DOI:10.1145/3386908

Adam Shostack and Mary Ellen Zurko

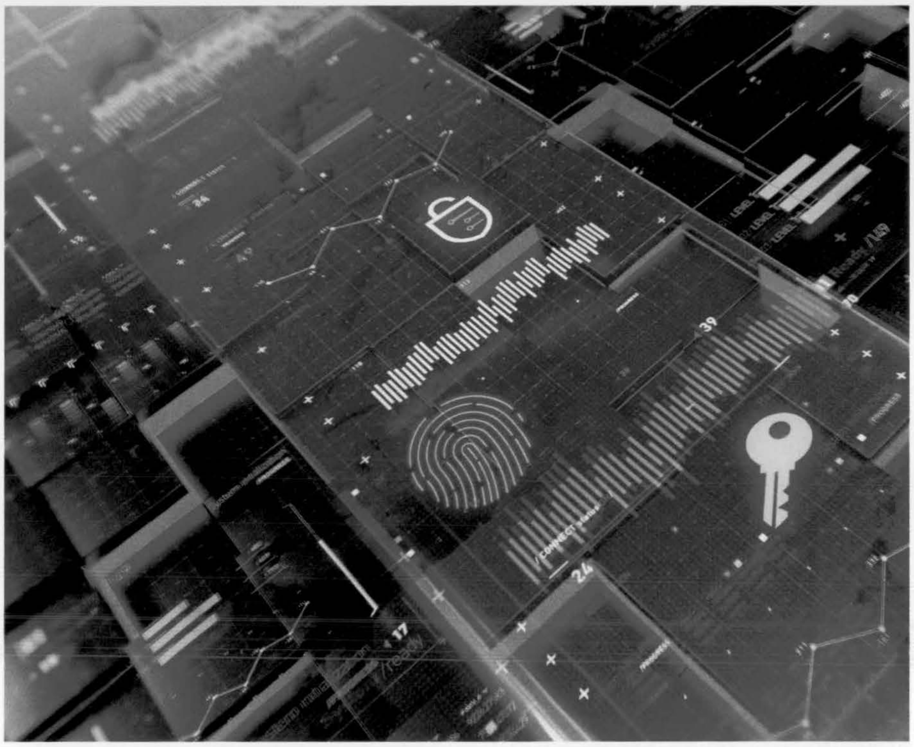
▸ Carl Landwehr, Column Editor

Privacy and Security

Secure Development Tools and Techniques Need More Research That Will Increase Their Impact and Effectiveness in Practice

Secure development is an important and pressing problem.

WRITING CODE THAT is secure, and provides security without vulnerabilities, is a critical challenge to cybersecurity. Writing code without vulnerabilities has long been at least as difficult as writing code without bugs. While there are many other potential sources of security exposures in software, developing code without known classes of vulnerabilities has always seemed like a tractable goal. It relies on human developers using tools, techniques, and processes to produce software that does not have particular known types of defects.



One of the most effective approaches—research into programming languages and tools—has yielded technologies that are shown to resist categories of vulnerabilities, largely by not allowing for them. Memory safe languages that manage memory allocation and deallocation, instead of requiring the programmer to do so, make it impossible for developers to create buffer overflow vulnerabilities and some other types of exposures, from missing array bounds checks, null pointer use, and data leakage via memory reuse. Thread-safe languages can address exposures where race conditions can be used to subvert security-related checks in the program.

Within the software development community, groups and organizations with a mission to develop software securely have incorporated tools and techniques into their software development life cycles to include a secure development life cycle. Early high-assurance software adopted formal methods to specify the security properties of the system, and code review to use humans to find such flaws at the coding level.[2] Microsoft created its Security Development Lifecycle adding root cause analysis, security education, threat modeling, specific secure coding requirements, and security testing that included penetration and fuzz testing. Practices tend to be adopted based on business need, perceived security impact, and fit with established or evolving development practices.

