r}) and without delay. If multiple updates are released in the same time interval, the update takes the most recent one. The update is applied even if a vulnerability for the previous version is not present yet. This is the theoretical limit for the enterprise because it is bounded only by the release speed of the vendor. However, this approach is likely impractical because updates require some time to be deployed in an enterprise to not break other functionalities.


Case 1)
update release (t_{U_r})
UU/U
UU/P
KU/P
KK/P
Case 2)
update release (t_{U_r})
VULN RESERVED (t_{V_r})
VULN PUBLISHED (t_{V_p})
UU/U
KU/U
KU/P
KK/P
Case 3)
VULN RESERVED (t_{V_r})
update release (t_{U_r})
VULN PUBLISHED (t_{V_p})
UU/U
KU/U
KK/U
KK/P
Case 4)
VULN RESERVED (t_{V_r})
VULN PUBLISHED (t_{V_p})
update release (t_{U_r})

At the time when a software update is available, we have 4 cases: *Case 1)* there is no reservation and publication of vulnerabilities before and after the release of an update for the current version. In this case, there is no exploitation of the vulnerability. *Case 2)* after a software update is released, a vulnerability is reserved and disclosed for the current version. *Case 3)* before the release of a software update a vulnerability is reserved for the current version, but the disclosure happens after the update release. *Case 4)* the reservation and disclosure of a vulnerability for the current version happen before the release of an update. Different update strategies can be applied but are all constrained by the presence of a new release. The exploitation events (vertical lines) can happen at any instant of time asynchronously from the reservation-disclosure process and the release of updates. They are classified following Tab. 1

Fig. 1: Combinations of vulnerability reservation, disclosure, and exploitation events with the presence of new updates.

TABLE 3: Update Intervals from SANS [30]

Percentage of enterprises that update weekly, monthly, quarterly, or with other delays. We evaluated the strategies with these update intervals. Enterprises that update weekly are comparable to the *Immediate* strategy. We associated 7 months for the update interval of 'Other' from [1].

Update interval	% Enterprises	Update Strategy Correspondence
Weekly	24.9	*Immediate*
Monthly	57.5	*Planned/Reactive/Informed Reactive* within 1 month delay
Quarterly	7.7	*Planned/Reactive/Informed Reactive* within 3 months delay
Other	10.0	*Planned/Reactive/Informed Reactive* within 7 months delay

TABLE 4: State of the Art *on APTs* - Main Research Topics

Research Category	State of the Art	This paper
APTs data sources	[11]	✓
Metrics for TI sources	[19], [18]	
Attackers characteristics	[13]*, [20]*, [12]	✓
Detection of attacks	[31], [32], [33], [34], [35], [36], [14], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46]	
Game Theory	[47], [48], [40]	
Exploitation likelihood	[36], [48]	✓
Analysis of update releases	-	✓

* Performed high level analysis on few APTs campaigns.

Planned strategy: The company updates its software to each new version with a delay from the release date (t_{U_r}). If multiple updates are released in the same time interval, the company takes the most recent one. This delay factors the time for regression testing and update deployment. The delays are taken from Tab. 3. As in the *Immediate* strategy, the update is *not* triggered by the knowledge of vulnerabilities but only on the availability of a new update.

Reactive strategy The enterprise updates the software only *after* the publication of a vulnerability by NVD (t_{V_p}) with a delay taken from Tab. 3. The new version installed is the first non-vulnerable update available at that time.

Informed Reactive strategy The enterprise updates the software only *after* the *reservation* of a vulnerability by MITRE (t_{V_r}). The new version installed is the first non-vulnerable update available at that time. This strategy describes an enterprise that pays an annual subscription fee to get information about the non publicly disclosed vulnerabilities from companies that provide 0-days data information (e.g. Exodus Intelligence, Zerodium). The strategy presents an update interval as the *Reactive* and *Planned* strategies.

3 Related Works

Tab. 4 shows the research categories addressed by the state-of-the-art on APTs. The majority of the research activity focused on the detection of APTs campaigns while few papers tried to characterize their behavior, estimate the risk, and evaluate update strategies from real data.

3.1 APT and Metrics for Threat Intelligence sources

Lemay et al. [11] presented a description of different resources about the activities of more than 40 APTs. Li et al. [19] utilized a set of metrics (Volume, Differential contribution, Exclusive contribution, Latency, Accuracy, and Coverage) to compare different public and private Threat Intelligence (TI) data feeds. They observed that in the majority of the data feeds there is no overlapping of Indicator of Compromise (IOC) and a high number of false positives. Similarly, Bouwman et al. [18] analyzed two paid TI and observed very few overlaps in the indicators for 22 APTs. The distinction is confirmed with a comparison with open TI data. Furthermore, they observed that TI data is employed in the decision process of companies, but there is a lack of metrics to determine the quality of these data. Several works [49], [25], [50] proposed a (semi-)automated approach based on report parsing to generate a database of IOC. However, merely relying on the results of the automated approach generates many false positives. For example in [25], we observed that CVEs are wrongly associated to the *admin@338* group in a report about the Poison Ivy malware, where several campaigns from different actors are described. We provide a manually curated database from which we can quantitatively evaluate the impact and cost of software update strategies.

Several studies evaluated the overlap among threat data feeds, showing poor accuracy. Mechanisms to semi-automatically extract information from reports are prone to false-positive.

3.2 Analysis of attackers characteristics

Ussath et al. [20] analyzed 22 reports about APT campaigns and mapped them into the three phases of an attack (initial compromise, lateral movements, and Command&Control). They found that most of them employ social engineering techniques and living-off-the-land techniques. Furthermore, they noted that 0-day vulnerabilities are not exploited frequently by APTs. Chen et al. [13] studied 4 APT campaigns to analyze the phases of these attacks and determine possible countermeasures. Urban et al. [12] analyzed 93 APT reports (66 different APTs) and determined that spearphishing is the main attack vector. They then collected OSINT data like domain names and social media information of 30 companies to determine how much information is available to the adversaries. Additional works on APTs analysis focused on describing the phases of the attacks and possible countermeasures [10], the analysis of the malware employed in a few well-known campaigns [21], or the prevalence of living-off-the-land techniques in certain samples [51].

To the best of our knowledge, we are the first to analyze more than 350 campaigns exploiting 118 different CVEs from the inspection of more than 500 reports. This massive analysis makes it possible to draw significant conclusions on the efficacy of update strategies.

Although several works provided insights into the APT ecosystem, the analysis focused on a handful of campaigns that make it hard to draw significant conclusions on the characteristics of APTs.

3.3 Detection of attacks

An orthogonal problem is to detect live APTs attacks once they get into the network. Different research proposed to employ machine learning [39], [34], [46], information flow tracking [37], [42], [44], [45], statistical correlation [52], and big data analysis [36], [14], [33], [35].

Shu et al. [41] employed a temporal computational graph to perform threat hunting activities via graph patterns matching and analyzed a case study on a DARPA threat detection competition. Pei et al. [38] developed a framework to generate a multi-dimensional weighted graph based on log entries and identify attacks by the presence of dense connections among logs using unsupervised learning techniques. They evaluated it over 15 APTs campaigns.

The state-of-the-art focused mainly on the detection and response against APT attacks, while there is a lack of investigation on the orthogonal problem of prevention.

3.4 Game Theory

Hu et al. [47] presented a two-layer attack/defense game to study APT attackers that make use of insiders and compute the best strategies for the attacker and defender. Sahabandu et al. [40] formulated a game-theoretic model to determine the optimal defender strategy in terms of tracking of information flow (Dynamic Information Flow Tracking). Yang et al. [48] proposed a Nash game to model the response strategy and minimize the loss of an enterprise against lateral movements in the network of APTs in the network. We instead focus on the initial access phase of APTs campaigns and we evaluated the efficacy of software update strategies based on real data of attacks.

Game theory is extensively applied to find an optimal strategy against targeted attacks. However, these studies employ artificial data and networks.

3.5 Analysis of exploitation likelihood

Many works employed ML and statistical methods to analyze vulnerabilities and predict the exploitation likelihood by joining data from resources like NVD, Exploit DB [3], historical data on attacks [4], [7], Dark Web forums [53], and Twitter [54], [8]. An extensive discussion of the academic literature on empirical cyber risk can be found in [55].

