Is 'deny access' a valid 'fail-safe default' principle for building security in cyber-physical systems?

Fabio Massacci, *Member, IEEE,*

Abstract—A landmark paper by Saltzer & Schroeder lists some timeless principles secure systems design. One of them is "systems should be fail safe" which security designers have always interpreted as 'deny access is always the fail-safe, secure default'. I argue that in the era of the Internet of Things, Cars, and Planes (and of recent plane crashes) such principles may no longer be sufficient.

1 Introduction

In 1975 Saltzer and Schroeder (S&S) enucleated eight design principles that shaped decades of security research and development [1]. Some are listed as key design principles for security protocols [2, §3.3], in software design methodologies such as Microsoft's STRIDE [3], [4], or in the 2004 opening article of the 'Building Security in' column by McGraw on the IEEE Security & Privacy Magazine [5, p.82].

As the world has changed, it is worth asking whether such principles (Reported in Table 1) are still sufficient for designing secure systems today. As pointed by Needham when rebutting the attacks against the protocol he co-authored with Schroeder [6], design principles are based on tacit assumptions that were *true then but possibly false now*.

If tacit assumptions fail to be true then principles based on those assumption might not be enough to obtain a good design. Unfortunately you'll never find tacit assumptions in a paper (they are tacit!). You'll never find "Tacit Assumption 0 $\sim$ XX Century: All free websites need 3rd parties advertisements to survive". A paper would go straight to discuss Javascript injection. 50 years in the future (or 50 in the past) researchers would wonder why people would do such a weird thing as dynamically self-injecting code from strangers in one's own webpage ('borrowing programs' in S&S's terminology). Indeed S&S require such borrowing be preceded by individual off-line agreements and other measures that, if deployed today, would 'break the Internet'.

2 Fail-Safe for Information Use

I would like to start the discussion with the most seemingly obvious principle: *fail-safe default*: *if no one has specified otherwise, the default should be to deny access*.

A recent revision of S&S principles in 2012 by Smith [9, p. 24] casted this authentic interpretation by renaming the 'Fail-safe default' principle as 'Deny by default' . This notion of fail-safe has been used from classical security protocols [2] to modern incentives for distributed computations [7]. It is also well understood in access control [8] and software design [4]. What can be wrong about such principle?

'Fail-safe' is of course a platitude security requirement: who can ever require a system to fail unsafely? S&S's point was sensible because they had a *specific notion* of safe failure.

F. Massacci is with University of Trento, Italy, and CINI, Italy

Hypothesis 1: 'denied access' is a 'safe state'.
Observation 2: if we base system access on positive permissions, a user asks for a permission and fails to get it then we are back to the 'denied access' state
Conclusion 3: by Hyp. 1 that's a safe state so fail safely.

This was true (then) because there was a tacit goal:

Assumption 0: the goal of a system is to *use information*.

A couple of quotation from S&S would make this apparent:

> Access = The ability to *make use of information* stored in a computer system
> [. . .] applications involve both storing information and simultaneous use by several individuals. The key concern in this paper is *multiple use*. [. . .] The objective of a secure system is to prevent all *unauthorized use* of information, a negative kind of requirement. (From [1], my emphasis)

All examples but one in the 'General Principles' section of S&S are about access to information (actually about confidentiality). The only example of write permission is about entering and modifying one's own supplied data.

Since safety is defined as 'something bad will never happen', the fail-safe definition in Hyp. 1 is consistent with the safety definition when 'use of information' is the goal. Suppose (1) the system is providing Alice some information, (2) Alice would like to do something different, but (3) the procedure does not grant her additional access rights (by deliberate design or by unexpected failures). Whatever the reason, the outcome is that the system permits Alice to use the same information that she had before she made her (failed) request but no more. This cannot obviously be bad.

Is the reasoning still valid? I would challenge this belief because Assumption 0 may be false and increasingly so.

3 Fail-Safe for Command and Control

We are now using software *to control things* from starting the fridge to stopping the car. Access to information is one goal, not *the* goal. Allowing humans to stop an autonomous car seems unwise, as well as allowing them to disconnect and reboot the heaters in during the Finnish winter. Overriding the software piloting an airplane looks dangerous: several sensors do provide the right information to the autopilot.

Except that sensors do malfunction, the control software might not do the 'right' thing and failure to grant users more control (as opposed to more access) can be bad.