Research that impacts what works and what could work for secure development is needed. Current research seems to play an unfortunately limited role in creating, proposing, evaluating, and proving tools, techniques, and processes that are used in practice for secure development. In particular, research is rarely brought to bear directly on tools and techniques as they are used, in the context they are used. We need more research into the effectiveness and results of secure development tools, techniques, and processes. That research can be judged on its impact on how software development works in practice. Properties of research influence how likely it is to have that impact.

Rigor in research scientific experimentation calls for a number of process requirements, including a statement of the hypothesis being tested, controlling the variables of the experiment to ensure the experiment actually tests the hypothesis, and analyzing experimental data and outcomes to mathematically prove the hypothesis (or disprove the null hypothesis). While these processes can form the basis of important foundational research in secure development, they often avoid the messy realities involved in bringing a technique into practice, precisely because those messy realities complicate experimental design.

Negative research results that fail to prove a secure development technique increases security, while important to the research field, are not likely to impact secure development in practice. An early lesson as a security developer in a large technology company was that telling developers not to do something was almost always ineffectual, if it was not paired with the alternative that they could use to achieve the goal of the deprecated practice. "Don't roll your own crypto" has to come with the crypto library that should be used. Additionally, finding a tool or technique experimentally ineffective in producing security does not prove it is ineffective outside of the controlled experiment, in the larger, messier, more diverse context of software development.

What are some of the things being done in research that are hopeful for practical transfer into secure development? Two current trends in security research provide some hope for secure development. One is that secure development has emerged as a topic in security research conferences, covering topics such as evaluating developers' ability to use crypto securely and appropriately, evaluating tools to help developers avoid introducing vulnerabilities, and measuring developers' ability to code security-relevant functionality.

The other hopeful trend is artifact evaluation. A lot of software development builds on existing software, using frameworks, libraries, and open source. Offering an artifact used to establish and validate a research idea reduces the barriers to transfer of that idea into software development. Making code available through open source, with license terms friendly to reuse, can increase its potential for use. Some research incentives are shifting to encourage artifact submission as part of the research paper submission and publication process, at security conferences such as USENIX Security and ACSAC.

We need more research into the effectiveness and results of secure development tools, techniques, and processes.

Generalizing secure development research beyond the experiment is a challenge. A challenge of experimental research studies on secure development is the extent to which the results can generalize beyond the participants and context of the study. The challenges with increasing the similarities between an experimental study and secure development contexts may argue for an experimental approach closer to observational astronomy, medical case studies, or even public health than controlled laboratory physics experiments.

One of the aspects that complicates the design of a study evaluating a secure development process is the place of security in development tasks. As early usable security research called out, security is often not the primary goal of the user. Many of the human-centered empirical evaluation methods in use by research fit best for evaluating tools and methods in the context of explicit primary goals. In the secure development area, one of the aspects studied is avoiding the creation of vulnerabilities while coding, which is an implicit secondary goal. Prompting a coder to explicitly consider security has been shown to impact their behavior while writing code for the research study. Thus, studies that prompt the developer that way may not transfer to development contexts where coders are not told every hour to consider security for the code they are about to write, and we might guess that in the real-world developers would quickly tune out such messaging. However, remaining silent on the need for security provides less security prompting than occurs in organizations with a secure development process.

One exciting type of study balancing these concerns is the use of "Build It, Break It, Fix It" (BIBIFI) competitions[3] as a different type of research context to study secure development. In BIBIFI, teams compete over several weeks to build software according to a spec, gaining points for functionality and performance. Then they compete to break each other's software, causing the vulnerable teams to lose points. The context provides more control than a research field study, but more ecological validity than a smaller-scope lab study. The resulting performance of each team, in terms of points, coding, and testing, can be analyzed for insights into vulnerabilities in a context that considers vulnerability-free code as only one part of the overall task. The contest may be be part of curriculum requirements. Both that assignment and competition can act as motivators to keep participants engaged better and longer than most research studies.

Another complexity of research studies of secure development processes is effective recruitment of appropriate demographics. The expertise and skills of the participants potentially impact everything from what can be studied to what the limitations are on transferring the resulting findings to other contexts. Development expertise can be approximated by aspects such as years of experience in development, languages used, types of products, and types of organizations worked in. How to contextualize or measure security expertise of any particular developers, and in the general population of developers, remains an open question. Some research is emerging comparing the impact of demographic variables on the results of security task studies.