Other works investigated actual compromises using logs. Marchetti et al. proposed a framework to prioritize the internal clients of an organization that are most likely to be compromised by an APT using internal (network logs and flow records) and external (social media) data [36] and to detect data exfiltration using a set of host-based features and flow records analysis [14]. Similarly, Bilge et al. [5] and Liu et al. [56] employed supervised learning algorithms to determine machines at risk of infection from *internal* logs on binary file appearance, *external* data of misconfigured services (e.g. DNS or BGP), and malicious behaviors (e.g. spam or phishing).

We extend this line of research by proposing a methodology to evaluate the probability of being compromised by APTs and the cost associated with the update strategy.

Analysis of historical data about vulnerability and attacks as well as live information provided by logs and social platforms allows one to evaluate the exploitation likelihood.

3.6 Analysis of update releases

From the client-side, Nappa et al. [27] proposed a systematic analysis of the update process and update delay on client applications, and performed a survival analysis of vulnerabilities based on data from Symantec. Similarly, Kotzias et al. [1] presented a longitudinal study of the update behavior for 12 client software and 112 server applications based on data from 28k enterprises. Sarabi et al. [57] employed Symantec dataset to model users' update delay as a geometric distribution and study 4 different products (Chrome, Firefox, Thunderbird, and Flash Player).

From the vendor-side, Arora et al. [58] analyzed vendors' patch behavior as a function of several factors like disclosure time, characteristics of the vendor, and severity of the vulnerability. Clark et al. [59] studied if agile methods produce a higher number of vulnerabilities in Firefox. They observed that rapid software releases do not increase the number of vulnerabilities in the code. Ozment and Schechter [60] analyzed the impact of legacy code on the number of vulnerabilities observed OpenBSD versions.

Similar to our work, Beres et al. [61] employed a discrete-event simulator to determine the exposure reduction produced by different security policies by varying update speed and mitigations. However, they modeled events like exploits and updates availability assuming fixed exponential functions looking at global trends observed by a security firm.

We present a quantitative evaluation of the effectiveness and cost of realistic update strategies by using historical data about APT campaigns.

Several works analyzed the update behavior of clients and vendors. However, there are only theoretical works on the efficacy of updates against targeted attacks for an enterprise.

4 Methodology

In this section, we present a methodology to evaluate the effectiveness and cost of update strategies.

The definition of probabilistic risk assessment [62] is:

$$Risk = Pr(Compr|Attack) \cdot P(Attack) \cdot Impact \qquad (1)$$

How to determine $P(Attack)$ is still an unsolved problem in cyber-security [63] while the $Impact$ of cyber-attacks has received extensive discussion [64], [65]. In this paper, we focus on $P(Compr|Attack)$ i.e. the *conditional* probability of being compromised given an attack (or campaign as used in this paper). We propose a methodology to compute the *conditional* probability of being compromised in Eq. 1 by employing historical data about releases available, vulnerabilities, and their exploitation in campaigns. Tab. 5 overviews our methodology.

Step 1: Extract APT and software data

To collect data we analyzed both unstructured (technical reports, blogs about APT campaigns, and vendor's repositories) and structured (MITRE Att&ck and NVD repositories) public sources.

Unstructured sources: Similar to Urban et al. [12], we manually collected data about APT campaigns from more than 500 technical reports and blogs. We started from the MITRE Att&ck APT groups list and one researcher:

- collected the reports associated with each APT group from the Threat Actor Encyclopedia [66], that relies on sources like Malpedia, MISP, AlienVault, and MITRE;
- extended this set of resources by searching on the Internet for reports using as keywords the APT name as stated in MITRE (e.g. Stealth Falcon) and the term "CVE" until data saturation was reached, i.e. new reports do not add new information to the APT campaigns. The reports are obtained from cyber-security companies like Kaspersky, FireEye, Palo Alto Networks, Google Project Zero as well as from technical forums and blogs.

Extracted Information

Two researchers independently analyzed the content of each report manually to identify the following information for a campaign:

- the *date* when the campaign is first observed;
- the *CVE(s)* exploited;
- the *attack vector(s)* employed.

We uniquely identify a campaign using the *date* in which it is first observed. If a campaign employs different attack vectors and/or different CVEs, we create multiple entries in the form <*APT_name,attack_vector,date*> or <*APT_name,CVE,date*>. Each entry is linked to one or more reports containing this information.

We do not perform open coding because the information in the reports is deterministic and already based on the MITRE industry standards on CVEs [1] and Initial Access Tactic[2]. The association of CVEs and attack vectors to a certain APT is based on the explicit attribution in the consulted resources. Let us consider the following snippet from a Mandiant report referring to APT12[3]:

In June 2014, the [Arbor Networks] blog highlighted that the backdoor was utilized in campaigns from March 2011 till May 2014. Following the release of the article, FireEye observed a distinct change in RIPTIDE's protocols and strings. . . . FireEye dubbed this new malware family HIGHTIDE.
On Sunday August 24, 2014 we observed a spear phish email sent to a Taiwanese government ministry. Attached to this email was a malicious Microsoft Word document (MD5: f6fafb7c30b1114befc93f39d0698560) that exploited CVE-2012-0158. It is worth noting. . .

we extracted the following information: *date*=08/2014; *CVE*=CVE-2012-0158; *attack vector*=spearphishing attachment.

The entries were then reviewed by a third researcher, not involved in the initial manual analysis, to resolve inconsistencies. Cohen's kappa values are 1, 0.976, and 0.863 for the CVE, date, and attack vector respectively (42 disagreements over 652 entries) which show a good agreement among the raters. Total agreement on CVEs is unsurprising as CVEs are unique strings and reported by copying and pasting the string into the data collection form. Such agreement would not happen between a manual rater and an automatic procedure as we already noted for DAPTSET [25] which is so riddled with false positives to be unusable. Simply, an automatic procedure will collect all CVEs including those that a human rater will see as clearly irrelevant (past campaign, related examples, etc.). Most disagreements are on the attack vector as the mapping of the natural description into the corresponding MITRE Att&ck category is sometimes amenable to interpretation (27 out of the 42 disagreements).

To resolve uncertainty among resources, we made the following *conservative* assumptions:

- if report A says CVE-1 is exploited by an APT campaign and report B says CVE-2 is exploited we mark both CVE-1 and CVE-2 as exploited by the APT in question.
- if report A says an APT campaign started on month X and report B says an APT campaign started on month Y we mark them as two distinct campaigns.

It is not uncommon that different security companies have non-overlapping information about APT campaigns [19], [18]. We discuss the implications of this choice in §7. Fig. 2 in §5 summarizes the number of reports per APT.

For the software, we retrieved versions for a subset of the targeted products (discussed in §5.4) with their release date.

1. https://www.cve.org/About/History
2. https://attack.mitre.org/tactics/TA0001/
3. https://www.mandiant.com/resources/darwins-favorite-apt-group-2

