DOI:10.1145/2822513

Steven B. Lipner

Privacy and Security

Security Assurance

How can customers tell they are getting it?

SECURITY ASSURANCE HAS been the most challenging topic I have had to contend with in my 45 years working on computer security. While designing and building security features for user identification and authentication or for access control can be challenging, the most difficult task is assurance: making systems that can resist attack. Assurance affects not only security features but also any system component that will respond to untrusted input.

Assurance is achieved by integrating security into the process of designing, building, and testing systems. If security is well integrated into the development process, the resulting software will generally resist attack. In this column, I summarize my experience building a process for security assurance and how and why it works. I also describe providing customers with confidence a system achieves an appropriate degree of assurance.

Creating a Process

While I started working on problems of assurance in the early 1970s, my perspective on assurance today is especially influenced by my 10 years of experience as the director of the team responsible for Microsoft's Security Development Lifecycle (SDL). The story of Microsoft's work to integrate security assurance into development has been told in the trade press and professional papers.[2,5,7] The company stepped up its focus on security in late 2001 in response to its concerns about the security of Microsoft products and to customer feedback—much of it heated—about the security vulnerabilities exploited by Code Red and Nimda worms.

It was clear the company's security challenges were not limited to security features but encompassed any component that dealt with potentially untrusted input. For that reason, Microsoft stopped *all* development on major products—Windows Server 2003 was the largest. This was a major commitment—it involved thousands of engineers, delayed the release of products by months, and impacted costs and revenues—but it was clear that an error handling untrusted input could occur anywhere.

Our team trained the entire development staff on techniques we believed would help improve products' resistance to attack, and required the developers to apply those techniques to the code for which they were responsible. Developers used tools and code review to search for vulnerabilities, changed defaults to reduce attack surface, and added mitigations that made it more difficult to exploit any vulnerabilities that might remain. This initial approach—the security push—reduced the prevalence and severity of security vulnerabilities in products that had gone through security pushes. It also motivated developers to build secure software.

The Slammer and Blaster worms of 2003 affected product versions that had not yet gone through security pushes and gave Microsoft clear evidence that security needed to be an integral part of its development process and culture. In 2004, Microsoft introduced SDL as a mandatory process for software development. Those of us who worked on the SDL knew security must be built into products and the best approach was to integrate security into the development workflow. In summary, the SDL required that threat models be produced as a product was being designed, that specific coding requirements be met and static analysis and attack surface analysis tools be applied as the code was being written, that mitigations be integrated into software, and that security testing (for

IMAGE BY ALPHASPIRIT

example, fuzz testing) be conducted as part of product qualification.

While the initial Security Development Lifecycle was released with little in the way of supporting tools, Microsoft updated the process to make it more effective and efficient. The SDL team built and mandated tools to support security assurance in design, development, and testing. We treated product vulnerability reports as feedback that told us whether we needed to update the SDL or do a better job of executing it. We introduced a tracking system that automated the task of ensuring product components had met the applicable requirements. The tools and tracking help ensure the SDL is followed, and provide a consistent mechanism for engineers.

By 2005–2006, the customers who had been providing heated feedback realized the company was serious about security and the SDL. Interactions with chief security officers (CISOs) became more cordial: in some cases, they urged other software vendors to adopt processes similar to the SDL. Whether because of that urging or because they had applied similar logic to ours, during the mid-2000s a number of other vendors developed processes similar to the SDL and adoption of such processes across the industry has continued to grow.[a]

Is a Process Enough?

When customers ask about secure development, they do so because they need products that will resist attack. Ideally a metric would enable customers to compare products without having to delve into the way products were built. Unfortunately, the quest for a measure of product security has been futile, and I expect it will continue to be so. We can measure the number and severity of reported product vulnerabilities, but when we do that, we are measuring the talents and interests of vulnerability researchers as well as inherent product security. A product that no one uses may have no reported vulnerabilities, but be extremely vulnerable if attacked.

Customers understand that measuring security is infeasible so they ask about a developer's process. They also understand that vulnerabilities will occur—both because processes are not perfect and because new attacks get discovered—so they want a process that incorporates continuous improvement.

So why do the SDL and similar processes improve product security? I believe an effective process has to meet three tests:

- *It must be fundamentally sound*. It must incorporate measures that will improve security if applied. Fundamental soundness implies continuous improvement: as new classes of vulnerabilities or techniques for building more secure software are discovered, the process must be updated.
- *It must be adapted to the development organization*. Individual developers must be trained and motivated (security must be part of the culture), elements of the process must be integrated into development, and development tools must reflect the requirements of the process. For example, if the process says, "do static analysis," the development organization must select a static analysis tool, "tune" it to their code base and identify "must-fix" errors.
- *It must affect the code*. Threat models result in a set of work items that must be addressed so the delivered code responds correctly to threats. Static analysis tools or fuzzers find errors that must be analyzed and fixed in the code.

