REASONABLE SOFTWARE SECURITY ENGINEERING

A PERSPECTIVE FROM ADAM SHOSTACK

Adam Shostack *is a consultant, entrepreneur, technologist, author and game designer. He's a member of the BlackHat Review Board and helped found the Common Vulnerabilities and Exposures (CVE). He's currently helping a variety of organizations improve their security, and also advises start-ups as a Mach37™ Star Mentor. While at Microsoft®, he drove the Autorun fix into Windows® Update, was the lead designer of the Security Development Lifecycle (SDL) Threat Modeling Tool v3 and created the* Elevation of Privilege *game. Adam is the author of* Threat Modeling: Designing for Security[1] *and coauthor of* The New School of Information Security.[2]

Many businesses today make promises like "we take your security seriously," or "we are secure by design." That's great, if your efforts are centered in engineering, rather than marketing or legal. In this Perspective article, we'll talk about the growing need for security engineering, including what, why, where, how and when.

WHAT DOES "SOFTWARE SECURITY ENGINEERING" MEAN?

As you build software, you make tradeoffs between features and properties such as functionality, cost, time to market, reliability, usability and more. Making those tradeoffs to build things is a fundamental goal of engineering work. One property that had been left out—yet is included more often with each passing day—is security.

Let's talk about why that was, and why it's changing. Before I do, let me talk about security as a property versus security as a feature. The distinction is a little subtle, but it's important. Let's take a login box as an example. At the top of web email providers, there's a link that says "sign in." That sign-in is a feature. It allows Google® or Tencent® to give you, and only you, access to your email. We can also talk about the security properties of that login box. It might resist brute-force attacks, where attackers try to login with one password after another. It might have few vulnerabilities. A lack of vulnerabilities is a security property, one shared by many features. If you're going to be secure by design, you need to think about these properties early, so you can design and build your systems in a secure way.

For a long time, software buyers (primarily governments) asked for explanations of software security properties, in programs like the "Common Criteria." Remarkably, writing about the software after it was created didn't lead to secure software. One watershed event in engineering for software security properties was the publication of the "Trustworthy Computing" memo at Microsoft. In that all-hands memo, Bill Gates declared that security mattered to Microsoft, and followed the memo with massive investment over more than a decade. (When I worked for Microsoft, the Trustworthy Computing security team had several hundred staff, and major product teams like Windows® and Office had their own security teams, too.) That effort demonstrated that investment in security engineering pays off, and more importantly, it does so in a way that executives can understand: a big, successful software company keeps spending money on it.

1 *Threat Modeling: Designing for Security,* John Wiley & Sons, Inc., USA, February 2014, *https://www.wiley.com/en-us/Threat+Modeling%3A+Designing+for+Security-p-9781118809990*

2 *The New School of Information Security,* 1st Edition, Pearson Education, USA, April 2008, *http://www.informit.com/store/new-school-of-information-security-9780321814906*

WHY YOU NEED SECURITY ENGINEERING

If your business is a technology business, then you know software is essential to your products or services. (I use the term "product" to mean "the stuff you sell"—so even if it's a service, I'm going to call it a product, and ask your forgiveness.)

Now if your business doesn't think it's a technology business, well, I'm not going to tell you you're wrong. I'm going to let web browser creator and venture capitalist Marc Andreesen do that. He famously said, "software is eating the world." What he meant is that it's hard to compete when you have humans doing work that computers can do. The computers are faster and cheaper. Even if applications are not directly sold to customers, there is still potential for fraud and abuse if those applications are not architected for robustness and engineered with reliability in mind.

So, even if you don't see your business as a software business, you now have software at the heart of what you're doing, and that's where your security efforts need to be. Security is an important property (whether the business is technology based or not), one that must be considered in all new technology deployments: systems, applications, mobile apps and so forth. Impending laws like the EU General Data Protection Regulation (GDPR)—which applies to any organization that offers goods or services in the EU—will require security engineering.[3]

WHERE YOU NEED SECURITY ENGINEERING

There are three places within your organization where you can try to meet your software security needs, and two of them are wrong. I'll start with those, and then talk about choices you can make.

The first wrong place is to put marketing in charge. Don't get me wrong, I love marketing. Would you believe, I think marketing is tremendously important? Really, you want to get the word out about what you're doing, how your product helps people solve the problems they face. And marketing can tell the world that your product is secure. In fact, one manufacturer of home networking products promoted the security of its routers on the company's website, which included materials headlined "EASY TO SECURE" and "ADVANCED NETWORK SECURITY." This anecdote stems from a January 5, 2017 press release in which the FTC announced that it was suing, in part because "the company failed to take steps to address well-known and easily preventable security flaws."