TABLE 5: Methodology overview

Step 1: Extract APT and software data	
INPUT	APT groups from MITRE Att&ck
OUTPUT	A set of campaigns in the form <*APT_name,CVE,date*>, <*APT_name,attack_vector,date*> and a set of software updates in the form <*sw,update,release_date*>
PROCEDURE	Identify campaigns information and software releases: • Collect resources describing campaigns for each APT based on Threat Actor Encyclopedia [66] and Internet searches using the MITRE APT name and ”CVE” as keywords; • Manually extract from resources the key information: *date* when campaign is observed, *CVE(s)* exploited, *attack_vector* employed; • For each CVE, automatically extract software and versions affected from NVD; • Manually extract from software vendors website the *update* number and *date* of release.
Step 2: Instantiate update strategy	
INPUT	A set of software updates (<*sw,update,release_date*>), update strategy (*Immediate*, *Planned*, *Reactive*, *Informed Reactive*), CVEs exploited in APT campaigns
OUTPUT	A matrix that describes the application of updates for the software in the period [2008-2020]
PROCEDURE	Create a matrix with rows identifying software versions and columns identifying months in [2008-2020] that determines the installed software version at a given time: • Select the entry corresponding to the first vulnerable version available on 01/2008 (same for all strategies); • Select another entry corresponding to a new version depending on the update strategy: on the release date of an update for the software (*Immediate*) or with a delay (*Planned*), on the publication (*Reactive*) or reservation date (*Informed Reactive*) of a CVE for the software with a delay; • Consider availability of non-vulnerable updates at the time of publication of a CVE when computing delay for *Reactive* and *Informed Reactive*.
Step 3: Instantiate APT campaigns events	
INPUT	Set of events for different campaigns (<*APT_name,CVE,date*>)
OUTPUT	A set of matrices of campaigns. Each matrix describes the software versions targeted by a certain campaign in the period [2008-2020]
PROCEDURE	For each campaign, create a matrix with rows identifying software versions and columns identifying months in the [2008-2020] that determines targeted software version at a given time: • Extract the affected software versions from the CVEs; • Select the entry of the affected software versions from the *date* of the campaign up to 2020.
Step 4: Generate pessimistic scenarios	
INPUT	A matrix that describes the application of updates for the software in the period [2008-2020]
OUTPUT	A matrix that describes the application of updates for the software in the period [2008-2020] and maintains both versions during the transition month
PROCEDURE	Update matrix to maintain the previous version in the month in which a new update is installed: • For each month in which the software version is updated to a new version, keep the entry corresponding to the previous version installed for that month only.
Step 5: Compute conditional probability of being compromised	
INPUT	A set of matrices of update strategies and a set of matrices of campaigns events
OUTPUT	The conditional probability of being compromised given a set of campaigns are targeting you ($P(Compr\|Attack)$) based on the update strategy, # of updates performed
PROCEDURE	Compute successful campaigns targeting installed software in the period [2008-2020]. For each matrix of update strategy: • Select a matrix of campaign events and compute the element-wise product of the matrix with the update strategy matrix to identify the intersection of installed and targeted software versions; • Sum rows of the resulting matrix to determine the months when a campaign is successful, save the campaign if successful; • Continue with another matrix of campaign events until no more campaigns. • Compute conditional probability as the number of successful campaigns divided by the number of matrix campaigns considered; • Compute the number of updates counting non-empty rows in the matrix of update strategy.
Step 6: Compare strategies effectiveness	
INPUT	The successful campaigns for the different update strategies
OUTPUT	Confidence Intervals (CI) for update strategies
PROCEDURE	Compare the CI intervals of different update strategies: • Compute the Agresti-Coull 95% CI for the proportion of successful campaigns by update strategy; • Compare intervals, if they overlap update strategies are similar; • Compute pair-wise agreement of successful campaigns for pair of update strategies and Agresti-Coull 95% CI for the resulting proportion of agreement. The interval identifies the expected range of proportion of campaigns that succeed against both update strategies.

This procedure was manual as it is not trivial to obtain past versions release date [27] because vendors' repositories are unstructured and not intended for past versions indexing.
Structured sources: For each CVE obtained from the unstructured sources, we automatically extracted the list of products and versions affected from NVD. This information is integrated with the Common Platform Enumeration (CPE) Dictionary. From the CPE, we extracted the list of vulnerable versions based on the CPE Match Strings.[4]

Step 2: Instantiate update strategies

We employed a matrix representation to compare update strategies and APT campaigns.

Each strategy is represented as a matrix in which the rows represent the versions of the different products (e.g. *Acrobat Reader* 9.2, *Flash Player 11.0.1.152*) and the column a specific date with a month-base granularity (e.g. 12/2009). A matrix cell is 1 if, on that date, that version of the product is installed and otherwise 0. For a first approximation, we avoid considering the presence of multiple versions installed for the same product[5].

All strategies start from the same version, that is the oldest vulnerable version of a campaign that is available at the beginning of 2008. A strategy updates its version based on the release date of a new version, the publication date, and the reservation date of a CVE for the *Immediate* and *Planned*, *Reactive*, and *Informed Reactive* strategy respectively.

The first two strategies (*Planned* and *Immediate*) update when a new release for the software is available (w/ and w/o an update interval respectively). If multiple software versions are released on the same date, they will update to the newest consistent version.[6]

For the latter two strategies (*Reactive* and *Informed Reactive*) the next version installed, if available, is the *first most recent version* that is not affected by the CVE. We also considered the availability of updates based on the attack scenario in *Step 4*.

Identification of the outcome of attack scenarios

Depending on the availability of an update at the time of the publication of the CVE we have to discern two scenarios:

- The release of an update is available *before* the publication of the CVE ($t_{U_r} \leq t_{V_p}$). The time when a company may decide to update because it is aware of the vulnerability is correctly computed from the time when a new vulnerability is published. This is the (implicit) assumption in [1], [27].
- The release of an update is available *after* the publication of the CVE ($t_{V_p} < t_{U_r}$). In this case, computing the time when a company may decide to update from the time of publication of the vulnerability will include an interval of time where a vulnerability for the version of a product is known but a non-vulnerable version has not been released yet ($t_{U_r} - t_{V_p}$). The time available to the company must be computed from the time when the release is available.

Step 3: Instantiate APT campaigns events

We created a matrix for each APT campaign with the same rows and columns of the update strategy matrix in *Step 2*. An entry is set to 1 if the version is affected by a CVE exploited by the campaign from the date when the campaign starts until 2020.

Step 4: Generate pessimistic scenarios

The updates and attacks have a month-based granularity because most of the resources *do not* contain information about the exact day in the month in which an update is published or a campaign is performed. We further discuss the limitation of these data in §7.

To balance possible interleaves between updates and campaigns within the same month, we performed two analyses: a pessimistic *APT-first* scenario and an optimistic *Update-first* scenario, that assume the campaign is executed before or after the update respectively.

To simulate the *APT-first* scenario, we create a new matrix from the update strategy matrix where we maintained the previous version also in the month in which the new update is installed. In other words, the two versions coexist in the month. Thus, we simulate the application of the update later in the month while allowing the APT to exploit the vulnerability. This is done by keeping selected the entry corresponding to the previous version also in the column in which we move to another version for each update strategy matrix generated in the *Step 2*.

Step 5: Compute conditional probability of being compromised

We evaluate at each instant of time, with a *month-base* granularity, the sequence of versions installed on a set of software products for each strategy and compare them with the software exploited by the APTs to determine the potentially successful campaigns.[7] We use the term *potentially successful* because the success of exploiting a vulnerability depends on the characteristics of the execution environment [67]. A campaign is considered successful if it exploits at least one of the software products considered. From the matrix of updates obtained from *Step 2,5* and the matrix of campaigns obtained from *Step 3*, we compute the *conditional* probability of being compromised given one is targeted by the campaigns at a given instant of time t_i. The probability is computed as the number of potentially successful campaigns at time t_i over the total number of campaigns active at that instant of time.

$$P(Compr|C, t = t_i) = \frac{|potentially\ successful\ campaigns_{t_i}|}{|active\ campaigns_{t_i}|} \tag{2}$$

4. For example, CVE-2016-4113 affects all the versions of *Flash Player* up to 21.0.0.213. The associated JSON NVD file does not provide the entire list of affected versions (including the updates) in the CVE description but a CPE URI of the form *cpe:2.3:a:adobe:flash_player:*:*:*:*:*:*:*:*", "versionEndIncluding":"21.0.0.213"*, we thus matched the CPE in the CPE dictionary to get the list of all prior versions affected.

5. We assume an update is applied on enterprise's machines at once.

6. For example, if the current *JRE* version installed is *6u6* and a new update for JRE *5u13* is released after that, the update is ignored because it represents a downgrade of a major update.

7. For example, in 12/2009 the CVE-2009-4324, affecting *Acrobat Reader* up to version 9.2, is exploited in the wild. If at any time from 12/2009 an update strategy updates to one of these versions, then the campaign is potentially successful.

where:

- $|potentially\ successful\ campaigns_{t_i}|$ is the number of active campaigns at time t_i that exploit at least one version of a product currently installed at that time.
- $|active\ campaigns_{t_i}|$ is the total number of active campaigns at time t_i.

We computed the $|potentially\ successful\ campaigns_{t_i}|$ by performing an element-wise product of the matrix of update strategy with each matrix describing an APT campaign. The resulting matrix identifies the versions that were installed *and* exploited by the campaign in a given month. With the sum of the rows of the resulting matrix, one obtains a vector of values ≥ 0 for each t_i.[8] If an entry at time t_i is > 0, then the campaign is included in $|potentially\ successful\ campaigns_{t_i}|$.

The overall percentage of potentially successful campaigns over the total number of campaigns in the entire interval of time is computed as:

$$\begin{aligned} P(Compr|C) &= \frac{|potentially\ successful\ campaigns|}{|campaigns|} \\ &= \frac{|\{C|\exists t_i : C \in potentially\ successful\ campaigns_{t_i}\}|}{|campaigns|} \end{aligned} \quad (3)$$

In other words, the total number of potentially successful campaigns is obtained from the set of campaigns that could be successful in at least one instant of time t_i. If a campaign can succeed in several instants of time, it is counted only once in the period of interest.