The SDL meets all three tests but differ-

a For a non-exhaustive list of companies that have adopted SDL-like processes, see http://www.safecode.org.

entiates itself by meeting the third. The industry has many processes—CMMi is one example—that stop with documentation. If a process does not actually affect code, it is just paper.

How Can Customers Tell?

If customers accept the proposition that an SDL-like process is effective, they are still left with a question whether their suppliers are actually implementing such a process in a way that affects delivered code. A supplier should be able to describe their secure development process, and in particular how the process affects code. In practical terms, this means describing not just process generalities, but what specific tools are used, what errors are "must fix," and how the results of security analysis are managed in the supplier's work item management system.

Some customers have documented their expectations for their suppliers' secure development processes, and put "teeth" into those expectations by having in-depth interviews with suppliers' secure development staffs.[b] This is a good approach although it has scaling problems: suppliers' secure development staffs will not scale to participate in interviews with every customer and not every customer has the resources to interview every supplier. The customer's interviewers must have hard-to-find expertise to evaluate what they are hearing and to consistently assess processes across a range of suppliers. But the experience of such interviews appears beneficial—it informs customers and provides useful feedback to suppliers.

A new ISO standard—ISO 27034-1: Information technology—Security techniques—Application security—recognizes the importance of developer process and of ensuring delivered code reflects the requirements of the process.[3] The standard is still evolving, but I believe it will provide a solid basis for consistently assessing suppliers' secure development processes. The final standard is planned to address all three of the tests listed previously, including the link between process and delivered code. Some customers have expressed an interest in using the standard to drive procurement decisions, supplanting vendor interviews.

b For one example, see Third Party Software Working Group, Appropriate Software Security Control Types for Third Party Service and Product Providers, Financial Services Information Sharing and Analysis Center; http://bit.ly/1Fw6jhs.

Perfect assurance would be great, but we do not have the tools or techniques to achieve it, and I do not see any reason we ever will.

Some Things That Don't Help

When I started working in computer security, I expected we would formally verify that a secure system corresponded to a mathematical model, and that the system's security was correct down to the source code; this approach failed. Formal verification was not able to cope with systems large enough to be useful and no customers wanted the systems that were simplified to enable that approach to be applied.[6]

Earlier this year, Dorothy Denning wrote a *Communications* Privacy and Security column advocating legal liability for developers who released products vulnerable to attack.[1] Unfortunately, the list of such developers is "all of them." If a liability regime were put in place, I expect it would result in a lot of lawsuits against software developers and slow software innovation as developers attempted legally rather than technically defensive coding. If the goal is to create software that is more secure and usable by customers, encouraging the use of best practices that address threats is a more rational approach than liability.

Denning's proposal includes an "out" for developers who release the source code for their products. That "out" might incentivize developers to release their source code and change some business practices, but I do not believe it would improve assurance. Fifteen years ago, some researchers argued that open source software was inherently more secure than closed, but faced with reality that argument faded away.[8] The recent spate of vulnerabilities in the open source OpenSSL package is one example: the Linux Foundation's Core Infrastructure Initiative is bringing developers and users of open source software together with the aim of introducing secure development practices—many similar to the SDL—to widely used open source software.[4]

What Next?

Security assurance is a challenge for developers and a necessity for customers. Perfect assurance would be great, but we do not have the tools or techniques to achieve it, and I do not see any reason we ever will. We can achieve practical assurance if we commit to practical measures and apply them.

- Secure development processes work if they reflect continuous improvement and are followed by the people who design, implement, and test code.
- Assessment of secure development processes is feasible, but it will take a lot of work and it must consider impact on delivered code, not merely process. ISO 27034 represents a way forward here.
- Neither product liability nor open source will serve as a "silver bullet" that substitutes for secure development processes that are rigorously applied and continuously improved.

References

1. Denning, D.E. Toward more secure software. *Commun. ACM 58*, 4 (Apr. 2015), 24–26.
2. Howard, M. and Lipner, S.B. *The Security Development Lifecycle.* Microsoft Press, 2006.
3. ISO/IEC, ISO/IEC 27034-1:2011. Information technology—Security techniques—Application security—Part 1: Overview and concepts; http://www.iso.org/iso/catalogue_detail.htm?csnumber=44378.
4. Linux Foundation. Core Infrastructure Initiative site; https://www.coreinfrastructure.org/.
5. Lipner, S.B. The trustworthy computing security development lifecycle. In *Proceedings of the Twentieth Annual Computer Security Applications Conference* (Tucson, AZ, 2004).
6. Lipner, S.B., Jaeger, T., and Zurko, M.E. Lessons from VAX SVS for high assurance VM systems. *IEEE Security and Privacy* (Nov.–Dec. 2012).
7. Microsoft Corporation. Life in the Digital Crosshairs, 2014; http://bit.ly/1NnOoS4.
8. Panel: Security and Source Code Access: Issues and Realities. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2000.

Steven B. Lipner (lipner@outlook.com) retired in 2015 as Partner Director of Software Security, Trustworthy Computing, at Microsoft Corporation. He continues to consult with and advise organizations on computer security and assurance.