Software Security Paper

CSC 424 – Software Engineering

James Setliff

April 18, 2021

Secure software is a complex topic involving a multi-disciplined, multi-faceted approach to the design, maintenance, and deployment of software that avoids common mistakes and takes a forward-thinking approach to preventing new assault vectors. There is no single answer to the question of what software security is, as thousands of vulnerabilities already exist between software, hardware, infrastructure, and personnel, but as developers, we must be mindful and vigilant when planning, designing, and implementing our programs to prevent abuse wherever possible. The responsibility does not lay solely with the programmer though, as all members of the team involved with the software development life cycle can and should contribute to making the most resilient product possible.

As we move into the future, the growth of connected devices and always-online services has grown exponentially. This leaves many of us open to malicious attacks via avenues that were previously impossible. Every time we make the choice to purchase a new fancy internet connected device, we are imposing risk on ourselves, and placing trust in the manufacturer to have taken the right steps to protect the consumer from being compromised. It is no exaggeration to say that your "smart tea kettle" or "connected coffee machine" can give hackers direct access to your password protected home network, as demonstrated by two BlackBerry engineers. "In other words, if a hacker can take control of your caffeine dispenser, they can gain access to everything else it connects to." [1] [2] With more and more devices becoming "smart" and "connected," how do software shops ensure they are not introducing these hazards into their own project?

To start we must understand the mistakes that developers make repeatedly. The University of Colorado lecturers Stevenson and Alharbi give us a list of nineteen "deadly sins" and some thoughts on how to mitigate these attacks. From simple buffer overruns, string formatting problems, integer overflows, SQL injection, command injection, cross-site scripting (XSS), failure to protect network traffic, and bad error handling to more advanced "magic URLs and hidden forms, misuse of SSL and TLS certificates, network name resolution, race conditions, improper file access, induced race conditions, use of non-cryptographically strong random number libraries, and unauthenticated key exchanges, there are plenty of demonstrable, repeated failings from developers everywhere. Many of these issues arise from the same basic inconsistencies, developers are often undereducated about the languages, libraries, and packages that they are using, and many do not learn the lesson until it is too late. Most importantly, these issues are not restricted to inexperienced developers. In 2011, Oracle's own MySQL.com site was hacked via an SQL injection attack. [3] Many of these attack vectors are well-understood and easy to prevent, however the pace of new development is often prioritized over combing through old code to hunt for vulnerabilities. Day after day and month after month, we see examples of large data stores being compromised and user data being sold on the open market to the highest bidder. Modern corporate structure simply does not value prevent these attacks over designing and implementing new features that will attract customers.

As stated previously, secure software is not a single step in the development cycle, but something that must be kept in mind throughout the development life cycle. In a paper from Open Web Application Security Project, the following principles are identified as key to security of software. "These principles play a vital role in application security and should be followed across all stages of the software development lifecycle. These principles also have room in other

security areas, including network and host security, and are generally derived from a broader field of engineering." For starters, always try to identify and secure the weakest link in your production chain. If, for instance, you have a strong authentication system it is unlikely an attacker will attempt to compromise the system in that way. Defenses should always be applied in a layered manner where possible. An example in this case is not to pass unencrypted data between internal servers simply because they are behind a firewall. In the event the firewall is bypassed, you have no secondary layer of protection. Developers should additionally always seek to handle errors gracefully, to prevent system failure due to an attack of this nature. Another useful approach to keep in mind is to always assign the lowest level of privilege necessary for accounts to perform their function. Improperly configured accounts can lead to security holes in an otherwise shored-up system. Compartmentalization and simplification also help contribute to not just ensuring limited scope of access but assists in identifying where a security breach or flaw has occurred and makes timely correction of that flaw easier. [4] These are just a few of the suggestions put forth in the paper, and with only the briefest of explanation, but it is apparent even from this list that there are many things to mindful of when designing and implementing software systems.

The dream of writing perfect, impenetrable code delivered on perfect, impenetrable networks built upon perfect, impenetrable frameworks, is simply that, a dream. Nobody is perfect, and that is probably doubly true for developers. Assuming that everyone at every stage of the software development life cycle is on-board and interesting in making the most resilient applications possible, there are still undiscovered attack vectors that some 12-year-old Eastern European kid will use to eventually compromise your software for fun… After all, technology is rapidly evolving, and IT Professionals have little time to play around with code and learn the latest and greatest hacking methodologies. This is where a strong software security testing policy can help identify vulnerabilities before they are exploited and add an additional layer of protection if done properly.