The number of software updates is obtained from the matrix representing the strategy, by counting the number of rows that contain at least one non-zero entry in the columns.

Step 6: Compare strategies effectiveness

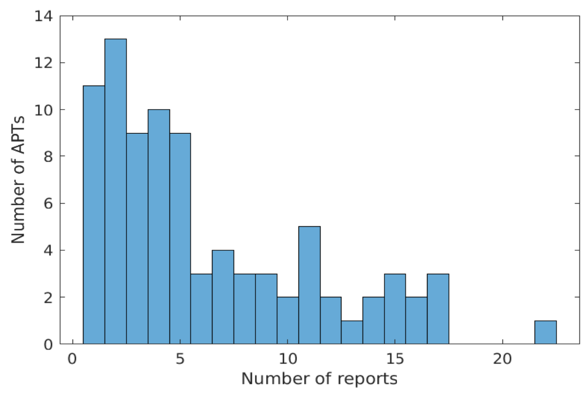
For each update strategy, we obtain from Eq. 3 a probability of being compromised based on the sample of campaigns considered. To predict the range in which the probability of being compromised for the entire population of campaigns resides we compute a confidence interval (CI). In case of binary outcomes (success, failure), we compute the Agresti-Coull confidence interval [68] that is recommended when the sample size is ≥ 40 [69]. From the CIs of different update strategies, we can then compare their performance. Two strategies are similar if their CIs significantly overlap.

We then determine the percentage of campaigns for which the two strategies behave in the same way by computing the proportion of campaigns that either succeeded or failed against both strategies. By computing the Agresti-Coull interval for the resulting proportion we obtain the range of similarity of the two strategies in terms of the percentage of campaigns that both succeed or failed against two update strategies.

5 Dataset

We considered only APT groups that launched at least one campaign from 2008 to 01/2020 and for which a precise date for the campaign is present in at least one report.

8. Values can be > 1 if campaigns can exploit different products.



Only for 11 APTs (13%) we were not able to find more than one resource for their campaigns. This is typically due APTs that are not particularly active or that are tracked by a single cyber-security company.

Fig. 2: Number of collected reports per APT.

TABLE 6: Attack vector campaigns and software vulns

	# of Campaigns	
Attack vector	w/o vuln	w/ at least one vuln
Spear phishing	130*	122*
Drive-by Compromise	15*	34*
Supply Chain Compromise	5*	0
Valid Accounts	3*	1
External Remote Services	3	0
Exploit Public-Facing Appl.	3*	7
Replic. via Remov. Media	0	1
Undetermined	38*	9*
Total	197 (190 unique)	174 (162 unique)

* Contains duplicates due to multiple attack vectors.

The final database contains information about 86 APT groups. For the excluded APTs, we either did not find information for their campaigns, or the date of their campaign was not known. For example, the Kaspersky article [70] provides a list of CVEs but does not provide information about the campaign when they were exploited. Fig. 2 shows the distribution of reports per APT.

For more than half of the APT campaigns saturation is reached with at most 5 distinct resources, while for some APTs we collected more than 15 and up to 22 different resources. Only for 11 APTs, we collected a single resource, which typically is a white paper containing detailed information about the APT's activity over an extended period of time.

We now answer *RQ1* with a quantitative analysis of the attack vectors employed, the vulnerabilities exploited, and the software products targeted.

5.1 Attack Vectors

We analyzed the attack vectors exploited in the different campaigns with the presence and absence of software vulnerability. Tab. 6 shows the different attack vectors and the number of campaigns in which are observed. We underline that a campaign can employ one or more attack vectors.[9]

We can observe that spear phishing is the main attack vector [12], present in 130 campaigns that do not exploit any vulnerability and 122 campaigns that exploit at least

9. For example, it is not uncommon to have campaigns that exploit both spearphishing and drive-by compromise.

TABLE 7: Top 10 client-side and Top 10 server-side/O.S. products exploited

The products are obtained from the CVEs exploited in a campaign. If a CVE affects multiple products, all the software are considered. Products are distinguished in *client-side*, *server-side* application, and *O.S.*

Vendor	Product	Software	# Campaigns (%)
Microsoft	**Office**	**Client**	**68 (41.9%)**
Microsoft	Windows 2008 Server	O.S.	49 (30.2%)
Microsoft	Windows 7	O.S.	43 (26.5%)
Microsoft	Windows Vista	O.S.	41 (25.3%)
Microsoft	Windows 2012 Server	O.S.	39 (24.0%)
Adobe	**Flash Player (EOL)**	**Client**	**35 (21.6%)**
Microsoft	Windows 8.1	O.S.	29 (17.9%)
Microsoft	Commerce Server	Server	19 (11.7%)
Microsoft	SQL server	Server	19 (11.7%)
Microsoft	Visual Basic	Client	19 (11.7%)
Microsoft	Visual FoxPro	Client	19 (11.7%)
Microsoft	BizTalk Server	Server	18 (11.1%)
Microsoft	Windows 10	O.S.	18 (11.1%)
Microsoft	Windows 8	O.S.	14 (8.6%)
Microsoft	IE (EOL)	Client	13 (8.0%)
Adobe	**Acrobat Reader**	**Client**	**11 (6.8%)**
Microsoft	.NET framework	Client	5 (3.1%)
Adobe	**Air**	**Client**	**5 (3.1%)**
Oracle	**JRE**	**Client**	**4 (2.5%)**
Oracle	JDK	Client	4 (2.5%)

one vulnerability. Interestingly drive-by compromise is not only employed when a vulnerability is present but also used to facilitate campaigns that employ social engineering to trigger users to download malware.

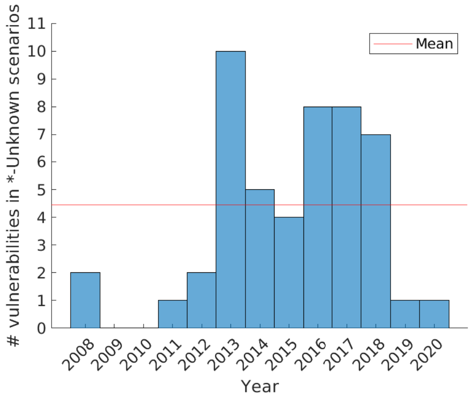
We have 47 campaigns for which we do not know the attack vector. For 9 of them, the report identified the vulnerability exploited but not the attack vector.[10] If this information is not present in the report, we avoided making assumptions. For the remaining campaigns, the information about the attack vector was vague or missing.[11]

5.2 Popular Products and CVEs

We observed 118 unique vulnerabilities exploited by the APTs in at least one campaign between 2008 and 2020. Some CVEs are exploited in several campaigns by different APTs.

Tab. 7 shows the ten most targeted client-side applications and the ten most targeted server/O.S. products based on the exploited CVEs. A campaign is counted over different products if the CVE employed is applicable to different software products. For example, CVE-2012-0158 affects Office, SQL server, Visual Fox Pro, and Commerce Server.[12] *Office* is by far the major target of campaigns followed by *Windows O.S.* and *Flash Player*. This is coherent with the attack vectors previously observed as they are commonly exploited via spearphishing with malicious attachments.

APTs tend to ”share” vulnerabilities during their campaigns. Only 8 APTs (`Stealth Falcon`, `APT17`, `Equation`, `Dragonfly`, `Elderwood`, `FIN8`, `DarkHydrus`, and `Rancor`) exploit CVEs that are not used by anyone else.[13] We are aware of vulnerabilities (e.g CVE-2017-0144) that are associated with `Equation` and used by other APTs, but we did not find enough information about the date when the vulnerabilities were employed. Roughly 35% of the APTs exploit CVEs observed in campaigns of other groups. 17 APTs share 4 or more vulnerabilities, while many APTs sharing a single vulnerability have only exploited that vulnerability during their campaigns (14 out of 20).



The number of unique vulnerabilities employed in *Unknown-Unknown (UU)* and *Known-Unknown (KU)* attack scenarios grows significantly in recent years, compared to the first years of observation. On average around 5 distinct vulnerabilities per year are exploited by APTs.

Fig. 3: Number of distinct vulnerabilities exploited over the years by different attack scenarios.

