Editors: Richard Ford, rford@se.fit.edu
Michael Howard, mikehow@microsoft.com

Becoming a Security Expert

Michael Howard
Microsoft

Whenever I present papers at software development conferences, I hear the same question every time: "How do you try to learn security?" I could be flippant and say, "Do or do not, there is no try," but there are some approaches that software engineers—whether in design, development, or testing—can take to learn more about security.

Instead of being flip, I follow up with two questions: "What area of the development process are you most interested in?" and "Why do you want to learn security?" The most common reply to the first question is development, and the majority answer to the second is, "Because I know nothing about the subject." The second answer isn't a surprise to me, but it's disturbing, so I'll rant for a moment, even at the risk of insulting your intelligence.

The world is amazingly connected today—just about everything depends on some form of connectivity, which brings with it not only productivity enhancement but also potential danger because cyberattackers can exploit it. This is why the answer to my second question is so alarming. At Microsoft, we hire thousands of engineers each year—some straight from school, others from academia, government, and industry—and the percentage of people who understand how to build secure systems is miserably slim. I congratulate anyone who wants to learn more about security because chances are they're trying to fill a critical void in their skill set and security is a critical skill that's in short supply. Who knows, adding security to their resumés might make them more attractive to potential employers!

General learnings

Anyone involved in the software industry should learn a few facts and skills that relate to software security. Let's take a look at each in more detail.

No single "magic tool" will make you secure

This is the classic "There is no silver bullet" mantra, restated, and that's exactly why *IEEE Security & Privacy* has a somewhat tongue-in-cheek Silver Bullet column. Allow me to be a little more forceful: there's no one single solution that will fix your security woes. It takes a more holistic approach that includes education, secure design, updated development toolsets, good testing techniques, and security responses. The Microsoft Security Development Lifecycle is one example of an end-to-end set of process improvements that foster more secure software (www.microsoft.com/MSPress/books/8753.asp).

Stay ahead of attackers

The security environment changes constantly. This is one of the reasons we mandate ongoing security education at Microsoft; what you learned last year is probably only a subset of what you know this year. Attackers come up with new attacks and defenders come up with new defenses. Sometimes, it's the other way around as attackers attempt to circumvent unexpected defenses. Here's the net message: if you create software, it's likely attackers will prod and probe your product, and they'll do it with the latest attack techniques, not just attacks from yesteryear. Therefore, you should understand the threat landscape. I like to tell people to do a couple of things. First, make a point of reading good books on software security. Next, subscribe to a security mailing list or newsgroup. One of the most popular is bugtraq (www.securityfocus.com). From experience, there's a lot of noise on bugtraq, so you'll have to do some filtering to find the gems that crop up every so often.

It's asymmetric!

A few years ago, I coined a phrase that appeared in *Writing Secure Code, 2nd Edition* (Microsoft Press, 2002): "the attacker's advantage and the defender's dilemma." Without going into detail, here are the four principles behind the phrase:

- Defenders must defend all points; attackers can choose the weakest one.
- Defenders can defend only against known attacks; attackers can probe for unknown vulnerabilities.

- Defenders must be constantly vigilant; attackers can strike at will.
- Defenders must play by the rules; attackers can play dirty.

James Whittaker, a security architect at Microsoft, offers a fifth:

- Attackers can remain anonymous. Attackers can take your code offline and spend as much time as they want looking for vulnerabilities.

The moral of this lesson is that no matter how hard you try, attackers will, in the long run, always have the upper hand. It's a simple fact of life, not an excuse. Security is humans pitted against humans. Therefore, it's critical that you improve your code's security posture at the design, implementation, and testing levels.

Critical design

For some reason, the software industry fixates on coding bugs. Plenty of tools exist that will help you find coding bugs, but few, if any, focus on design vulnerabilities. Yet, many security issues are due to an insecure design, or more problematic, a design might have been fine five years ago, but is insecure today.

Secure design principles

You can read about secure design principals in several security texts, but there's nothing quite like applying the principals to a product you have built or will build. The classic secure design text is Jerome Saltzer and Michael Schroeder's *The Protection of Information in Computer Systems* (1975; http://web.mit.edu/Saltzer/www/publications/protection). It's been around for more than a quarter century, yet its spirit is still true today. At the end of one of my secure design classes at Microsoft, I ask students to think of the products they help create—perhaps Microsoft Word's spell checker, a networked service in Windows, or an Internet-facing game—and then ask them to improve their product's features by applying all of Saltzer and Schroeder's principles. The main reason for doing this is to bring the principles into focus. In my experience, software engineers and designers tend to learn by doing rather than just reading. With that said, read up in the core secure design principles and then, even if it's only as a mental exercise, apply them to your product. Next, consider how you would prioritize the changes into the product that these principles demand.

Understand the risk you face

If your code is open to the Internet, it's open to attack. Therefore, it's imperative that you understand what parts of the application attackers might attempt to compromise. This is the reason Microsoft requires product groups to perform threat analyses of their applications—it lets the central security group and the development teams determine whether they have appropriate mitigations and defenses in place to protect both the applications and customers in the event of attack. Microsoft's Adam Shostack recently posted numerous articles about the threat modeling process and how Microsoft is improving it (http://blogs.msdn.com/sdl).

