



Trent Jaeger
Associate Editor in Chief

Toward Fail Safety for Security Decisions

I was thinking about what I wanted to write about for my first editorial in *IEEE Security & Privacy*, when I came across this statement in the 2019 Application Security Risk Report by Micro Focus¹ (free, but it requires registration): “Because people are prone to error, manual security tasks that can otherwise be automated are at risk of being done incorrectly.” While this is not a surprising statement in and of itself, it reminded me of the many ad hoc decisions being made by people in various roles in computing systems—from programmers to administrators to end users—that we require to be correct for a system to achieve security goals.

I normally work and teach topics related to operating systems and software security, where we aim to build methods to detect and/or resolve security problems in current systems. As a result, our aim is to automate security decisions as much as is practical. However, with increasing frequency, we are running into cases where systems request manual security decisions, but these often complex and ad hoc decisions are being made by people with little guidance and essentially no safety net. While this may not be a surprise to people who work on usable security or social engineering, a question is this: How can researchers help systems avoid creating vulnerabilities resulting from such manual decisions?

Examples of How User Decisions Impact Security

To provide some context in which to discuss the prevalence and variety of ad hoc manual security decisions that must be made, let me give you a brief review of some examples (from my research and teaching) of the types of these decisions that we depend upon people to make correctly. Mobile systems often depend

on users to authorize security-sensitive access requests made by third-party apps. However, such decisions may put users’ privacy at risk. Consider the problem of authorizing whether a third-party app can use a device sensor (such as a camera or microphone). Mobile systems lack insight into the functionality that users expect from third-party apps, probably because they want lots of these apps to be developed for their platforms. However, the result is that they depend on users to authorize security-sensitive accesses with little or no guidance about the potential risks. For example, once an app has been granted access to a sensor, it may use that sensor freely in many cases. One type of malware, called a *remote access Trojan*, exploits this approach to obtain access to use device sensors to spy on users.

We have proposed methods to help users limit the misuse of sensors by restricting the freedom apps have to exploit such manual authorizations.^{9,10} However, risks remain as long as the system design does not account for potential abuses systematically.

In addition, authorizations that users are asked to make in mobile systems may have unforeseen side effects. When a user grants an Android permission to an app, some of these permissions may result, invisibly to the user, in changes to the file system’s access control policy. Even the Android original equipment manufacturers (OEMs) may have difficulty foreseeing the implications of such side effects. We recently found vulnerabilities caused indirectly by users granting Android permissions that permit malicious apps to modify critical files and compromise privileged services.⁷ In this case, manual user decisions interact with the permissions configured manually by Android OEM administrators to create exploitable situations that were not foreseen.

The first two examples highlight the manual decisions made by end users primarily, but of

Digital Object Identifier 10.1109/MSEC.2021.3096614
Date of current version: 28 October 2021



course, the origins of many risks are the results of ad hoc manual decisions made by programmers. I teach a course in secure software development, so I see inexperienced programmers who frequently use unsafe application programming interfaces (APIs; for example, C string functions). These students are often unaware of why those APIs are unsafe, much less of how to produce safe code. Some interactive development environments identify problems and replacements, but clearly inexperienced programmers have more flexibility in using unsafe languages than they should. And, undoubtedly, this is only the tip of the iceberg; there remain many decisions that may impact security even for type-safe programming environments.

Unfortunately, all of these types of ad hoc security decisions appear in the emerging Internet of Things (IoT). First, IoT systems depend on users to make decisions about authorizing IoT apps to access devices, which may grant malicious apps the ability to eavesdrop.⁴ Second, researchers have found that determining who has the authority to make security decisions regarding changes in permissions is exacerbated in multiuser IoT settings.¹³ Third, IoT users are allowed to make unsafe programming decisions, such as in the form of trigger-action rules, which have been found to lead to errors.⁸ In IoT systems, end users may make ad hoc decisions in the roles of programmer, administrator, and end user at one time.

One could argue that these are weaknesses introduced because the

developers of systems “punt” critical security decisions to their users, be they programmers, administrators, or end users. However, while, ideally, the developers should prevent these situations, it may be inevitable that some manual decisions that impact security may be delegated to users, especially for user-configured systems, such as mobile and IoT systems. A question we examine is: If there will be manual decisions that may impact systems security, how should we build systems to prevent those decisions from becoming exploitable?

A Design Goal: Fail-Safe Decisions

Saltzer and Schroeder suggested that secure systems provide fail-safe defaults to prevent ad hoc security decisions from violating a system’s security goals.¹¹ By fail-safe defaults, they argue that we should “base access decisions on permission rather than exclusion.” That is, we should have an approach that expresses the permissions that are granted rather than those that are denied. Fundamentally, the problem with the latter is that if we make a mistake in choosing which privileges to deny, then an error may lead to unauthorized access. In the former case, a mistake will result in denied access, which will not violate a security goal (thus, it is “safer”).

