State of the Systems Security

Eric Bodden
Heinz Nixdorf Institute at Paderborn University &
Fraunhofer Institute for Mechatronic Systems Design (IEM)
eric.bodden@upb.de

ABSTRACT

Software-intensive systems are increasingly pervading our everyday lives. As they get more and more connected, this opens them up to far-reaching cyber attacks. Moreover, a recent study by the U.S. Department of Homeland Security shows that more than 90% of current cyber-attacks are enabled not by faulty crypto, networks or hardware but by application-level implementation vulnerabilities. I argue that those problems can only be resolved by the widespread introduction of a secure software development lifecycle (SDLC). In this technical briefing I explain where secure engineering currently fails in practice, and what software engineers can do if they want to make a positive impact in the field. I will do so by explaining major open challenges in the field, but also by resorting to success stories from the introduction of SDLCs in industry.

KEYWORDS

Secure engineering, software engineering, systems security

ACM Reference format:
Eric Bodden. 2018. State of the Systems Security. In *Proceedings of 40th International Conference on Software Engineering , Gothenburg, Sweden, May 27-June 3, 2018 (ICSE '18 Companion),* 2 pages.
DOI: 10.1145/3183440.3183462

Software-intensive systems are increasingly pervading our everyday lives. Security vulnerabilities in those systems frequently open them up to far-reaching cyber attacks. The topic has recently captured the attention of policy makers. As one example, in its *Joint Communication on Resilience, Deterrence and Defence: Building strong cybersecurity for the EU*, the European Commission has confirmed Digital Security as a priority area in which urgent action is needed. [2] Recent devastating cyber-attacks have caused the estimated losses due to cybercrime to skyrocket to USD 600 billion annually [8], and call for the immediate introduction of strict secure-software-engineering standards. [4]

In this technical briefing I will explain where such standards already exist, why they are useful, and why they have to be standards not describing required security features of software products, but rather describing requirements for a secure software development lifecycle (SDLC). [6] This is because one can only develop software-intensive systems securely by following the principle of *security by design*, thinking security from the beginning to the end.

I will discuss current challenges and solutions for the secure engineering of software-intensive systems: Why are current systems as insecure as they are? What does it take to implement a Secure Software Engineering Lifecycle? How can one secure software architectures, which role does code analysis play? Which are current hot topics in the security community that we as software engineers should address? I will address those questions by referring to current security incidents and by explaining a state-of-the-art secure engineering lifecycle. In doing so, I will refer to hands-on experiences that I have gained in projects during which we introduced security engineering into major engineering companies. I will next briefly refer to those topics here.

1 HUMAN-CENTRIC SYSTEMS SECURITY

A major reason for why current software-intensive systems are as insecure as they are, is because those systems are engineered by humans for humans. Not only are the humans fallible and make mistakes, the processes in which they are involved often fail to avoid such mistakes, for instance by failing to acknowledge the problem in the first place, or by prioritizing time-to-market over security. While this may see like an obvious assessment at first, it also means that to solve the software security crisis it is insufficient to build the next big software tool. While tools certainly must be an important part of the solution, they can only be building blocks, or rather supporting structures in a secure development lifecycle.

To improve the state of the software-systems security throughout, we must understand how software architects and develop actually go about designing and implementing software systems, and how they can be helped to do so securely during the process. We must understand how customers administer those software systems, which security issues can arise at those levels and how administrators can be helped to avoid those issues. Last but not least, we must better understand how to provide long-term security, for instance by implementing efficient security-response cycles, and by providing architectures that allow the swift but controlled exchange of insecure components in the first place. Such research must put the human factor into the focus, which is why we call it *human-centric systems security.* [5]

2 PRESSING RESEARCH QUESTIONS

Given the above, an obvious question is what we as software engineers can do to aid the solution of the problem. I see major challenges in two areas: (1) empirical software engineering and (2) tool development across all levels of the SDLC.

ICSE '18 Companion, Gothenburg, Sweden

DOI: 10.1145/3183440.3183462

Empirical research. Empirical research is needed to better understand where and why security-critical mistakes arise within in the software development lifecycle. So far, first observations and publications have only painted a rough picture of this landscape. Further studies are needed to assess the positive impact (or negative side-effects) of mitigation strategies, and to assess the applicability of such strategies in different development methodologies, e.g., waterfall, V-model, SCRUM, DevOps, and so on. Based on those empirical findings, software engineers can then make good use of their skills by providing powerful tools that help avoid security-critical mistakes where they are known to happen.

