Software Security

CSC 424 – Software Engineering II

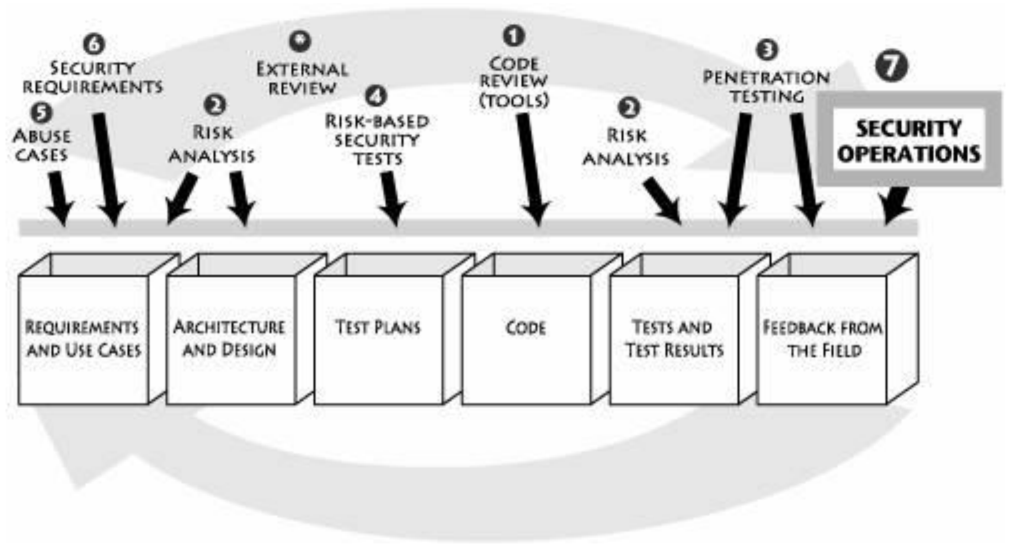
Zachary Light

4/23/2021

In the modern age, software exists in nearly every aspect of life. Almost every person has a smart phone, most people have computers and many people have many more smart gadgets in their home or office that they interact with on a daily basis. Just on a cell phone, there could be a great many different software programs including the operating system (OS) and any additional apps that come standard or are downloaded. With this much exposure to software, not to mention the importance of some things that rely on software, software security is more important than ever before. Software needs to be secure to protect personal information, bank and credit accounts, and the software programs themselves.

So, what is software security? The security of software systems is a vast topic, but in general, security is the freedom from danger and threat [1]. Software security is employed to assess, detect, minimize and protect against threats to the system [1]. Practicing these techniques protects the system from vulnerabilities like hacking [1]. It keeps the system performing properly and often safeguards personal information. Security must be kept in mind at all stages of a software's lifecycle to keep it safe [1]. Software security is not always perfect but its important to make it as secure as possible and to fix any additional problems along the way. The earlier a vulnerability can be found the easier it is to patch [1]. This also keeps the software and users safe from malicious attacks or even harmless bugs. If a problem is found before it can be exploited the issue is easily resolved.

There are many techniques involved in software security. Each technique has its own strengths and weaknesses when it comes to assessing or protecting against vulnerability. The more that are used in a software, the safer it will be. Seven of the top software security techniques include code review using tools to find vulnerabilities, bugs, and weaknesses, architectural risk analysis to identify flaws, penetration testing, risk-based security testing, Abuse cases to examine how a system behaves under an attack, security requirements, and security operations [2].



Code review requires a thorough understanding of the software design, and the language used [3]. A good code review starts with a review of threat models and design specifications before moving onto the source code [3]. Knowing the design and the potential threats can help to find vulnerabilities easier and faster. Code should only be reviewed if it is complete, or at least if that particular section is complete [3]. There is no point in reviewing code if it is incomplete as the developer may be in the process of writing in security, or they may later write code that has vulnerabilities. To make the process easier, begin the code review using tools to catch the simple problems [3]. Upon finding any problems, be sure to fix them all right away before continuing the review [3].

There are a number of tools you can use to help with code review. Some tools apply work with multiple languages. Many of these tools are even free. There's Apache Yetus, which supports eight languages including Java, C++, and Python [4]. Moose supports five languages including Java and C++ [4]. Semgrep works with six different languages including Java and Python [4]. It also has a cool feature called "language-agnostic mode" in which it has experimental support for eleven other languages [4]! There's also SonarQube which supports twenty-six different languages including C++, Java, Python, and Kotlin [4]. There are more free tools that work on multiple languages as well as several paid tools for multiple languages. There are still even greater numbers of both free and paid tools built for one specific language. They all perform slightly differently so the more that are used in code review the more bugs that will be caught.