In Lappeenranta, the internet connection of a heating system was victim of a DDoS attack. The attack by itself did *not* shut down the system, it only prevented heaters to call 'mummy'. To mange the error, the heaters rebooted to try re-establishing a connection. As there was a DDoS attack, there would still be no connection home after a quick reboot. . . So, as the people living in the affected blocks discovered to their dismay, the system went into an endless reboot loop eventually shutting down the heating. Manual disconnection and re-connection within a firewall was the only choice after a freezing week [10].

What if the heater software decided that manual access was not acceptable to start from a pristine state and you really needed a access from home? Things may have been worse, and you might have ended up on a deadly flight as the unfortunate crews and passengers of the Boeing Max planes of Lion Air and Ethiopian Airlines [11].

As the FT's journalist Brooke Masters put: "The Lion Air pilots' desperate struggle with the anti-stall software [to gain control over it] holds critical lessons for car makers experimenting with self-driving technologies" [12]. The Ethiopian Airlines' pilots tried to obtain administrative privileges over the stall control software at 05:43:11 of March 10, 2019. [13, p.12]. In IT terms they tried a 'sudo pull up -f', but the software thought otherwise: at 05:43:20 administrative override was denied and, still mislead by faulty sensors, the software made the EA airplane dive down into its demise.

In cyber-physical systems users asks permission to control objects (typically to override existing behavior). If the permission granting process fails, the object ends in a 'perfectly fine' state (from the software's view point) that is also a 'out of control' state (from the user's perspective).

Pilots would clearly need a lot of access when the systems under their control are misbehaving. In the simplest of cases, the pilot in command could always take control. As vehicles get driverless, design problems will become harder. For example, we can imagine a future where there is an experienced airline (or manufacturer) pilot will be on duty somewhere 24/7 who can take over the control of an aircraft in trouble by an always available satellite link and fly it to safety. Yet, we also like to avoid that the control is not taken over towards a terrorist-driven crash.

4 Conclusions

Let's revise the reasoning behind positive permissions:

Assumption 0 $\sim$ XXI Century: The goal of a cyber system can be either to *use information* or to *control physical objects*.
Observation 4: if we base system access on positive permissions and fail to actively grant a permission then we fall back to a 'deny access' *and* 'deny control' state.
Question 5: Are we sure that 'deny control' is a 'safe state'?

Deny access by default might not be so safe after all. Indeed, the DHS catalogue of security measures [14] indeed requires that "The organization selects an appropriate failure mode (e.g., fail open or fail close)" but doesn't say how to select it. Software design methodologies for IoT and autonomous vehicles have not yet a design guidance on what should make 'Deny by default' a safe or unsafe decision (See [15] for a review). Specific studies on cyber-physical systems, e.g. [16, §2.1], refer back to NIST standards which go in length on the principle of restricting access (e.g. four out of six major security objectives are about restricting access [17, p. 11]) but cursorily address fail- safe procedures, except to tell us that they should exist [**?**, p. 78]. How such mystic fail-safe procedures should interact with detailed instructions on restricting access is left to the reader. . .

The S&S principals were designed for information systems and do not by themselves generate a fail-safe system in the command and control sense. To build secure software for cyber-physical system, i.e. *the* systems in our future, we might need to extend them. Let us know what you think.

Aknowledgements

I would like to thank Richard Clayton, Sandro Etalle, Virgil Gligor, and Mike Schroeder for useful comments that greatly improved this paper. Any remaining error is mine. This work is partly supported by the European Unions Horizon 2020 grants 830929, CyberSec4Europe - cybersec4europe.eu. and 770138, OPTICS2 - www.optics-project.eu.