5.3 Evolution in exploiting vulnerabilities

Fig. 3 shows the evolution of the number of unique vulnerabilities exploited in the **-unknown* attack scenarios[14] in our database. It represents a lower bound of the vulnerability exploited in the wild. Project Zero [72] collects information about 0-days in the wild by including also unattributed attacks. The mean number of distinct vulnerabilities exploited per year is roughly 5. We can observe how the numbers grew significantly in recent years. However, it can be influenced by the limited number of reports for campaigns in the early period (2008-2011), where it was less likely to report information about cyber-attacks. The drop for 2019/2020 is due to the natural delay of publicly reporting campaigns caused by the proximity of the period of data collection with the date of the campaigns themself. Thus, we expect the values to be higher if recomputed in the future.

Looking at the occurrence of a CVE in an APT campaign, the majority of the APTs prefer to exploit CVE already published, with few APTs as exceptions.[15]

5.4 Software for Analysis of Update Strategies

As discussed in §4, the collection of update releases from vendors' websites is a manual procedure. Here, for a first approximations, we focus on collecting updates for a subset of all software targeted by APTs.

Tab. 7 shows the most targeted products by vendor. We decided to cover the most exploited client-side product for each vendor because (1) from Tab. 6 most of the campaigns

10. For example, some vulnerabilities (e.g. CVE-2012-0158) can be exploited via spearphishing techniques and drive-by compromise.

11. For example, the Sony hack campaign in 2014 [71].

12. We do not have information about the exact software targeted. For example, they could all have exploited Office.

13. Only three APTs have exploited more than one vulnerability during all their campaigns.

14. Either already reserved *(KU)* or not reserved *(UU)*.

15. `Stealth Falcon, PLATINUM, APT17`

exploit attack vectors directed to client-side software and (2) it is not uncommon to have products from these vendors in an enterprise computer. For Adobe, the *Flash Player* product is end of life (EOL) thus we decided to include the other two software products *Reader* and *Air*. Even if *Flash Player* is EOL in 2020 we still think it is interesting to see how different update strategies would affect the security of enterprises because it has been frequently exploited in the last years. Also, vendors' EOL of products, unfortunately, does not coincide with the disappearance from the field and end of exploitation as we are observing with Internet Explorer [73].

For a first approximation, we limited the analysis to the *Office* 2016 release only, as different releases (*Office* 2013, *Office* 365) can be seen as different products as they require buying a different license each. We considered the Knowledge Base (KB) updates from the Microsoft Update Catalog as the versions of the software. We assumed that KB updates for *Office* are cumulative, i.e. the package contains all previously released fixes.

In summary, we collected releases of updates for 5 different software products from 3 different vendors: Office, Flash Player, Acrobat Reader, Air, and JRE. We considered only releases for Microsoft Windows O.S. as it covers at least half of the enterprise computers [74]. With this set of software products, we cover 44% of the campaigns (that exploit software vulnerabilities), 62% of the APT groups, and 33% of the CVEs.

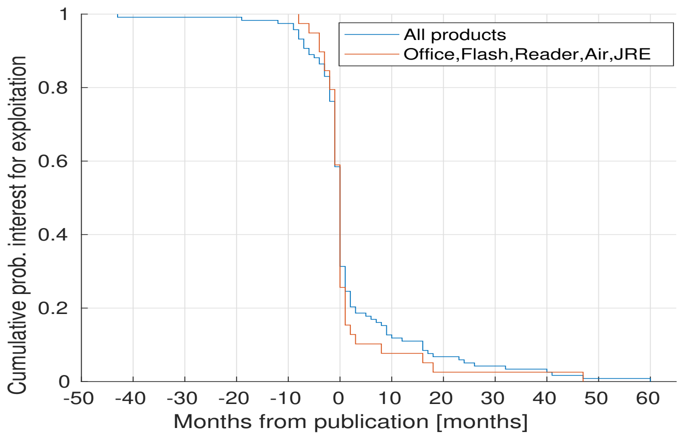
6 Quantitative Analysis of Updates

We now present an analysis of the speed of exploitation of individual vulnerabilities and the prevalence of **-Unknown* and *Known-Known* attacks in APT campaigns. We then quantitatively evaluate the effectiveness and cost of the different update strategies against the APT campaigns.

6.1 Survival Analysis

We performed preliminary survival analysis on the vulnerabilities to compute the interval in months that passed from the publication of the CVE and the *first* campaign that exploited the CVE (exploit age). Fig. 4 shows the Kaplan-Meier plot for all the products in our database and for the set of products discussed in §5 (*Office*, *Flash Player*, *Reader*, *Air*, and *JRE*). We can see that roughly 40% of the vulnerabilities are exploited for *the first time* before the publication. This is coherent with what was observed by Chen et al. [8], where 49% of the CVEs are exploited before the NVD score is published. Furthermore, roughly 27% of the vulnerabilities are exploited *the first time*[16] within a month from the publication from NVD showing that APTs are fast to exploit new CVE [75]. Another interesting fact is that a significant number of vulnerabilities are exploited a few months before the NVD publication. This phenomenon can be partially explained because the observation of attacks in the wild brings software vendors to know about the vulnerability and thus the publication of a CVE. It is important to underline that this value does not mean that ≈40% of the campaigns are unpreventable because 1) **-Unknown* attacks can exploit several vulnerabilities[17] and 2) many of these

16. Among all the APTs.
17. A famous example is Stuxnet.



The survival is based on the *first* time the CVE is exploited in a campaign. More than half of the vulnerabilities are exploited for the *first time* within one month from the publication. However, there is high survivability of a small set of CVEs (roughly 10%) that are exploited after more than 1 year from the publication. If we consider only *Office*, *Flash Player*, *Reader*, *Air*, and *JRE* the behavior is similar.

Fig. 4: Proportion of survival of CVE from publication (NVD) for all products and a subset (*Office*, *Flash Player*, *Reader*, *Air*, and *JRE*).

CVEs are exploited multiple times from different APTs after months from their first exploitation.

If we only consider the vulnerabilities exploited the first time in *KK* attacks (as in [76]), we observe that roughly 47% of them are exploited within 30 days from their publications[18]. In contrast with previous results [77], we observed a long tail for part of the vulnerabilities, one out of 10 CVE is exploited after one year from its publication, and 1 out of 20 after more than two years.

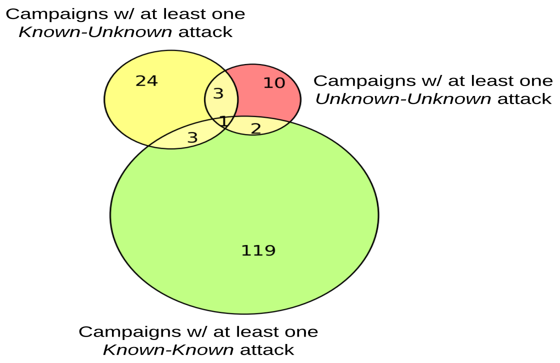
6.2 Classification of APT campaigns

Each APT campaign exploiting at least one vulnerability fits into one of these (possibly overlapping) groups:

- Campaigns with at least one *Known-Known (KK)* attack. In other words, the campaign exploited at least one vulnerability (either preventable or unpreventable) that was already present in the NVD database.
- Campaigns with at least one *Known-Unknown (KU)* attack. In other words, the campaign exploited at least one vulnerability (either preventable or unpreventable) that was not present in the NVD database but an entry was already reserved by MITRE.[19]
- Campaigns with at least one *Unknown-Unknown (UU)* attack. In other words, the campaign exploited at least one vulnerability (either preventable or unpreventable) that was not even reserved by MITRE.

Out of 352 campaigns, less than half of them employ at least one vulnerability (Tab. 6). Figure 5 shows the resulting Venn diagram for the 162 campaigns of interest. 119 out of 162 campaigns employed only vulnerabilities in *Known-Known* attacks.

18. Bilge et al. [76] observed a similar value of roughly 42%
19. Thus, a small number of people known already some information about the vulnerability. E.g. vulnerability researchers.


Campaigns w/ at least one *Known-Unknown* attack
24
3
10
Campaigns w/ at least one *Unknown-Unknown* attack
1
3
2
119
Campaigns w/ at least one *Known-Known* attack

The majority of campaigns exploited at least one vulnerability in a *KK* attack (after publication by NVD and after reservation by MITRE). Only a few launched *UU* attacks (both before reservation by MITRE and before publication by NVD).

Fig. 5: Classification of APT Campaigns.

APTs heavily exploit known CVE to compromise their target. The prioritization of updates is thus a key factor that can significantly reduce the impact of APTs campaigns.