Resilient architectures. For instance, at the architectural level it shows that current architectures are often monolithic, yet assign the same level of privilige to the entire monolithic executable. The result: if an attacker succeeds in "owning" part of that system, she owns it entirely at once. Hence we should provide architects with method and tools to decompose systems according to the principle of least privilege [10]. While this "distrustful decomposition" [3] cannot avoid attacks, it can greatly reduce the impact of attacks when they succeed, to the degree that the incentive for an attack is removed entirely.

Tool-assisted vulnerability discovery. At the level of implementations, developers require better tools for the automatic discovery of coding vulnerabilities. Static-analysis tools are still notorious for long running times and rather high false-positive rates, but even worse they do not integrate well into the software-engineering processes. Most developers simply don't enjoy using them, which leads to widespread tool abandonment. [7] Dynamic analysis tools are notorious for their bad coverage, i.e., low recall. The solution may lie in a hybrid combination of static and dynamic techniques, but also in using better user-interface concepts. While first pieces of research have shown first successes in those areas, more research is needed to reach tools that developers enjoy using.

Long-term security through safe remote updates. But software development does not end with the implementation. We need better tools to secure software systems as they are deployed and to update their code in the long run. While updates are easy to achieve on desktop PCs and smartphones, they are a major issue in real-time systems such as industrial control systems. Those systems are often designed with lifetimes of 20 years or more, yet frequently cannot be shut down even for minutes without generating losses in the order of millions of dollars. This is why right now most companies owning production plants feel much more threatened by possible problems caused by a security updates than they feel threatened by the potential of a targeted cyber attack. This is why software engineers should aim to devise solutions that can guarantee the safe deployment of security updates, even in strong real-time settings.

3 THE MATTER OF EDUCATION

Last but by no means least, software engineers professors must integrate secure engineering deeply into their teaching curriculum. The students of today are the software engineers of tomorrow, and still too many students leave universities today freshly trained in general software-engineering methodology but with little or no knowledge about an SDLC. Fortunately, first courses exist that have evolved over several years, and that others can build on. [1, 9]

Similarly, targeted trainings are required for the millions of software-engineering practitioners that are out their designing, implementing and maintaining software systems today. They must be aided by good tutorials, good tools, good documentation. This is by no means a simple task, and moreover one that is hard to do at scale. Nonetheless, it is essential if we really to want to secure software systems at the large scale.

REFERENCES

[1] Eric Bodden. 2017. Undergraduate course "Secure Software Engineering". https://www.hni.uni-paderborn.de/swt/. (2017).

[2] European Commission. 2017. Joint Communication on Resilience, Deterrence and Defence: Building strong cybersecurity for the EU. http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52017JC0450. (2017).

[3] Chad Dougherty, Kirk Sayre, Robert C Seacord, David Svoboda, and Kazuya Togashi. 2009. *Secure design patterns.* Technical Report. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

[4] European Union Agency for Network and Information Security (ENISA). 2017. Principles and opportunities for a renewed EU Cybersecurity Strategy. https://www.enisa.europa.eu/publications/enisa-position-papers-and-opinions/enisa-input-to-the-css-review-b/view. (2017).

[5] Thorsten Holz, Norbert Pohlmann, Eric Bodden, Matthew Smith, and Jörg Hoffmann. 2017. Human-Centered Systems Security - IT-Sicherheit fr NRW 4.0. http://www.it-sicherheit-nrw.de/. (2017).

[6] Michael Howard and Steve Lipner. 2006. *The security development lifecycle.* Vol. 8. Microsoft Press Redmond.

[7] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *Software Engineering (ICSE), 2013 35th International Conference on.* IEEE, 672–681.

[8] Net Losses. 2014. Estimating the global cost of cybercrime. *McAfee, Centre for Strategic & International Studies* (2014).

[9] Andrew Meneely. 2017. Undergraduate course "Engineering Secure Software" (SWEN-331). http://www.se.rit.edu/~swen-331/. (2017).

[10] Jerome H Saltzer and Michael D Schroeder. 1975. The protection of information in computer systems. *Proc. IEEE* 63, 9 (1975), 1278–1308.