Editors: James A. Whittaker, jw@se.fit.edu
Michael Howard, mikehow@microsoft.com

Building More Secure Software with Improved Development Processes

Michael Howard
Microsoft

Few software developers follow security best practices to produce more secure code. Worse, they think of security after the fact. But it's a mistake to separate security considerations from the general software development process. Most development processes use a spiral or waterfall approach in which discrete phases focus on requirements, design, implementation, verification, and release, but the first three phases can change throughout the process, potentially affecting a product's security.

In this installment, I draw on experiences gained as a member of Microsoft's central security team to outline some basic best practices for the software development process. These practices benefitted Microsoft products released since the inception of its Trustworthy Computing initiative in 2002. The following points are a subset of the Security Development Lifecycle process implemented at Microsoft. I also recommend *Processes to Produce Secure Software*.[1]

Security-focused development process goals

This is the easy part—the goal of a process to help build more secure software is to produce more secure software! Actually, there's a little more to it than that. The goal should be to reduce the chance that the designers and developers will inject security vulnerabilities into the design and code in the first place. This goal has the positive side effect of producing more secure software.

You can take several key actions to integrate security into your software development process.

Create a central security team

This group's role is to be an internal security "consulting" organization for the rest of your development team. The team defines process requirements and best practices, defines and build tools, helps perform code and design reviews, does threat analysis, and provides education for the software development staff.

Get executive buy-in

Senior executives must buy into the need to improve software security. The real battle developers face is that few financial data exist on the return on investment of a better development process that leads to more secure software. In other words, we don't know the cost benefit of writing more secure code. Most IT people know the cost of security in terms of lost uptime and service-level agreements, but not necessarily the actual monetary cost. You might want to sell security to management as a competitive advantage, a cost saving for customers, or in terms of the cost to fix post-shipment bugs versus catching them during development.

Raise awareness through education

If you assume that new employees have no notion of what it takes to build more secure software, you'd likely be correct. The most effective security-enhancing change you can make is to provide up-to-date and ongoing education for your software development teams. People won't design, develop, test, and document secure systems until they know the issues.

A major factor in the security unawareness problem is the lack of "security as threats" education available to students today. Most school security classes teach "security as crypto," which doesn't create more secure software. Understanding how a firewall works doesn't help identify and remove integer arithmetic vulnerabilities from code or, better yet, reduce the chance that security defects are added to the code in the first place. Industry has a large part to play in fixing the widespread lack of security expertise.

Education shouldn't spotlight engineers' incompetence—instead, it should raise their awareness and show them what happens when

code contains security vulnerabilities. If there's one thing I've learned in the past few years, it's that developers just need good guidance. Tell them what they should do, provide numerous examples of correct and incorrect design and coding behavior, and they will design and build more secure software.

You can't build more secure software unless you understand the threats to your system.

Even competent developers make mistakes, and some of those will be security related. So, be prescriptive when demonstrating the secure way to perform a development task. For example, don't just show code that includes a buffer overrun; state which library functions to call to help mitigate the overrun, and teach developers to never trust the length of the incoming buffer. Don't tell them that embedding a secret key in code is bad; instead, show them which functions or designs to use.

Ongoing security education is important because the security landscape is constantly evolving. If history is any indicator, next year we'll see new vulnerability classes, which means your engineers must be aware of new threats.

One last thing: remind your staff that security is everyone's problem; it isn't applicable only to security features. Adversaries can attack any code, no matter how small or seemingly insignificant. Take a look at most security vulnerabilities in products from any vendor—very few are found in the products' security features.

Understand your adversary

Understanding threats to your product, once it's deployed in the real world, is paramount when building secure software. You can't build more secure software unless you understand the threats to your system and how attackers will attempt to compromise the software. At Microsoft, we use threat modeling to understand these issues. Section 4.1 of the Common Criteria version 2.1 (http://csrc.nist.gov/cc/CC-v2.1.html) defines threats as, "the potential for abuse of protected assets."

The goal of threat modeling is to understand assets and the threats to which they are exposed, and to create appropriate countermeasures or mitigations to reduce risk to an acceptable level. Several good resources on threat modeling are available.[2–4] McGraw also proposes alternative techniques to determine the means by which an adversary could compromise a system through the use of "abuse cases."[4]

Results from threat modeling should feed directly into product design: no design is complete until it accommodates the potential threats against it. Revisit this stage regularly through the development life cycle—remember, attacks only get better!

Use secure design practices

Designers and architects should adhere to good security design principles, such as least privilege, reduced attack profile, simplicity, fail-closed defaults, appropriate protection of key material, and so on. Several books and articles offer many good design concepts toward this goal.[3, 5–7]

Build more secure code

Software construction errors can lead to implementation flaws, some percentage of which will become security vulnerabilities. Thus, a major goal is to reduce the chance that developers introduce security vulnerabilities. To this end, you should employ secure coding best practices: many good references can help.[3,5,8] You must also have security code reviews, and review all the code—not just new code. I outline a simple process to advance security code reviews in a recent article.[9]

Security tools are a powerful adjunct to code review, but don't use tools to replace competent engineers and good discipline. The advantage of tools is that they can help scale the code-review task; reviewing thousands of files can be slow and tedious, and some tools can find low-hanging fruit. However, most tools don't find deep and complex bugs; you need human review for that.