6.3 Evaluation of software updates strategies

We now answer *RQ2* by applying our methodology (§4) to compute the overall probability of being compromised (Eq. 3) in the interval of time [Jan 2008-Jan 2020] with the updates strategies and update interval presented in §2 for the software discussed in § 5.4. Tab. 8 summarizes the results in terms of the number of updates required, the conditional probability and the odds ratio for the *optimistic* (*Update first*) and *pessimistic* (*APT first*) scenarios.

Updating the software as soon as a new release is available (*Immediate* strategy) provides the optimal lower-bound probability of being compromised. Even in this case, roughly 1 out of 4 campaigns can compromise the target. Although an immediate update can be applied in some critical situations, if we consider a more realistic approach in which the software is updated with some delay in the month (*Immediate* with *APT first*), the odds of being compromised increases by a factor of 5.

The *Planned* strategy provides a similar, although slightly better, probability of being compromised compared to a strategy that waits for the presence of public vulnerabilities (*Reactive* strategy). However, waiting to update when a CVE is published presents 8x times fewer updates. Thus, if an enterprise cannot keep up with the updates and need to wait before deploying them, can consider being simply reactive. For the *Planned* strategy the number of updates decreases with bigger intervals because the updates are shifted outside of the period of observation. If a longer update interval is used, the probability of being compromised increases by a factor of 9 and 20 for 3 months and 7 months update intervals respectively. Interestingly, for the 7 months delay, we have that the *Reactive* and *Informed Reactive* perform slightly better than the *Planned* strategy.

Comparing the *Reactive* and *Informed Reactive* strategies, there is a small advantage in knowing about not publicly known vulnerabilities only if the update interval is small.

TABLE 8: Optimistic (*Update first*) and pessimistic (*APT first*) overall *conditional* probability of being compromised for different update strategies and update interval with the associated # of updates for the period [01/2008-01/2020]

Update Interval	Strategy	#Updates	Prob. (*Update first — APT first*)	Odds
/	Immediate	360	22.2-58.3%	1x-4.9x
1 Month	Planned	357	58.3-63.9%	4.9x-6.2x
	Reactive	44	61.1-66.7%	5.5x-7.0x
	Informed Reactive	44	58.3-66.7%	4.9x-7.0x
3 Months	Planned	350	72.2-75.0%	9.1x-10.5x
	Reactive	44	73.6-76.4%	9.8x-11.3x
	Informed Reactive	44	73.6-76.4%	9.8x-11.3x
7 Months	Planned	337	86.1-87.5%	21.7x-24.5x
	Reactive	44	84.7-86.1%	19.4x-21.7x
	Informed Reactive	44	84.7-86.1%	19.4x-21.7x

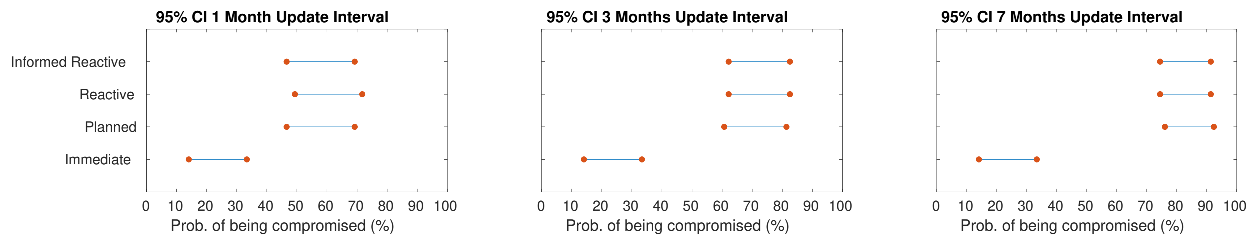
Once the enterprise waits 3 to 7 months, the vulnerability is now publicly known and actively exploited by the APTs.

We reported in Fig. 6 the Agresti-Coull Interval for each update strategy for the different update intervals. The *Planned*, *Reactive*, and *Informed Reactive* strategies are almost identical as we see a significant overlap of the CI among these three strategies. The probability of being compromised lies within [52%-74%] for the *Planned* and *Informed Reactive* and [55%-77%] for the *Reactive* in the pessimistic scenario. In the case of an optimistic *Update-first* scenario, we observe that there is a clear difference between the *Immediate* and the *Planned* strategies, while this advantage is lost in the case of the pessimistic *APT-first* scenario. To evaluate the similarity we computed, for each pair of strategies, the proportion of campaigns that either succeeded or failed against both strategies. We estimate the Agresti-Coull CI for the resulting proportions. The results show that the *Planned* and *Reactive* behave in the same way for at least 90% up to 99% of the campaigns for a 1 month update interval, for 88% and up to 98% for a 3 months update interval, and for 92% up to 99% for a 7 months interval. While, the *Reactive* and *Informed Reactive* behave in the same way for 90% and up to 99% of the campaigns for a 1 months update interval, and for 94% up to 100% for a 3 and 7 months interval.

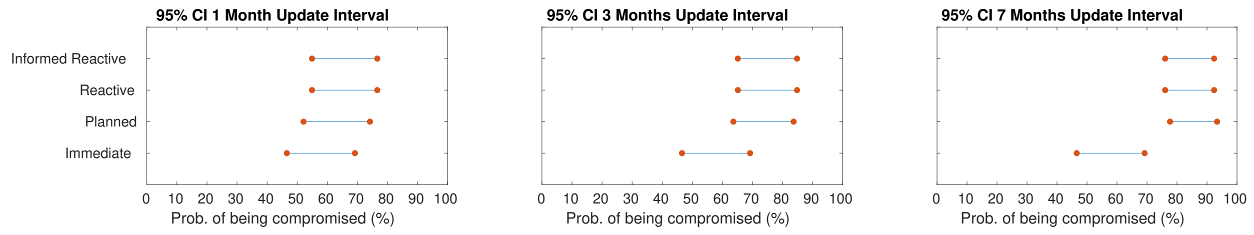
Since hackers focus on new versions, a strategy that always updates to the new version but with a delay gives time to the APT to target and exploit a vulnerability. In contrast, a reactive approach that updates rarely might present to attackers an older version that does not include the new vulnerable code [78]. In other words, either you update always and immediately to the new versions or just updating lately has the same risk profile but cost you a lot more than updating rarely [79].

7 Limitations

The dataset obtained is based on publicly available reports. While this is just a small part of existing campaigns, this paper is the first that tries to aggregate a manually validated dataset of APTs campaigns, CVE, and vulnerable products



(a) *Update-first* (optimist) scenario. There is a difference between the *Immediate* and the other strategies. However, *Planned*, *Reactive*, and *Informed Reactive* behave similarly thus updating to each new version with some delay or relying on reserved CVE does not worth it.



(b) *APT-first* (pessimist) scenario. The *Immediate* and *Planned* present a similar behavior for the 1 month update interval but differ with bigger intervals. In the pessimistic scenario the *Planned*, *Reactive*, and *Informed Reactive* behave similarly thus updating to each new version with some delay or relying on reserved CVE does not worth it.

Fig. 6: Agresti-Coull Interval (CI) for the update strategies with different update intervals

and it is a first step in the direction of an open and extensive dataset on APT campaigns.

The process to obtain information about campaigns was semi- automated but required manual effort to analyze and to extract the key information about campaign dates, CVE, and attribution. We assume that this type of information reported by reputable security companies is not deliberately wrong, and our methodology strives to find multiple sources reporting the same campaign to control for possible errors. Since keyword-based automated searches (e.g. [25]) present limitations in the number of false associations that they generate, we decided that a manual approach would provide a more precise description of the APT ecosystem. Although the manual extraction of information from reports does not present difficulties, it can include erroneous matching of APT campaigns. To limit that, the manual analysis was performed by two researchers independently and inconsistencies were resolved by a third researcher.

We decided to ignore reports about campaigns where not enough information about the start and attribution was available. Thus, it is possible that certain vulnerabilities discussed in the reports are not included in the dataset.

We applied a conservative approach in extracting information from different reports reporting mutually disjoint CVEs exploited on the same date. Thus, potentially assuming fewer campaigns with a higher number of CVEs each. The probability of being compromised must be seen as an upper bound of what APT can achieve. However, the odds ratio between update strategies remains the same.

We relied on the NVD data as the industry standard but it is known to contain errors in the list of product names, CVE publication date [80] and vulnerable versions [81], [82]. We leave for future work the application of these approaches to find inconsistencies. We relied on the data of observation of the campaigns as reported in the reports we consulted. This information could be wrong and detect only a more recent campaign. We tried, when possible, to find multiple resources about the campaign. The collection of release dates for the software discussed in §5.4 is collected manually given that vendors' repositories are not intended for past versions. Thus, the releases collected and employed in the evaluation might have errors and this could affect the *Immediate* and *Planned* strategies.