Much like software security, security testing is an expansive discipline that covers a variety of topics and approaches. It is also not a guarantee in any way that your software is bulletproof. The process of security testing is simply intended to reveal flaw in your security processes and help ensure your software remains functional and protected during and after an attack. There are seven main types of security testing as follows. Vulnerability scanning is done through automated software that scan your software components for know vulnerability signatures. Security scanning is used to identify network and system weaknesses and can be conducted manually and with automated testing tools. Penetration testing, or pen testing, simulates an attack from an external hacking attempt. Risk assessments help managed the security risks associated with failures throughout different points of an organization's software. Security auditing is often conducted in-house and is used to comb over code for potential vulnerabilities in your own codebase. Ethical hacking is sometimes used by large organizations using bug bounty systems, where hackers with a more rigid moral compass use their skills to identify and report exploits to the organization, often for financial compensation. And finally, Posture assessment is used to combine all the above methodologies to produce a security posture report for an organization. [5] Often, different parts of this security testing discipline are left out of an organizations security plan to keep costs down, and as noted earlier in this paper, often lead to data leaks or otherwise compromised software. In a properly funded and maintained security

plan, these approaches will be applied throughout the software development life cycle, with detailed security test plan reports being generated to show where vulnerabilities are identified and where additional resources can be allocated to prevent further breaches. The question is what the tools are that organizations and developers have at their disposal to help identify and mitigate these risks.

These tools are often broken into a few classes, including but not limited to Static Application Security Testing, Dynamic Application Security Testing, Software Composition Analysis, Database Security Scanning, Test Coverage Analyzers, and Application Security Testing Orchestration tools. A basic argument for the use of these tool is made on Carnegie Mellon University's SEI blog as, "Bugs and weaknesses in software are common: 84 percent of software breaches exploit vulnerabilities at the application layer. The prevalence of software-related problems is a key motivation for using application security testing (AST) tools. AST tools are effective at finding known vulnerabilities, issues, and weaknesses, and they enable users to triage and classify their findings." [6] Static Application Testing is often performed by someone in house using a tool that scans code for vulnerabilities. There are dozens of available programs for this purpose, like ABASH, dotTest, Dlint, etc. with a list provided by the National Institute of Standards and Technology sorted by language. [7] In contrast, Dynamic Application Security Testing is often performed from outside of the system and looks for issues with interfaces, requests and responses, scripting attacks, injections, authentication and more. Intruder is a primary example of a tool of this type. Additional use of tools helps to round out necessary coverages based on the needs of the software, for instance, WireShark, a network analysis tool that captures packets in real time, providing details about protocols, encryption, and information your packets might be exposing, might be used to help identify issues along a network if a network interface is used in production. Other mobile testing tools might be employed for a mobile based application.

In the long run, implementing both the software security methodologies and security testing approaches and tools should save time and effort on rework by identifying and correcting issues earlier in the software development life cycle. This would translate to not only a better product, but potentially lower costs and an untarnished image by avoiding bad publicity due to software compromises. As future developers and members of software development teams, we have a responsibility to our teams, ourselves, and our customers to take a proactive stake in the security of the software we write and maintain. Being mindful or the common mistakes listed at the beginning of this paper is a good first step, however what we really need is an understanding of why those common mistakes are problematic to begin with. As with anything, it is always better to have a deep understanding of the why and not just the surface level understanding of the how. Holding ourselves accountable to these higher standards of development will mean a higher overall quality to the product we provide, and fewer future headaches. Software security is, at its core, a game of cat and mouse. The old saying, "If you build a better mousetrap, the world will beat a path to your door" has special meaning for software development. Some will be interested in your more secure product as a consumer, and others will show up simply to try and break it. In that respect, there is always a bigger fish, but that cannot prevent you from trying. Do not rest on your laurels, be proactive about writing good, clean code, and work with the rest of your team to build a security conscious attitude, and you will be successful.

[1] S. Margaritelli, "The easy way your “smart” coffee machine could get hacked and ruin your life," Quartz, 3 February 2017. [Online]. Available: https://qz.com/901823/the-easy-way-your-smart-coffee-machine-could-get-hacked-and-ruin-your-life/. [Accessed 18 April 2021].

[2] Producer, "BlackBerry Security Summit 2016: How Your Tea Kettle could be a Gateway for Hackers," BlackBerry, 22 July 2016. [Online]. Available: https://www.youtube.com/watch?v=WelqFVH6eZE. [Accessed 18 April 2021].

[3] H. Stevenson and K. Alharbi, "Software Security," Spring 2012. [Online]. Available: https://home.cs.colorado.edu/~kena/classes/5828/s12/presentation-materials/stevensonhunteralharbikhali.pdf. [Accessed 18 April 2021].

[4] E. Lebanidze, "Securing Enterprise Web Applications at Web Applications at Web Applications at Web Applications at the Source: An the Source: An the Source: An the Source: An Application Security Application Security Application Security Application Security Perspec," [Online]. Available: https://owasp.org/www-pdf-archive/Securing_Enterprise_Web_Applications_at_the_Source.pdf. [Accessed 18 April 2021].

[5] Editor, "What is Security Testing? Types with Example," Guru99, [Online]. Available: https://www.guru99.com/what-is-security-testing.html. [Accessed 18 April 2021].

[6] T. Scanlon, "10 Types of Application Security Testings Tools: When and How to Use Them," Carnegie Mellon University, 9 July 2018. [Online]. Available: https://insights.sei.cmu.edu/blog/10-types-of-application-security-testing-tools-when-and-how-to-use-them/. [Accessed 18 April 2021].

[7] Editor, "Source Code Security Analyzers," National Institute of Standards and Technology, [Online]. Available: https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html. [Accessed 18 April 2021].