2.6. Process Improvement

In reviewing their experience in the CVD process, participants should capture ideas that worked well and note failures. This feedback can be used to improve both the Software Development Lifecycle and the CVD process itself.

The CVD process can create a pipeline for regular patching cycles and may reveal blocking issues that prevent a more efficient software patch deployment mechanism. A successful program provides the vendor with a degree of crowdsourcing for security research and testing of its products. However, CVD should be considered complementary to a vendor's internal research and testing as part of the Software Development Lifecycle, not as a wholesale replacement for internally driven security testing.

CVD and the Security Feedback Loop

A successful CVD program feeds vulnerability information back into the vendor's Software Development Lifecycle. This information can result in more secure development processes, helping to prevent the introduction of vulnerabilities in the first place.

Yet the reality of today's software is that much of its legacy code was not originally produced within a secure development process. Andy Ozment and Stuart Schechter studied the impact of legacy code on the security of modern software and how large code changes might introduce vulnerabilities [1]. The positive news is that *foundational vulnerabilities*—ones that existed in the very first release and carried through the most recent version of the software—decay over time. We can find them, fix them, and make the code base stronger overall. However, the bad news is that as the low-hanging fruit of foundational vulnerabilities are fixed, the remaining foundational vulnerabilities tend to be more subtle or complex, making them increasingly difficult to discover.

Furthermore, ongoing development and code changes can introduce new vulnerabilities, making it unlikely for the security process to ever be "finished." Even with modern architecture development and secure coding practices, software bugs (and in particular security vulnerabilities) remain a likely result as new features are added or code is refactored. This can happen for many reasons, not all of them technical. A recent article highlighted the difficulty of getting teams of people to work together, resulting in poor software architecture [2]. While the authors were primarily concerned with maintainability and performance, bugs (and particularly security vulnerability bugs) are an important side effect of inadequate architecture and teamwork process.

Another possibility is that, even with good internal processes and teamwork, no software model or specification can comprehensively account for the variety of environments the software may operate in [3]. If we cannot predict the environment, we cannot predict all the ways that things may go wrong. In fact, research has shown that it appears impossible to model or predict the number of vulnerabilities that may be found through tools like fuzzing—and, by extension, the number of vulnerabilities that exist in a product [4,5]. The best advice seems to be to assume that vulnerabilities will be found indefinitely into the future and work to ensure that any remaining vulnerabilities cause minimal harm to users and systems.

A successful CVD process helps encourage the search for and reporting of vulnerabilities while minimizing harm to users. Developers supporting a successful CVD process can expect to see the overall security of their code improve over time as vulnerabilities are found and removed.

Improving the CVD Process Itself

Feeding lessons learned back into the development process, CVD can:

- reduce creation of new vulnerabilities
- increase pre-release testing to find vulnerabilities

Participation in CVD may allow discussions between your developers and security researchers on new tools or methods for vulnerability discovery such as static analysis or fuzzing. These tools and methods can then be evaluated for inclusion in ongoing development processes if they succeed in finding bugs and vulnerabilities in your product. Essentially, CVD can facilitate field testing of new analysis methods for finding bugs.

< 2.5. Ethical Considerations | 2.7. CVD as a Wicked Problem >

References

1. A. Ozment and S. E. Schechter, "Milk or wine: Does software security improve with age?" in *USENIX Security*, 2006.
2. K. Matsudaira, "Bad Software Architecture Is a People Problem," *Communications of the ACM,* vol. 59, no. 9, pp. 42-43, September 2016.
3. J. M. Wing, "A Symbiotic Relationship Between Formal Methods and Security," in *Proceedings of the Conference on Computer Security, Dependability and Assurance: From Needs to Solutions*, 1998.
4. E. Bobukh, "Equation of a Fuzzing Curve — Part 1/2," 18 December 2014. [Online]. Available: https://blogs.msdn.microsoft.com/eugene_bobukh/2014/12/18/equation-of-a-fuzzing-curve-part-12/. [Accessed 23 May 2017].

5. E. Bobukh, "Equation of a Fuzzing Curve — Part 2/2," 6 January 2015. [Online]. Available: https://blogs.msdn.microsoft.com/eugene_bobukh/2015/01/06/equation-of-a-fuzzing-curve-part-22/. [Accessed 23 May 2017].