Through architectural risk assessment, flaws can be identified that would otherwise expose information assets to risk [5]. Strategies to reduce or eliminate those risks are developed and implemented in an order based on how much impact they will have on the business [5]. Fixing flawed architecture often requires large redesigns [5]. This is what separates architectural risk assessment from simple code review.

Penetration testing is a type of ethical hacking where a person or program tests software, hardware, or systems to identify potential vulnerabilities that a malicious hacker could exploit [6]. The ethical, or white hat, hacker gathers intel on the system then attempts to break in through pre-identified possible points of entry [6]. The results whether successful or not are reported back to the developers [6]. This is an invaluable type of testing that requires a very specific set of knowledge and skills. Many people would use these skills maliciously but there are also several security experts that put them to use for good. Sometimes they even do so without the company's approval and let the company know afterward what they have done. This is the equivalent to having a thief break into a bank to test its security.

Risk based security testing is similar to penetration testing. Risk based security testing acts on known risks and attempts to exploit them [2]. Tests are designed specifically around known risks [2]. These types of tests can help eliminate or at least minimize identified risks in the software. They help pinpoint exactly how a risk might hurt the software and can then adjust the code to prevent this type of attack.

"If software is going to be used, its going to be abused" [2]. Always assume software will be abused. If you know it will be abused, its best to know how it will handle the abuse. Abuse cases happen later in the software security assessment [2]. Once the flaws are identified, tests can be run to put intentional abuse on the system. This will give an idea of how the system might react to attacks that either can’t be stopped or ones that the developers are not yet aware of. Abuse cases require you to think like an attacker in order to better assess threats [2].

Security requirements are parallel to design requirements [7]. These must be predetermined before development and ensured throughout development [7]. Security requirements are a list of security measures that must be in effect upon release of the app to the consumer [7]. Often, they are though of from previous cases, common industry requirements, and user cases [7]. Examples include password encryption, and information hiding.

Security operations is a bridging of the gap between software security and information security [2]. It seeks to solve the problem where developers often overlook security issues [2]. Using knowledge and techniques from information security can help make software more secure as well [2].

Some other methods that can fit under the umbrella of these seven ideas include defensive programming, secure coding, threat modeling, understanding the attack surface, sandboxing, code auditing, application security, and defense in depth. Many of these require expertise, or at least a little practice [1]. Apply as many as possible to a software to improve security. Of course, there are limitations. Time is always a factor, as well as budget and human resources. Select the best grouping of security practices that are available to the project and the software should be quite secure. While learning to code, security is often overlooked as languages, logic and technique are prioritized. Practicing simple security methods early on is a great way to increase expertise in software security.

Reference List:

1. T. D. Edmiston. What is Software Security? *Medium*. [Online] Available from: https://medium.com/the-framework-by-tangram-flex/what-is-software-security-e03a5ee7a6b5 [Accessed 23 April 2021].

2. G. McGraw. (2013). *Software Security: Building Security In*. Addison-Wesley

3. M. Chmielewski et. al. Code Reviews: Find and Fix Vulnerabilities Before Your App Ships. *Microsoft*. 2 October 2019. [Online] Available from: https://docs.microsoft.com/en-us/archive/msdn-magazine/2007/november/code-reviews-find-and-fix-vulnerabilities-before-your-app-ships [Accessed 23 April 2021].

4. List of tools for static code analysis. *Wikipedia*. [Online] Available from: https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis#Multi-language [Accessed 23 April 2021]

5. G. Peterson et. al. Architectural Risk Analysis. *Cybersecurity and Infrastructure Security Agency*. 3 October 2005. [Online] Available from: https://us-cert.cisa.gov/bsi/articles/best-practices/architectural-risk-analysis/architectural-risk-analysis#:~:text=Architectural%20risk%20assessment%20is%20a%20risk%20management%20process,business%20information%20assets%20that%20result%20from%20those%20flaws. [Accessed 23 April 2021].

6. L. Rosencrance and P Mehta. pen testing (penetration testing). *TechTarget*. October 2018. [Online] Available from: https://searchsecurity.techtarget.com/definition/penetration-testing#:~:text=Penetration%20testing%2C%20also%20called%20pen%20testing%20or%20ethical,be%20automated%20with%20software%20applications%20or%20performed%20manually. [Accessed 23 April 2021].

7. J. Boote. Are you making software security a requirement? Synopsis. 22 July 2020. [Online] Available from: https://www.synopsys.com/blogs/software-security/software-security-requirements/ [Accessed 23 April 2021].