A problem is that in complex systems a false denial may cause users to take countermeasures that reduce the effectiveness of defenses. I have been told how people personally disable security features

Executive Committee (ExCom) Members: Carole Graas, President; Christian Hansen, Sr. Past President; Jeffrey Voas, Jr. Past President; Lou Gullo, VP Technical Activities; Carole Graas, VP Publications; Jason Rupe, VP Meetings and Conferences; Qiang Miao, VP Membership; Preeti Chauhan, Secretary; Steven Li, Treasurer

Administrative Committee (AdCom) Members: Carole Graas, Evelyn Hirt, Qiang Miao, J. Bret Michael, Jason Rupe, Daniel Sniezek, Loretta Arellano, Pierre Dersin, Lou Gullo, Yan-Fu Li, Nihal Sinnadurai, Robert Stoddard, Alex Dely, Donald Dzedzy, Ruizhi (Ricky) Gao, Z. Steven Li, Farnoosh Naderkhani, Charles H. Recchia

<http://rs.ieee.org>

The IEEE Reliability Society (RS) is a technical Society within the IEEE, which is the world’s leading professional association for the advancement of technology. The RS is engaged in the engineering disciplines of hardware, software, and human factors. Its focus on the broad aspects of reliability allows the RS to be seen as the IEEE Specialty Engineering organization. The IEEE Reliability Society is concerned with attaining and sustaining these design attributes throughout the total life cycle. The Reliability Society has the management, resources, and administrative and technical structures to develop and to provide technical information via publications, training, conferences, and technical library (IEEE Xplore) data to its members and the Specialty Engineering community. The IEEE Reliability Society has 28 chapters and members in 60 countries worldwide.

The Reliability Society is the IEEE professional society for Reliability Engineering, along with other Specialty Engineering disciplines. These disciplines are design engineering fields that apply scientific knowledge so that their specific attributes are designed into the system/product/device/process to assure that it will perform its intended function for the required duration within a given environment, including the ability to test and support it throughout its total life cycle. This is accomplished concurrently with other design disciplines by contributing to the planning and selection of the system architecture, design implementation, materials, processes, and components; followed by verifying the selections made by thorough analysis and test and then sustainment.

Visit the IEEE Reliability Society website as it is the gateway to the many resources that the RS makes available to its members and others interested in the broad aspects of Reliability and Specialty Engineering.



Digital Object Identifier 10.1109/MSEC.2020.3044408

to avoid fixing what they perceive to be false denials of functionality because security systems can be too difficult to debug. We certainly do not want users to disable security features as a resolution. Thus, a variety of systems “overpermission” their users and applications to avoid false denials. For example, it is difficult for developers to predict all the uses expected of IoT systems configured by users.⁸ This leaves an issue of how we can enable users to make security decisions that are fail safe.

One focus has been to enable users to make better decisions. Work that improves the usability of securing systems should enable users to produce decisions that are more likely to comply with security goals, even fail safety in some cases. Van Oorschot describes design principles and guidelines for usable security in his book *Computer Security and the Internet: Tools and Jewels (Information Security and Cryptography)*.¹² The principle of making “safe choices easy” advocates to “[d]esign systems with ‘paths of least resistance’ yielding secure user choices,” which encourages fail-safe defaults. However, as we have described, it may be difficult for programmers to predict safe defaults, leading to potential problems.

Researchers have long known how to enforce forms of fail safety. For example, capability systems have long proposed techniques to prevent the delegation or use of a capability that may lead to security errors^{5,6} by checking capabilities against a safety policy (for example, a mandatory access control policy) prior to delegation. However, these low-level mechanisms do not apply for the types of manual decisions we have discussed here. In addition, the number of manual decisions that would require some form of safety policy checks could be high and growing for domains like the IoT.

Using Fail Safety Effectively

We still lack effective approaches to manage the fail safety of manual decisions. One problem is that specifying fail-safety requirements may itself turn into yet another ad hoc security decision for people to make. For example, if we have to define fail safety specifically for each device deployment in each IoT system, then fail safety will not scale.

Since about the same time of Saltzer and Schroeder’s work,¹¹ researchers have envisioned safety based on information-flow security policies for secrecy³ and integrity.² Researchers have long envisioned that information-flow security should form a foundation for fail safety. Unfortunately, commercial systems and programs were not designed with information flow in mind, so legacy systems often do not comply.