What more should be done? On the research side, there should be an explicit acknowledgment of the topic of research into the security results of secure development processes. The security research community should explicitly recognize that part of our responsibility as security researchers is to foster the full spectrum of research into better security: foundational research, practical research, and the transition of research into use (both successful and unsuccessful). A workshop venue for papers on security research and the challenges of tech transfer would be a solid step in identifying community and early work in the area.

Perhaps the largest barrier to such research is researcher access to secure development processes and their results. This requires cooperation with developers and development organizations. While each individual organization would profit from knowledge that would enable them to get the best results from their secure development process expenditure, getting there

Cross-community collaboration between researchers and development organizations is key to making progress.

would require a range of developers and organizations to cooperate with research, with no clear short-term upside. For inspiration on overcoming that, we look to near miss programs in aviation, which contribute to the safety of general aviation. One of these systems, the Aviation Safety Reporting System (ASRS) is comprised of confidential reporting, expert analysis by former pilots and air traffic controllers, publication of anonymized data, and rewards for those submitting reports. The rewards are that regulators are required to treat submission as "evidence of constructive engagement," and will reduce penalties on that basis. The ASRS is operated by NASA, a respected scientific agency, so reports are not sent to a regulator, such as the FAA.

There are proposals[1] for a near miss database for cyber. Mirroring the structure of ASRS, a scientific agency or FFRDC would collect confidential reports, analyze them, and publish lessons. Regulatory agencies would commit to giving consideration to companies who have programs that candidly report near misses. A near miss in cyber is a place where some controls function and others do not. So a spam filter might not stop an email message containing a phishing URL, and the click on the URL might be caught by a safe-browsing list or firewall. Being able to quantify the "misses" experienced in the field is in some ways analogous to public health data, and could make a case for various types of investigations in SDP. Voluntary and rewarded near miss reporting should encounter far less industry opposition (after all, it is voluntary). Questions of scope could be examined over time by looking at the variance between the voluntary responses. Because near misses give us data about both successes and failures, they represent a rich vein that we do not yet mine.

With such a capability, researchers could delve into the causes of near misses, and consider if the components of a SDP relate to important root causes. Root causes might be important for many reasons. They might be the most common problems, might be problems for which compensating controls are expensive, difficult, or ineffective. Researchers might argue, and have evidence for, other criteria.

Conclusion

Getting at the ability for researchers to evaluate secure development practices in context is a difficult problem, but critical for evaluating the ecological validity of practices in the wide variety of software development contexts that exist. Cross-community collaboration between researchers and development organizations is key to making progress.

The quality of software, including but not limited to security, is important to society as we become increasingly dependent on those qualities. How software development processes influence the qualities of software is thus an important societal question, worthy of study. As we improve the empirical evaluation of secure development processes as a result of these collaborations, we will benefit from a broad and deep approach to expanding our scientific inquiries. ■

References

1. Bair, J. et al. That was close: Reward reporting of cybersecurity near misses. *Colo. Tech. LJ 16* (2017), 327; http://ctlj.colorado.edu/?page_id=796#tabs-796-0-5
2. Lipner, S., Jaeger, T., and Zurko, M.E. Lessons from VAX/SVS for high-assurance VM systems. *IEEE Security and Privacy 10*, 6 (June 2012), 26–35; https://dl.acm.org/citation.cfm?id=2420631.2420857
3. Votipka, D. et al. Understanding security mistakes developers make: Qualitative analysis from Build It, Break It, Fix It. To appear in *Proceedings of the 29th USENIX Security Symposium (USENIX) Security 20*, 2020.

Adam Shostack (adam@shostack.org) is President of Shostack & Associates, a consultancy in Seattle, WA, USA.

Mary Ellen Zurko (mez@alum.mit.edu) is a member of the Technical Staff, MIT Lincoln Laboratory, Lexington, MA, USA.

The authors' views expressed here are not necessarily those of their employers.