We used a month-based date granularity for the publication of the CVE, the release of new versions, and the date of the campaigns because the exact day in a month in which the campaign started is not known. This decision has a potential impact on the results. If a campaign for a CVE published on 29/01/2017 started on 01/02/2017 then in our case the exploit age is one month, even if the CVE is exploited a few days after the publication. However, the results we observed (e.g. exploit age of vulnerabilities) are coherent with previous observations of attacks in the wild [76], thus we think that the number of these cases is minimal and do not affect the results.

The same considerations apply to the results in Tab. 8: if a release is performed on 15/02/2019 and a campaign exploiting the software is executed on 03/02/2019, the month granularity would traduce both actions as performed on 02/2019. We thus considered two complementary scenarios: an *optimistic* scenario (*Update first*) and a *pessimistic* scenario (*APT first*). In the *Update first* the example above will traduce in the defender be able to update before the execution of the campaign. While in the *APT first* we assumed the opposite.

Finally, those companies that have an update interval that is less than a month will present a probability of being compromised that stays between the *Immediate Update-first* and the *Immediate APT-first* scenarios.

We assumed that a campaign will be carried on from the date when the campaign started up to the end of the observation (i.e. 2020). This causes an inflation of the number of campaigns that are active at a given instant of time in Eq. 2. However, we follow a conservative approach and assumed that if an APT has access to a vulnerability it will always be able to employ it given that one is under attack. We discuss extensions in the §8.

8 Conclusions and Future Work

In this work, we proposed *a methodology to quantitatively investigate the effectiveness and cost of software updates strategies against APT Campaign*. We applied the methodology to build a database of APT campaigns and presented an analysis of the attack vectors, vulnerabilities, and software exploited by 86 different APTs in more than 350 campaigns over 12 years. The database is publicly available on Zenodo [26].

In contrast to expectation, we showed that preventive mechanisms like updates can influence the probability of being compromised by APT. However, software updates based on wrong measures of risk can be counterproductive. Our analysis shows that a purely *Reactive* update strategy (wait until a vulnerability gets out) presents results very similar to a *Planned* strategy (always update to the newest version), but with only 12% of the updates. Furthermore, the *Informed Reactive* strategy, where updates are applied based on reserved information about not publicly known vulnerabilities (e.g. by paying for information on 0-days), does not produce significant advantages compared to the *Reactive* strategy and it is useless if the enterprise has several months of delay before applying the update.

In summary, for the broadly used products we analyzed, if you cannot keep updating always and immediately (e.g. because you must do regression testing before deploying an update), then being purely reactive on the publicly known vulnerable releases has the same risk profile than updating with a delay but costs significantly less.

Future work can extend the analysis to a more complete set of software products and evaluate a subset of campaigns by targeted enterprises, attacker preferences, or network exposure based on IDS alerts [63]. To achieve that, one would require to have company-specific information to move from a conditional probability to an absolute probability.

We also plan to extend the evaluation by considering campaigns as active only for a limited period. Further data about the lifetime of campaigns in the wild is required.

Acknowledgments

We thank the student Veronica Chierzi for helping in the data collection of update releases. This work was partly funded by the European Union under the H2020 Programme under grant n. 830929 (CyberSec4Europe) and n.952647 (AssureMOSS).

References

[1] P. Kotzias *et al.*, "Mind your own business: A longitudinal study of threats and vulnerabilities in enterprises," in *Proc. of NDSS-19*, 2019.
[2] I. Pashchenko *et al.*, "A qualitative study of dependency management and its security implications," in *Proc. of ACM-CCS-20*, 2020.
[3] M. Bozorgi *et al.*, "Beyond heuristics: learning to classify vulnerabilities and predict exploits," in *Proc. of SIGKDD-10*, 2010.
[4] L. Allodi and F. Massacci, "Comparing vulnerability severity and exploits using case-control studies," *TISSEC-14*, 2014.
[5] L. Bilge *et al.*, "Riskteller: Predicting the risk of cyber incidents," in *Proc. of ACM-CCS-17*, 2017.
[6] L. Allodi, "Economic factors of vulnerability trade and exploitation," in *Proc. of ACM-CCS-17*, 2017.
[7] J. Jacobs *et al.*, "Improving vulnerability remediation through better exploit prediction," in *Proc. of WEIS-19*, 2019.
[8] H. Chen *et al.*, "Using twitter to predict when vulnerabilities will be exploited," in *Proc. of SIGKDD-19*, 2019.
[9] L. Allodi *et al.*, "The work-averse cyberattacker model: Theory and evidence from two million attack signatures," *Risk Analysis*, 2021.
[10] A. Rot and B. Olszewski, "Advanced persistent threats attacks in cyberspace. threats, vulnerabilities, methods of protection," in *Proc. of FedCSIS-17*, 2017.
[11] A. Lemay *et al.*, "Survey of publicly available reports on advanced persistent threat actors," *Computers & Security*, 2018.
[12] T. Urban *et al.*, "Plenty of phish in the sea: Analyzing potential pre-attack surfaces," in *Proc. of ESORICS-20*, 2020.
[13] P. Chen *et al.*, "A study on advanced persistent threats," in *Proc. of IFIP-14*, 2014.
[14] M. Marchetti *et al.*, "Analysis of high volumes of network traffic for advanced persistent threat detection," *Computer Networks*, 2016.
[15] T. Steffens, *Attribution of Advanced Persistent Threats - How to Identify the Actors Behind Cyber-Espionage*. Springer, 2020.
[16] Kaskersky, 2016, https://securelist.com/cve-2015-2545-overview-of-current-threats/74828/. Accessed: 2020-11-01.
[17] Crowdstrike, 2014, https://www.crowdstrike.com/blog/french-connection-french-aerospace-focused-cve-2014-0322-attack-shares-similarities-2012/. Accessed: 2020-11-01.
[18] X. Bouwman *et al.*, "A different cup of ti? the added value of commercial threat intelligence," in *Proc. of USENIX-20*, 2020.
[19] V. G. Li *et al.*, "Reading the tea leaves: A comparative analysis of threat intelligence," in *Proc. of USENIX-19*, 2019.
[20] M. Ussath *et al.*, "Advanced persistent threats: Behind the scenes," in *Proc. of CISS-16*, 2016.
[21] N. Virvilis and D. Gritzalis, "The big four - what we did wrong in advanced persistent threat detection?" in *Proc. of ARES-13*, 2013.
[22] L. Anthony (Tony) Cox Jr, "What's wrong with risk matrices?" *Risk Analysis: An International Journal*, vol. 28, 2008.
[23] H. Kunreuther, "Risk analysis and risk management in an uncertain world 1," *Risk Analysis: An International Journal*, vol. 22, 2002.
[24] D. A. Scheufele and D. Tewksbury, "Framing, agenda setting, and priming: The evolution of three media effects models," *Journal of communication*, vol. 57, 2007.
[25] G. Laurenza and R. Lazzeretti, "daptaset: A comprehensive mapping of apt-related data," in *Proc. of FINSEC-19*, 2019.
[26] G. D. Tizio *et al.*, "Advanced Persistent Threats (APTs) campaigns database," 2022. [Online]. Available: https://doi.org/10.5281/zenodo.6514817
[27] A. Nappa *et al.*, "The attack of the clones: A study of the impact of shared code on vulnerability patching," in *Proc. of SSP-15*, 2015.
[28] C. Xiao *et al.*, "From patching delays to infection symptoms: Using risk profiles for an early discovery of vulnerabilities exploited in the wild," in *Proc. of USENIX-18*, 2018.
[29] C. Bogart *et al.*, "How to break an API: cost negotiation and community values in three software ecosystems," in *Proc. of FSE'16*, 2016.
[30] SANS, "Sans vulnerability management survey 2019," 2019, https://www.sans.org/reading-room/whitepapers/analyst/membership/38900. Accessed: 2020-12-01.
[31] P. Giura and W. Wang, "A context-based detection framework for advanced persistent threats," in *Proc. of ICCS-12*, 2012.
[32] W. Zhao *et al.*, "Extended petri net-based advanced persistent threat analysis model," in *Computer Engineering and Networking*, 2014.
[33] P. Bhatt *et al.*, "Towards a framework to detect multi-stage advanced persistent threats attacks," in *Proc. of SOSE-14*, 2014.
[34] S. Chandran *et al.*, "An efficient classification model for detecting advanced persistent threat," in *Proc. of ICACCI-15*, 2015.
[35] I. Friedberg *et al.*, "Combating advanced persistent threats: From network event correlation to incident detection," *Computers & Security*, 2015.
[36] M. Marchetti *et al.*, "Countering advanced persistent threats through security intelligence and big data analytics," in *Proc. of CyCon-16*, 2016.

