James Rinehart

CSC 424

Software Security

4/23/2021

In the modern world, we face modern threats. The computer systems we build are no different, and the number of threats they face are staggering. That is why concepts like Software Security shine like a beacon to ship of lost sailors. Software Security is not a tangible thing, it is an idea, more specifically it is a methodology of software design wherein the software is built from the ground up with security in mind at every step of the way. As surprising as it may be, this is a fairly new concept. The first textbooks and classes on the subject did not start appearing until as late as 2001 [1]. This is most likely because the creators of computers and their associated networks never could have imagined just how ubiquitous their creation would be in the future, and such ubiquity drew attention from malicious minds. This resulted in what we see today, with the concept of Application Security dominating our security world. Application Security is the approach to security that involves already written code and software; the security protocols are designed around protecting vulnerable code [1]. With Software Security, most of the threats are eliminated by eliminating the vulnerable code. Through Software Security best practices, programmers are able to increase the security of their code with less effort in the long term.

In the field of security, testing is paramount. If you are able to detect the vulnerabilities in your software, then you are able to more effectively secure that software. The Software Security approach is no different, however, the focus of the testing is slightly different. The tests will need to reveal more nuanced vulnerabilities such as weaknesses in the architecture of the software. For this, tests such as penetration testing with architectural risk analysis can be done to reveal issues such as the rather common buffer overflow [2]. Risk analysis is a process that happens throughout software development; in any risk analysis situation, you will first need to take inventory of what you are trying to protect. Once you have identified your assets, you can then start to design the software. During the design phase, that is when the bulk of risk analysis is done. First, potential risk vectors must be identified, usually the most common are similar from project to project. Then, architectural risk analysis comes into play. Three main things happen during architectural risk analysis: known vulnerability analysis, ambiguity analysis, and underlying platform vulnerabilities [2]. The preconditions for vulnerabilities to be exploited are examined as well as all possible states exploitation may lead to [2]. The reason for focusing on known risks is because similar to any sort of quality assurance procedure, risk analysis can only prove when there is a vulnerability, not the lack of one [2]. When ambiguity exists between the requirements and development of software, vulnerabilities will inevitably surface. Ambiguity analysis attempts to crack down on that by eliminating the potential misunderstandings between the customer's requirements for the software and the developer's implementation of the software's actions [2]. This leads to less vulnerabilities by having the whole team working together to create a clearly defined and organized product that is more efficiently secured. Unfortunately, even if the software itself is immune, the platform upon which the software runs may have its own weaknesses. Vulnerabilities may exist within the software's execution environment which allow malicious actors to bypass much of the software's built-in security. These vulnerabilities may come from places such as: the operating system, the network, application platforms (such as PHP, ASP.net, and Jakarta), and interactions between components

[2]. Risk analysis, with architectural risk analysis, can be an incredible tool for discovering vulnerabilities that might otherwise go unfound until exploitation.

Another invaluable tool in Software Security is the idea of the code review. As simple as it sounds, sending your code off to a peer to review can identify many vulnerabilities that you had originally missed. This reduction in vulnerabilities makes for a far more secure finished product, as well as a more satisfied customer. Additionally, code review can identify more than security vulnerabilities. It can help uncover errors or inefficient coding patterns, leading to an increase in the quality of the software. This would also increase the satisfaction of the customer. Finally, and most importantly, code review allows you to learn much more from your peers by having the input from senior members of the team. Conversely, if you are the senior member, you would be the one teaching your peers. This leads to a more positive and productive work environment by inducing higher quality teamwork.

When it comes to Software Security, there are many methods of implementation. However, some of the best practice techniques concerning Software Security are defensive programming, threat modeling, and understanding your attack surface [3]. Also, Software Security operates most efficiently when paired with additional security methodologies, such as Application Security.

Defensive programming is a form of designing improved software and source code. It does this in three main parts. General quality: improving the quality of the code by removing software bugs and problems. Coding comprehension: increasing the readability of your code by adding comments and generally de-obfuscating the code. Predictability: making the software behave in predicable ways despite unexpected inputs or actions [4].