However, some commercial systems are reducing the gap with respect to information flow. We recently found that the Android 11 system’s access control policy has only about 30 resources that may be involved in a violation of information-flow integrity,⁷ even when accounting for how users may grant Android permissions. In this case, we apply an automated analysis to detect when manual decisions may use unsafe resources or expand the number of unsafe resources relative to information-flow integrity requirements.

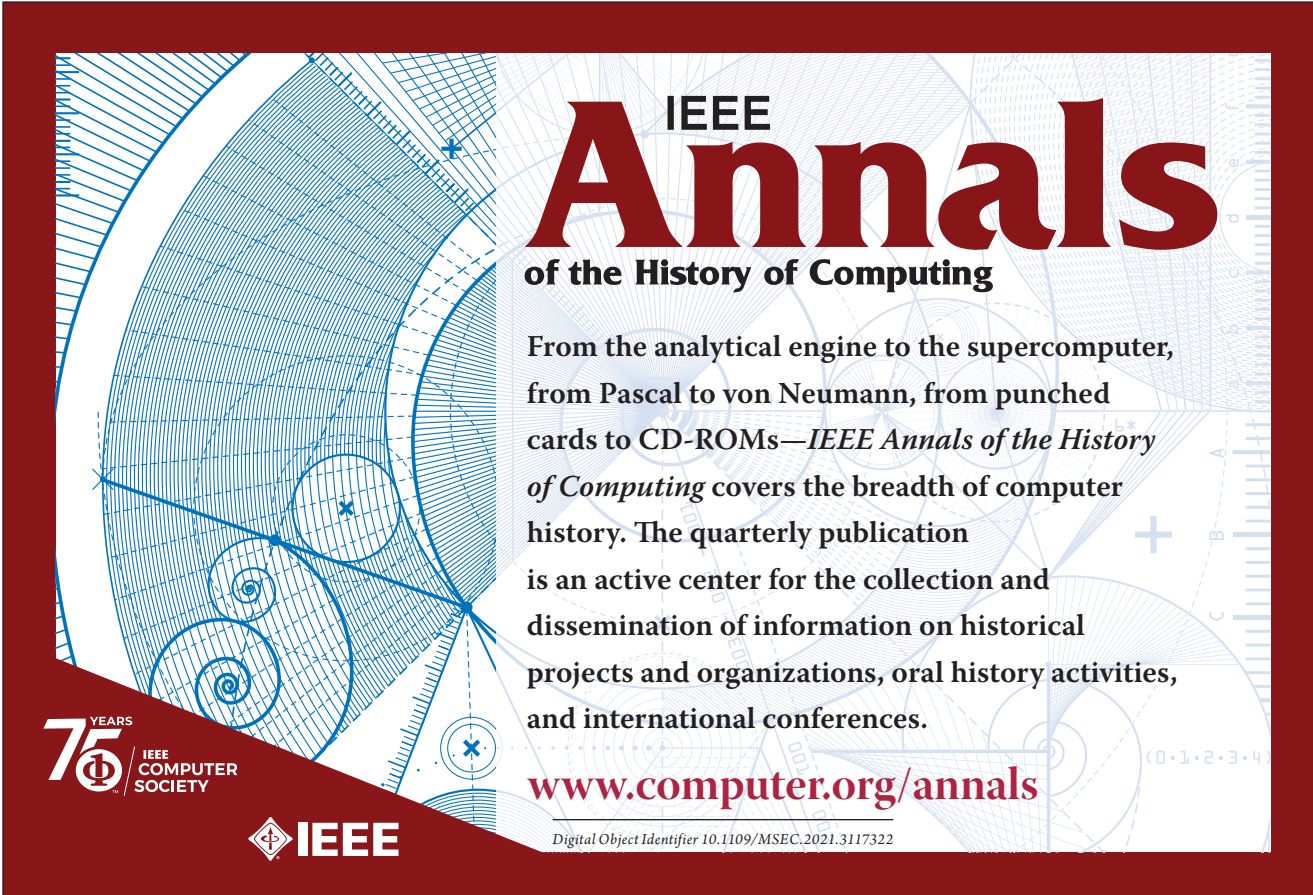
However, in IoT systems, we need to consider safety from additional perspectives. For example, IoT safety may be defined in terms of the physical properties of a system. Thus, we will likely need new principles for fail safety, but ideally these rules will be independent of a specific app and/or a specific deployment like information flow. In addition, we must consider safety for security decisions in IoT systems for manual decisions for the various roles of programmers, administrators, and end users.