Understand what's exposed

At Microsoft, we call the idea of what's exposed *attack surface analysis*,[1] or more simply, how much code is open to untrusted users. Its goal is to reduce the severity of potential vulnerabilities. Internet Information Services (IIS) 6, a set of services for Web servers, for example, has a great security track record. Since its release in 2003, Microsoft has issued only two security bulletins for it.[2,3] However, both of these security bugs are common to IIS 5.1 and IIS 5.0, yet they're lower in severity in IIS 6 because the code isn't installed by default. Another example is the security vulnerability in the Windows Local Security Authority Subsystem Service (LSASS) process that led to the Sasser worm. The coding vulnerability is present in Windows Server 2003, yet Windows Server 2003 computers weren't affected by Sasser because the networking end point is only accessible to local administrators. In Windows 2000, however, the end point is available to remote and anonymous users (attackers). Attack surface analysis is an acknowledgment that you can never have 100 percent secure code—it might be secure today, but that could change tomorrow.

Critical developer skills

I want to keep this developer section short so as not to muddy the waters with too much information. In my opinion, the single most important skill developers can ever understand is the notion that data is bad—so bad in fact, that it can lead to bad security vulnerabilities, such as buffer overruns, SQL injection, and cross-site scripting, to name a few.[4] If your application consumes untrusted data, such as remote and anonymous input, then that data should be treated as toxic waste until it's analyzed and validated by well-written code in the application. When performing a code review,[5] it's important to follow this tainted data until it's validated. In short, never trust data.

Critical tester skills

Of all the possible security-related testing techniques, nothing comes close to *fuzz testing*[6,7] for finding reliability and security bugs.[8] Fuzz testing is the simple act of building malformed data and throwing it at a parser or network parser with the sole intention of making applications crash. Remember, not all

crashes are the same; a crash might very well be just a crash. But some crashes are special because, with a little more work, they could lead to code execution. The moral of this story, and a critical skill that everyone in the development team should understand, is that crashes shouldn't be written off as mere crashes. Rather, they should be investigated to make sure there's no chance of code execution. But err on the side of assuming a crash could lead to code execution.

A small number of skills exist that anyone in the software development business can learn to improve software security. Whether you're a developer, architect, or tester, it's important that you understand the nature of the constantly evolving security landscape and build defenses into applications at the design phase, never trust input, and then verify that the input handling is robust in the face of intentionally malformed data. Knowing these skills and applying them will lead to more secure software. And that's good for everyone. □

Acknowledgments

Thanks to Steve Lipner, James Whittaker, and Adam Shostack for providing feedback, corrections, and additional material for this article.

References

1. M. Howard, "Mitigate Security Risks by Minimizing the Code You Expose to Untrusted Users," *MSDN Magazine: The Microsoft Journal for Developers*, Nov. 2004; http://msdn.microsoft.com/msdnmag/issues/04/11/AttackSurface/default.aspx.
2. Microsoft Security Bulletin MS06-034, "Vulnerability in Microsoft Internet Information Services Using Active Server Pages Could Allow Remote Code Execution," 11 July 2006; www.microsoft.com/technet/security/Bulletin/MS06-034.mspx.
3. Microsoft Security Bulletin MS04-030, "Vulnerability in WebDAV XML Message Handler Could Lead to a Denial of Service," 12 Oct. 2004; www.microsoft.com/technet/security/Bulletin/MS04-030.mspx.
4. M. Howard, "A Security Lesson that Transcends Programming Language and Operating System Religion," 22 June 2007; http://blogs.msdn.com/sdl/archive/2007/06/22/a-security-lesson-that-transcends-programming-language-and-operating-system-religion.aspx.
5. M. Howard, "A Process for Performing Security Code Reviews," *IEEE Security & Privacy*, vol. 4, no. 4, 2006, pp. 74–79.
6. S. Lambert, "Fuzz Testing at Microsoft and the Triage Process," 20 Sept. 2007; http://blogs.msdn.com/sdl/archive/2007/09/20/fuzz-testing-at-microsoft-and-the-triage-process.aspx.
7. J. Whittaker, "Testing in the SDL," 24 May 2007; http://blogs.msdn.com/sdl/archive/2007/05/24/testing-in-the-sdl.aspx.
8. J. Whittaker, "Reliability vs. Security," 7 Dec. 2007; http://blogs.msdn.com/sdl/archive/2007/12/07/reliability-vs-security.aspx.

Michael Howard *is a principal security program manager in the Security Engineering group at Microsoft. His research interests include secure design, development and testing policies, and best practice. Howard is the coauthor of* Writing Secure Code for Windows Vista, The Security Development Lifecycle, Writing Secure Code *(Microsoft Press), and* 19 Deadly Sins of Software Security *(McGraw Hill). Contact him at mikehow@microsoft.com.*

Interested in writing for this department? Please contact editors Richard Ford (rford@se.fit.edu) and Michael Howard (mikehow@microsoft.com).