Threat modeling works to identify and communicate threats and mitigations within the context of protecting something of value [5]. A threat model will usually include a description or design of what it is that is threatened. Then, it will have a list of assumptions that can be checked in the future as the threat landscape changes. Following that list, it will contain a list of potential threats to the system and in a separate list it will have actions to be taken for each threat listed. Finally, a threat model will have a way of validating itself and the threats, with verification of successes of actions taken [5].

Understanding your attack surface is critical to the success of your security procedures. In order to fully understand your attack surface, you must carry out an Attack Surface Analysis. First, the attack surface of an application is: the sum of all paths for data and commands into and out of the application, the code that protects these paths, all valuable data used in the application,

and the code that protects that data [5]. An Attack Surface Analysis is about identifying which parts of a system need to be reviewed and tested for security vulnerabilities [5]. The idea is to understand the areas of risk in software, and to make the developers and security team aware of the regions of the software that are open to attack. An Attack Surface Analysis allows you to identify what functions or parts of the software you need to review, what areas of your code require protection, and when to do reassessments in the future [5].

Each one of these techniques is valuable to you in securing your code through Software Security. However, they lose most of their value their own. In order to build an effective security procedure, you must combine all of these techniques together in addition to other styles of defense like Application Security. In addition to those testing and coding techniques, some habits that a security aware designer should keep in mind are routine patches, thorough education of users, using automation, the enforcing of least privilege, and segmenting your network [6]. Routine patches allow you to decrease the amount of time potential attackers have to work with when attempting to exploit vulnerabilities in your code. Thorough education of users prevents many types of security breaches that are unstoppable to security protocols, such as giving out your password to others. Automation is already used to great effect in attacking your systems using things such as port scanners. Fortunately, there are automated systems for configuring security systems and the like, that equally reduces the effort required to maintain those systems. Enforcing of least privilege is when users are given only the access privileges that the need to complete their jobs/tasks. This lowers the area of your attack surface by eliminating unnecessary access rights which may cause compromises [6]. Finally, if you have access to the network you are operating within, and have administrative control over it, you should segment that network as much as possible. This also goes for other applications such as coding. If valuable information can be kept separate, then do so. This will prevent a single breach from compromising your entire system.

In conclusion, security is one of the most important parts of our lives, yet it is one of the most overlooked. However, with the proper implementation of procedures with methodologies such as Software Security and Application Security, most of your security needs will be met.

Sources:

#1. McGraw, Gary. “What Is Software Security?” Software Integrity Blog, Synopsys, 31 Mar. 2004, www.synopsys.com/blogs/software-security/software-security/.

#2. Peterson, Gunnar, et al. “Architectural Risk Analysis.” *Cybersecurity and Infrastructure Security Agency*, United States Government, 2 July 2013, us-cert.cisa.gov/bsi/articles/best-practices/architectural-risk-analysis/architectural-risk-analysis.

#3. Edmiston, Taylor D. “What Is Software Security?” TheFramework, Medium, 28 Aug. 2019, medium.com/the-framework-by-tangram-flex/what-is-software-security-e03a5ee7a6b5.

#4. Hoffman, Tyler. “Defensive Programming - Friend or Foe?” Interrupt, Memfault, 15 Dec. 2020, interrupt.memfault.com/blog/defensive-and-offensive-programming.

#5. Cheatsheet Series Team. “Attack Surface Analysis Cheat Sheet.” Attack Surface Analysis - OWASP Cheat Sheet Series, OWASP, cheatsheetseries.owasp.org/cheatsheets/Attack_Surface_Analysis_Cheat_Sheet.html.

#6. Synopsis Editorial Team. “Top 10 Software Security Best Practices: Synopsys.” Software Integrity Blog, Synopsys, 6 July 2020, www.synopsys.com/blogs/software-security/top-10-software-security-best-practices/.