The second wrong group to have in charge of your security is legal. Now, lawyers are absolutely essential to help you understand how the FTC's enforcement power relies on doctrines of unfairness and deception, and how that enforcement power differs from regulatory power. (I might be wrong about that—you know what to do. And no! Don't ask Facebook. Ask your lawyer.) I believe that lawyers are best suited to give you legal advice. Key words: legal advice. That is, they advise you about the law. Sometimes, that's simple: if you kick puppies, you will, deservedly, go to jail for animal cruelty. Other times, it's a lot more nuanced. Should you put the words "Advanced Network Security" on your product when it doesn't have advanced network security features? Probably not. But what should you do? There are lots of standards, but they're imprecise.

The answer is that you need to engineer for security. Marketing can tell you if the features are competitive and meaningful to customers. Legal can tell you that your new feature might violate the GDPR. But your approach to security needs to be grounded in engineering working collaboratively with business teams and even customers.

3 For more on GDPR, see ISACA's resources and tools at *https://www.isaca.org/info/gdpr/index.html*.

HOW TO DO SECURITY ENGINEERING

Security engineering is a big, complex topic, and there's a lot of advice out there on how to do it. A recent speech by Suzanne Schwartz of the FDA laid out an approach that's worth thinking about (even if the Food and Drug Administration does not oversee your business). Schwartz said that your security should be comprehensive, structured and systematic.[4] That's a powerful framing of the goals. I'm going to stay at the high level for a moment, and talk about each of them, then get specific about how to execute.

Strategy: Comprehensive, Structured and Systematic

You can view these three goals as tests: Is my program comprehensive? Is it structured? Is it systematic?

So, what is a comprehensive program? It's one that covers everything you do. You cannot cover only products where there's a "clear and present danger," the "high risk" products, or "ones that have hired a security person." You probably can, and should, devote more resources to the first two sets. Maybe the third hired a security person because someone "had a bad feeling about this." You cannot say, "Bob, it's ok, we understand that your team likes to 'move fast and break things,' so sure, just skip fixing your security bugs." (Well, you can. Maybe you should have that lawyer advise you, first.)

Turning to structure (defined by the Oxford English Dictionary [OED][5] as "the arrangement of and relations between the parts or elements of something complex"), do you have defined steps with inputs and outputs, responsibilities and escalation paths? Do you record those decisions and review them from time to time? Now, a structured program doesn't need to apply the same detailed actions to each thing it looks at. For example, you can create a risk management process in which steps for managing people's highly sensitive data are more rigorous than your process for managing their cat videos. If one step is "use static code analysis" then you could say, "for high risk projects, all severity 1 and 2 issues must be fixed; for low risk projects, severity 2 issues must be triaged." Whatever decisions you make, the process itself still needs structure, and that structure can't depend on which side of the bed you woke up on. It needs to integrate with the work you do to build products. (This has long been a crusade of mine. Reports get less attention from engineers than bugs or tickets. When I was responsible for the Microsoft Threat Modeling tool, it had one-click bug filing. Our team's first open source release was a plugin layer so that it could file bugs in a wide variety of other systems.)

Structured and systematic are closely related, and thus it's interesting to see both in Schwartz's recommendation. Systematic means (again per the OED), "done or acting according to a fixed plan or system; methodical: a systematic search of the whole city."

So, where structure is about integration into development, being systematic means that security is part of the whole product cycle, from concept through delivery to end of life. It's a part of every agile sprint. That said, the security work you do at each stage is different. As you conceive of the product, you threat model to think about what can go wrong. As you build it, you build features to defend against the problems you envision. You use secure coding techniques—including good language selection, secure coding and static analysis—to ensure that you don't accidentally add problems. You might use the Open Web Application Security Project (OWASP) top ten, dynamic analysis software testing (DAST) or fuzzing in testing. As you operate, you watch for problems, and you give people who find security problems an easy way to report them to you, confident that they won't be sued. Maybe you even reward them with a "bug bounty." You also keep an eye on your platform and stack to catch vulnerabilities in the open source or commercial libraries and platforms you use. And when it's all done, you light a match and burn the data you no longer need.