Finally, another challenge is how to leverage fail-safety requirements effectively. As an analogy, consider the significant body of research in vulnerability detection and testing that evaluates artifacts against security requirements, some of which could be considered safety requirements. However, such evaluation methods are typically not integrated with the ecosystem of the artifact, limiting the utility of such techniques. Ideally, we can envision a risk analysis that identifies manual decisions whose outcomes may violate fail-safety requirements (such as information flow) to aid system developers in identifying which manual decisions may lead to problems. Then, we need to consider how to develop techniques that aid in manual decision making as an integral part of each system’s ecosystem to improve our ability to achieve fail safety.

As complex systems (including mobile and IoT systems) evolve, it will be interesting to see how researchers propose to build systems that help users make safe decisions and provide methods to validate those decisions against safety criteria. What safety criteria should be defined? Can researchers develop methods that can achieve or at least be measured in terms of fail safety? Can these methods account for the variety of safety problems associated with user decisions as programmers, administrators, and end users? How can methods to improve the security of user decision making be integrated effectively into ecosystems? How might we use artificial intelligence as an intelligent cybersecurity assistant to improve fail safety in such decisions? *IEEE Security & Privacy* welcomes the submission of articles on challenges in securing systems in each of these dimensions. ■

References

1. "2019 application security risk report," Micro Focus International PLC, Newbury, Berkshire, U.K., 2019. [Online]. Available: <https://www.microfocus.com/en-us/assets/security/application-security-risk-report>
2. K. J. Biba, "Integrity considerations for secure computer systems," MITRE, Bedford, MA, Tech. Rep. MTR-3153, 1977.
3. D. Denning, "A lattice model of secure information flow. *Commun. ACM*, vol. 19, no. 5, pp. 236–243, May 1976. doi: 10.1145/360051.360056.
4. E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *Proc. IEEE Symp. Security Privacy*, 2016, pp. 636–654.
5. L. Gong, "A secure identity-based capability system," in *Proc. IEEE Symp. Security Privacy*, 1989, pp. 56–63. doi: 10.1109/SECPRL.1989.36277.
6. P. A. Karger and A. J. Herbert, "An augmented capability architecture to support lattice security and traceability of access," in *Proc. IEEE Symp. Security Privacy*, 1984, pp. 2–12. doi: 10.1109/SP.1984.10001.
7. Y.-T. Lee et al., "Polyscope: Multi-policy access control analysis to triage android systems," in *Proc. 30th USENIX Security Symp.*, Aug. 2021.
8. M. Palekar, E. Fernandes, and F. Roesner, "Analysis of the susceptibility of smart home programming interfaces to end user error," in *Proc. IEEE Workshop on Internet Safe Things*, 2019, pp. 138–143. doi: 10.1109/SPW.2019.00034.
9. G. Petracca, Y. Sun, A.-A. Reineh, J. Grossklags, P. McDaniel, and T. Jaeger, "Entrust: Regulating sensor access by cooperating programs via delegation graphs," in *Proc. 28th USENIX Security Symp.*, Aug. 2019, pp. 567–584.
10. G. Petracca, A. Atamli-Reineh, Y. Sun, J. Grossklags, and T. Jaeger, "Aware: Preventing abuse of privacy-sensitive sensors via operation bindings," in *Proc. 26th USENIX Security Symp.*, 2017, pp. 379–396.
11. J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proc. IEEE*, vol. 63, no. 9, 1975, pp. 1278–1308. doi: 10.1109/PROC.1975.9939.
12. P.C. van Oorschot, "Computer security and the Internet—Tools and jewels," in *Information Security and Cryptography*. Berlin: Springer-Verlag, 2020. doi: 10.1007/978-3-030-33649-3.
13. E. Zeng and F. Roesner, "Understanding and improving security and privacy in multi-user smart homes: A design exploration and in-home user study," in *Proc. 28th USENIX Security Symp.*, 2019, pp. 159–176.



The graphic features a dark red background with a light blue technical drawing of a gear mechanism. The text is in white and red. The IEEE logo is at the bottom left, and the IEEE Computer Society logo is at the bottom right. The main title 'IEEE Annals of the History of Computing' is in large red letters. Below it, a paragraph describes the journal's content. At the bottom, the website URL 'www.computer.org/annals' is in red, and the Digital Object Identifier '10.1109/MSEC.2021.3117322' is in white.

IEEE Annals of the History of Computing

From the analytical engine to the supercomputer, from Pascal to von Neumann, from punched cards to CD-ROMs—*IEEE Annals of the History of Computing* covers the breadth of computer history. The quarterly publication is an active center for the collection and dissemination of information on historical projects and organizations, oral history activities, and international conferences.

www.computer.org/annals

Digital Object Identifier 10.1109/MSEC.2021.3117322

75 YEARS
IEEE
COMPUTER
SOCIETY

IEEE