[37] G. Brogi and V. V. T. Tong, "Terminaptor: Highlighting advanced persistent threats through information flow tracking," in *Proc. of NTMS-16*, 2016.

[38] K. Pei *et al.*, "HERCULE: attack story reconstruction via community discovery on correlated log graph," in *Proc. of ACSAC-16*, 2016.

[39] I. Ghafir *et al.*, "Detection of advanced persistent threat using machine-learning correlation analysis," *Future Generation Comp. Syst.*, 2018.

[40] D. Sahabandu *et al.*, "DIFT games: Dynamic information flow tracking games for advanced persistent threats," in *Proc. of IEEE CDC-18*, 2018.

[41] X. Shu *et al.*, "Threat intelligence computing," in *Proc. of ACM-CCS-18*, 2018.

[42] S. M. Milajerdi *et al.*, "HOLMES: real-time APT detection through correlation of suspicious information flows," in *Proc. of SSP-19*, 2019.

[43] Y. Shen and G. Stringhini, "ATTACK2VEC: leveraging temporal word embeddings to understand the evolution of cyberattacks," in *Proc. of USENIX-19*, 2019.

[44] W. U. Hassan *et al.*, "Tactical provenance analysis for endpoint detection and response systems," in *Proc. of SSP-20*, 2020.

[45] X. Han *et al.*, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *Proc. of NDSS-20*, 2020.

[46] A. Alsaheel *et al.*, "Atlas: A sequence-based learning approach for attack investigation," in *Proc. of USENIX-21*, 2021.

[47] P. Hu *et al.*, "Dynamic defense strategy against advanced persistent threat with insiders," in *Proc. of INFOCOM-15*, 2015.

[48] L. Yang *et al.*, "A risk management approach to defending against the advanced persistent threat," *IEEE Transactions on Dependable and Secure Computing*, 2018.

[49] X. Liao *et al.*, "Acing the IOC game: Toward automatic discovery and analysis of open-source cyber threat intelligence," in *Proc. of ACM-CCS-16*, 2016.

[50] K. Satvat *et al.*, "EXTRACTOR: extracting attack behavior from threat reports," in *Proc. of EuroSP'21*, 2021.

[51] F. Barr-Smith *et al.*, "Survivalism: Systematic analysis of windows malware living-off-the-land," in *Proc. of SSP-21*, 2021.

[52] J. Sexton *et al.*, "Attack chain detection," *Statistical Analysis and Data Mining*, 2015.

[53] M. Almukaynizi *et al.*, "Proactive identification of exploits in the wild through vulnerability mentions online," in *Proc. of CyCon-17*, 2017.

[54] C. Sabottke *et al.*, "Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits," in *Proc. of USENIX-15*, 2015.

[55] D. W. Woods and R. Böhme, "Systematization of knowledge: Quantifying cyber risk," in *Proc. of S&P-21*, 2021.

[56] Y. Liu *et al.*, "Cloudy with a chance of breach: Forecasting cyber security incidents," in *Proc. of USENIX-15*, 2015.

[57] A. Sarabi *et al.*, "Patch me if you can: A study on the effects of individual user behavior on the end-host vulnerability state," in *Proc. of PAM-17*, 2017.

[58] A. Arora *et al.*, "An empirical analysis of software vendors' patch release behavior: Impact of vulnerability disclosure," *Inf. Syst. Res.*, 2010.

[59] S. Clark *et al.*, "Moving targets: Security and rapid-release in firefox," in *Proc. of ACM-CCS-14*, 2014.

[60] A. Ozment and S. E. Schechter, "Milk or wine: Does software security improve with age?" in *Proc. of USENIX-06*, 2006.

[61] Y. Beres *et al.*, "Analysing the performance of security solutions to reduce vulnerability exposure window," in *Proc. of ACSAC-08*, 2008.

[62] B. C. Ezell *et al.*, "Probabilistic risk analysis and terrorism risk," *Risk Analysis*, vol. 30, 2010.

[63] L. Allodi and F. Massacci, "Security events and vulnerability data for cybersecurity risk estimation," *Risk Analysis*, vol. 37, 2017.

[64] R. Anderson *et al.*, "Measuring the changing cost of cybercrime," in *In Proc. of WEIS-19*, 2019.

[65] S. Dambra *et al.*, "Sok: Cyber insurance - technical challenges and a system security roadmap," in *Proc. of SSP-20*, 2020.

[66] ThaiCERT, 2019, https://www.thaicert.or.th/downloads/files/A_Threat_Actor_Encyclopedia.pdf. Accessed: 2020-06-01.

[67] S. Dashevskyi *et al.*, "TESTREX: a testbed for repeatable exploits," in *Proc. of CSET'14*, 2014.

[68] A. Agresti and B. A. Coull, "Approximate is better than "exact" for interval estimation of binomial proportions," *The American Statistician*, vol. 52, 1998.

[69] L. D. Brown *et al.*, "Interval Estimation for a Binomial Proportion," *Statistical Science*, vol. 16, 2001.

[70] Kaspersky, "Equation group: questions and answers," 2015. [Online]. Available: https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/08064459/Equation_group_questions_and_answers.pdf

[71] Novetta, "Operation blockbuster - unraveling the long thread of the sony attack," 2016, https://www.operationblockbuster.com/wp-content/uploads/2016/02/Operation-Blockbuster-Report.pdf. Accessed: 2020-10-01.

[72] ProjectZero, "0day "in the wild"," 2021, https://googleprojectzero.blogspot.com/p/0day.html. Accessed: 2021-05-15.

[73] Kaspersky, 2020, https://securelist.com/ie-and-windows-zero-day-operation-powerfall/97976/. Accessed: 2020-12-01.

[74] I. Stevanovic, "Operating system market share – bill gates is still alone at the top!" 2020, https://kommandotech.com/statistics/operating-system-market-share/. Accessed: 2021-01-04.

[75] FireEye, "Think fast: Time between disclosure, patch release and vulnerability exploitation - intelligence for vulnerability management, part two," 2020, https://www.fireeye.com/blog/threat-research/2020/04/time-between-disclosure-patch-release-and-vulnerability-exploitation.html. Accessed: 2020-10-01.

[76] L. Bilge and T. Dumitras, "Before we knew it: an empirical study of zero-day attacks in the real world," in *Proc. of ACM-CCS-12*, 2012.

[77] K. Nayak *et al.*, "Some vulnerabilities are different than others - studying vulnerabilities and attack surfaces in the wild," in *Proc. of RAID-14*, 2014.

[78] S. Dashevskyi *et al.*, "A screening test for disclosed vulnerabilities in FOSS components," *IEEE Trans. Software Eng.*, vol. 45, 2019.

[79] F. Massacci *et al.*, "Solarwinds and the challenges of patching: Can we ever stop dancing with the devil?" *IEEE Secur. Priv.*, vol. 19, 2021.

[80] A. Anwar *et al.*, "Cleaning the nvd: Comprehensive quality assessment, improvements, and analyses," *IEEE TDSC*, 2021.

[81] Y. Dong *et al.*, "Towards the detection of inconsistencies in public security vulnerability reports," in *Proc. of USENIX-19*, 2019.

[82] V. H. Nguyen *et al.*, "An automatic method for assessing the versions affected by a vulnerability," *Empirical Software Engineering*, 2016.

Giorgio Di Tizio received the MSc degree in computer science from the University of Trento, Italy. He is currently working toward the Ph.D. degree at the University of Trento. His interests include cyber threat intelligence and cybercrime. Contact him at *giorgio.ditizio@unitn.it*.

Michele Armellini is a master student in computer science from the University of Trento, Italy. He is currently working toward the M.Sc. degree at the University of Trento.

Fabio Massacci is a professor at the University of Trento, Italy, and Vrije Universiteit Amsterdam, The Netherlands. He received the Ten Years Most Influential Paper award by the IEEE Requirements Engineering Conference in 2015. He is the European Coordinator of the AssureMOSS project. Contact him at fabio.massacci@ieee.org.