All this work—structured, systematic and comprehensive—needs to be grounded in engineering. It can be driven by someone in engineering or in security, if "security" is organizationally elsewhere. If the security person or lead doesn't have close ties into product engineering, then the role could be understood as raising security concerns, and not worrying about shipping, etc. If they are inside engineering, then they may get pressure to sign off on issues that should be escalated.

4 Miliard, Mike; "FDA exec to medical device manufacturers: 'Bake security into the design'," *Healthcare IT News*, 13 September 2017, *http://www.healthcareitnews.com/news/fda-exec-medical-device-manufacturers-bake-security-design*

5 *Oxford English Dictionary*, "structure," Oxford University Press, United Kingdom

Secure Development Lifecycle

The work to secure product engineering is usually packaged as a secure development lifecycle (SDL or SDLC). There are waterfall and agile variations, and each organization customizes what each step entails for itself.

To get started, you need managerial and technical proof points. To date, we've been talking at a management level. To convince software engineers to do security work, you need to start with the understanding that they're already too busy. Their default argument will be "What do we stop doing?" or "How much longer should we take?"

The answer is that lots of agile teams spend time up front in ways that let them move faster. They engineer build pipelines rather than copying VMs to production, so that they can always, perfectly, spin up a new one. They write test code to help them find problems. Each of these involves work that lets you move faster, and similarly, energy invested in software security will reduce future unplanned work: It will reduce last minute pain when penetration testing discovers security flaws; it will reduce the pain of an outsider finding a bug and generating an emergency response, full of unplanned work.

The technical proof you'll need is that the work finds important bugs. There have been two good starting points for that, fuzzing and threat modeling. Fuzzing means sending random input to a program, to see what breaks. It works remarkably well on programs written in C and C-like languages. If you're using one, setting up a fuzzer can be low effort and high reward. But with more modern languages, fuzzing doesn't work as well. Threat modeling will likely work, and find good bugs.

Threat modeling means a set of structured techniques to address four key questions about a project:

(1) What are we working on?
(2) What can go wrong?
(3) What are we going to do about it?
(4) Did we do a good job?

The easy way to get started with threat modeling is to get a copy of *Elevation of Privilege*. It's a game I created while at Microsoft to teach people how to threat model.[6]

To me, threat modeling is at the core of security engineering. It enables you to know if you're being comprehensive and systematic. If you don't have agreement on what you're working on, then perhaps there's a dev team adding some blockchain to your product. How would you know if you haven't drawn a picture? If you don't know what can go wrong, how can you claim to be systematic about addressing it? (The tie to structure is not as fundamental, but good threat modeling relates to choices about what defensive features you're going to build or deploy in a structured way.)

TRAINING	REQUIREMENTS	DESIGN	IMPLEMENTATION	VERIFICATION	RELEASE	RESPONSE
Core security training	Establish security requirements	Establish design requirements	Use approved tools	Dynamic analysis	Incident response plan	Execute incident response plan
	Create quality gates / bug bars	Analyze attack surface	Deprecate unsafe functions	Fuzz testing	Final security review	
	Security & privacy risk assessment	Threat modeling	Static analysis	Attack surface review	Release archive	

FIGURE 1: Secure software development process model at Microsoft

Source: *Microsoft Security Development Lifecycle (SDL) – version 5.2*, "Introduction," Microsoft Corporation, USA, 2012, *https://msdn.microsoft.com/en-us/library/windows/desktop/cc307406.aspx*. © 2012 Microsoft Corporation. All rights reserved. Licensed under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported, *https://creativecommons.org/licenses/by-nc-sa/3.0/*

6 See details at *https://www.microsoft.com/en-us/download/details.aspx?id=20303* or *https://www.threatmodelingbook.com/resources*.

WHEN TO DO SECURITY ENGINEERING

When to get started? There's no time like the present. If you're just starting to engage in software security engineering, focus on a single product, and, if you can, a product that's still in its early days. That way, you can learn what works for your engineering culture and then roll it out to additional products.

CONCLUSION

The world is changing rapidly. The way you secure your products needs to change as well; it must be executed as security engineering: a comprehensive, systematic and structured approach that meets the new and evolving needs of the business.

We have learned how to do software security engineering. That's not the same as saying it's easy to do, but we know how to threat model, how to scale threat modeling and how to use it to drive that comprehensive, systematic and structured approach.

Also contributing to this article: Larry Marks, Tara Singh, Michael Krausz, Meenu Gupta, Krishna Seeburn and David Vohradsky