References

[1] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.
[2] L. Gong, "Fail stop protocols: An approach for designing secure protolcols," Computer Science Laboratory SRI International, Menlo Park California, Tech. Rep., 1994.
[3] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack, "Threat modeling-uncover security design flaws using the STRIDE approach," *MSDN Magazine-Louisville*, pp. 68–75, 2006.
[4] M. Howard and D. LeBlanc, *Writing secure code*. Pearson Education, 2003.
[5] G. McGraw, "Software security," *IEEE Security & Privacy*, vol. 2, no. 2, pp. 80–83, 2004.
[6] R. M. Needham, "The changing environment for security protocols," *IEEE Network*, vol. 11, no. 3, pp. 12–15, 1997.
[7] S. Azouvi, A. Hicks, and S. J. Murdoch, "Incentives in security protocols," in *Proc. of Security Protocols Workshop (SPW-2018)*. Springer, 2018.
[8] G. Edjlali, A. Acharya, and V. Chaudhary, "History-based access control for mobile code," in *Secure Internet Programming*. Springer, 1999, pp. 413–431.
[9] R. E. Smith, "A contemporary look at Saltzer and Schroeder's 1975 design principles," *IEEE Security Privacy*, vol. 10, no. 6, pp. 20–25, Nov 2012.
[10] R. Chirgwin, "Finns chilling as DDoS knocks out building control system," *The Register*, Nov 2016.
[11] S. Pfeifer and P. Waldmeir, "Parked planes at Boeing plant underscore woes," *The Financial Times*, Apr 2019.
[12] B. Masters, "Boeing crashes hold important lessons for carmakers," *The Financial Times*, Apr 2019.
[13] Aircraft Accident Investigation Bureau, "Aircraft accident investigation preliminary report ethiopian airlines group b737-8 (max)," Federal Democratic Republic of Ethiopia, Ministry of Transport, Tech. Rep. AI-01/19, March 2019.
[14] National Cyber Security Division, "Catalog of control systems security: Recommendations for standard developers," Department of Homeland Security, Tech. Rep., April 2011.
[15] L. Williams, "Secure software lifecycle," in *The Cyber Security Body Of Knowledge*, 2019, to appear.
[16] A. Cardenas, "Cyber-physical systems security," in *The Cyber Security Body Of Knowledge*, 2019, to appear.
[17] K. Stouffer, J. Falco, and K. Scarfone, "Guide to industrial control systems (ICS) security," *NIST special publication*, vol. 800, no. 82, pp. 16–16, 2011.

TABLE 1
Saltzer and Schroeder Design Principles. Extract from [1]

Principle	Description
Economy of mechanism	Keep the design as simple and small as possible. [. . .] design and implementation errors that result in unwanted access paths will not be noticed during normal use (since normal use usually does not include attempts to exercise improper access paths). As a result, techniques such as line-by-be inspection of software and physical examination of hardware that implements protection mechanisms are necessary. For such techniques to be successful, a small and simple design is essential.
Fail-safe defaults	Base access decisions on permission rather than exclusion [by E. Glaser in 1965]. [. . .] The default situation is lack of access, and the protection scheme identifies conditions under which access is permitted. [. . .] A design or implementation mistake in a mechanism that gives explicit permission tends to fail by refusing permission, a safe situation, since it will be quickly detected. On the other hand, a design or implementation mistake in a mechanism that explicitly excludes access tends to fail by allowing access, a failure which may go unnoticed in normal use.
Complete mediation	Every access to every object must be checked for authority. [. . .] [This principle] forces a system-wide view of access control, which in addition to normal operation includes initialization, recovery, shutdown, and maintenance. It implies that a foolproof method of identifying the source of every request must be devised. It also requires that proposals to gain performance by remembering the result of an authority check be examined skeptically. If a change in authority occurs, such remembered results must be systematically updated.
Open design:	The design should not be secret [by P. Baran in 1964]. The mechanisms should not depend on the ignorance of potential attackers, but rather on the possession of specific, more easily protected, keys or passwords. This decoupling of protection mechanisms from protection keys permits the mechanisms to be examined by many reviewers without concern that the review may itself compromise the safeguards. [. . .] Finally, it is simply not realistic to attempt to maintain secrecy for any system which receives wide distribution.
Separation of privilege:	Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key. [by R. Needham in 1973]. The reason is that, once the mechanism is locked, the two keys can be physically separated and distinct programs, organizations, or individuals made responsible for them. From then on, no single accident, deception, or breach of trust is sufficient to compromise the protected information.
Least privilege	Every program and every user of the system should operate using the least set of privileges necessary to complete the job. Primarily, this principle limits the damage that can result from an accident or error. It also reduces the number of potential interactions among privileged programs to the minimum for correct operation, so that unintentional, unwanted, or improper uses of privilege are less likely to occur. [. . .] The military security rule of need-to-know is an example of this principle.
Least common mechanism	Minimize the amount of mechanism common to more than one user and depended on by all users [by G. Popek in 1974] Every shared mechanism (especially one involving shared variables) represents a potential information path between users and must be designed with great care to be sure it does not unintentionally compromise security. Further, any mechanism serving all users must be certified to the satisfaction of every user, a job presumably harder than satisfying only one or a few users. For example, given the choice of implementing a new function as a supervisor procedure shared by all users or as a library procedure that can be handled as though it were the users own, choose the latter course.
Psychological acceptability	It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly. Also, to the extent that the users mental image of his protection goals matches the mechanisms he must use, mistakes will be minimized. If he must translate his image of his protection needs into a radically different specification language, he will make errors.