Tools aren't limited to just helping the code-review process. Some, such as Security Innovation's Holodeck (www.sisecurity.com) and fuzz testing (www.cs.wisc.edu/~bart/fuzz/fuzz.html.), can help uncover security flaws by perturbing the environment in which an application runs or by creating purposefully malformed packets.

Tools are an adjunct to the development process, not a replacement for lacking skills. When defining your security-process budget, focus on your team members first, then buy appropriate tools.

Use external reviewers

Having skilled security practitioners outside your group review your design and code is often worthwhile. When you choose a security consulting company, determine what its strengths are: Some specialize in network penetration testing, some in C and C++, others in Java or .NET, while others have mainly Web expertise. Choose wisely.

At Microsoft, we perform two types of reviews. Some occur at specific milestones during product development, for example, prior to design completion or the beta version; others happen during the final security review I describe later in this article. Large projects, such as Windows, employ both. You must

balance the amount of code and design data available and the timeline at which the review is to take place. Waiting too long might make it hard to change certain product aspects; too early could mean that not enough material will be ready for analysis.

Have security-focused events

Holding security-focused events can help find elusive security vulnerabilities. Events begin with mandatory refresher training for all product group engineers and proceed with the following tasks through the event's duration:

- Developers look for security vulnerabilities in shipping and sample (software development kit) code.
- Testers perform fuzz, abuse, and least-privilege testing.
- Program managers, designers, and architects re-review threat models and design documents to ensure that they reflect current threats.
- Documentation staff reviews online and printed documentation for deployment and usage best-practice violations.

Note that these intense security events alone don't make code more secure. They are merely checkpoints to look for security vulnerabilities in case some might have been missed. Also, perform these security-focused events when you've met your security objectives, not when a date on a schedule arrives.

Perform a final security review

The last stage before a product ships is to verify that it's ready from a security perspective. The central security team I described previously should perform the review, which might include penetration work on network-facing and security-critical components, code review, incoming and postponed bug analysis, threat model review, and fuzz testing. The review team should also verify that the development team adhered to best practices.

Establish a response process

Even if developers could eliminate every software security vulnerability before a product shipped, over time, attackers might discover weaknesses, and software that once was thought to be secure won't be. A good example is integer overflow. Five years ago, few people knew the security ramifications of such a defect, yet today these attacks are common. Thus, developers must prepare a response process for inevitable problems.

One response process component involves evaluating vulnerability reports and releasing security advisories and updates. The other is conducting root-cause analysis on each reported vulnerability and taking appropriate action, which could range from issuing an update in response to an isolated error to updating code-scanning tools to initiating major subsystem code reviews.

The objective of the response phase is to learn from errors and use the vulnerability reports to detect and eliminate other problems before adversaries discover them. The response process also helps the product and security teams adapt development processes to prevent similar errors from appearing in the future. For more information on the Microsoft Security Response Center, see http://channel9.msdn.com/ShowPost.aspx?PostID=19449.

There is no silver bullet for creating secure code, but we can raise the bar by making some improvements to the development life cycle. I urge all software developers and software development companies to evaluate their processes today to see how they can improve them to help deliver more secure software to their customers. ▢

Acknowledgments

This article is derived in part from the Security Development Lifecycle presently in deployment at Microsoft. SDL's initial development was a joint effort by the author and Steve Lipner of Microsoft.

References

1. *Processes to Produce Secure Software*, S.T. Redwine and N. Davis, eds., National Cyber Security Summit, 2004; www.webappsecure.com/bbs/upload/1/20040423005438/secure_software_process.pdf.
2. F. Swiderski and W. Snyder, *Threat Modeling*, Microsoft Press, 2003.
3. M. Howard and D. LeBlanc, *Writing Secure Code*, Microsoft Press, 2002.
4. G. McGraw, "Software Security," *IEEE Security & Privacy*, vol. 2. no. 2, 2004, pp. 80–83.
5. J. Saltzer and M. Schroeder, "The Protection of Information in Computer Systems," 1975; http://web.mit.edu/Saltzer/www/publications/protection.
6. J. Saltzer and M. Schroeder, "Saltzer's and Schroeder's Design Principles," 2000; http://nob.cs.ucdavis.edu/classes/ecs153-2000-04/design.html.
7. J. Viega and G. McGraw, *Building Secure Software*, Addison-Wesley, 2001.
8. J. Viega and M. Messier, *Secure Programming Cookbook for C and C++*, O'Reilly, 2003.
9. M. Howard, "Expert Tips for Finding Security Defects in Your Code," *MSDN Magazine*, Nov. 2003; http://msdn.microsoft.com/msdnmag/issues/03/11/SecurityCodeReview/default.aspx.

***Michael Howard** is a senior security program manager in the Security Engineering group at Microsoft. He is coauthor of* Writing Secure Code *(Microsoft Press, 2002) and "Processes to Produce Secure Software," issued by the Software Process Subgroup of the Task Force on Security across the Software Development Lifecycle. He focuses on security design and development and testing best practice and security process improvement. Contact him at mikehow@microsoft.com.*