Software Security

CSC 424 Software Engineering II

Sarah Wathke

Due: 4/23/2021

The security of software is a vital thing to create and maintain, especially in cases of dealing with especially vulnerable data. Secure software can be described as "software developed or engineered in such a way that its operations and functionalities continue as normal even when subjected to malicious attacks. The systems and resources in its environment remain safe and the attacks detected and remove" [2]. Software security is the concept of implementing practices to ensure the protection of software against malicious attacks and hackers. It is crucial to have in order to protect data and integrity, as one small mistake can lead to a great loss. There are three processes involved in this practice: designing, creating, and testing security software [1]. Many different methods exist that can allow for software developers to ensure that their projects are as safe as possible. While it can sometimes mean more work, it makes sense to go through the extra steps required to ensure the safety of the program, especially in cases of protecting vulnerable data.

Some notable events of mishaps involving software security have happened in recent years. Just ten years ago, Sony was the victim of a simple SQL injection attack by LulzSec (a hacker group). Approximately one million user accounts had private information released, which violated the company's privacy policy. Specific of the personal data released included passwords, email addresses, home addresses, birth dates, and more. Situations like this have happened to other big companies as well in years since then. Once this happens to a company, a customer has to consider whether they can trust that company and their software, which is why it is so important for a company to consider how integrable their software is. An important thing to consider is prevention rather than curing. At the point of the designing stage, security of software should be kept in mind [1]. Risk assessments should be performed at each stage of the developmental process.

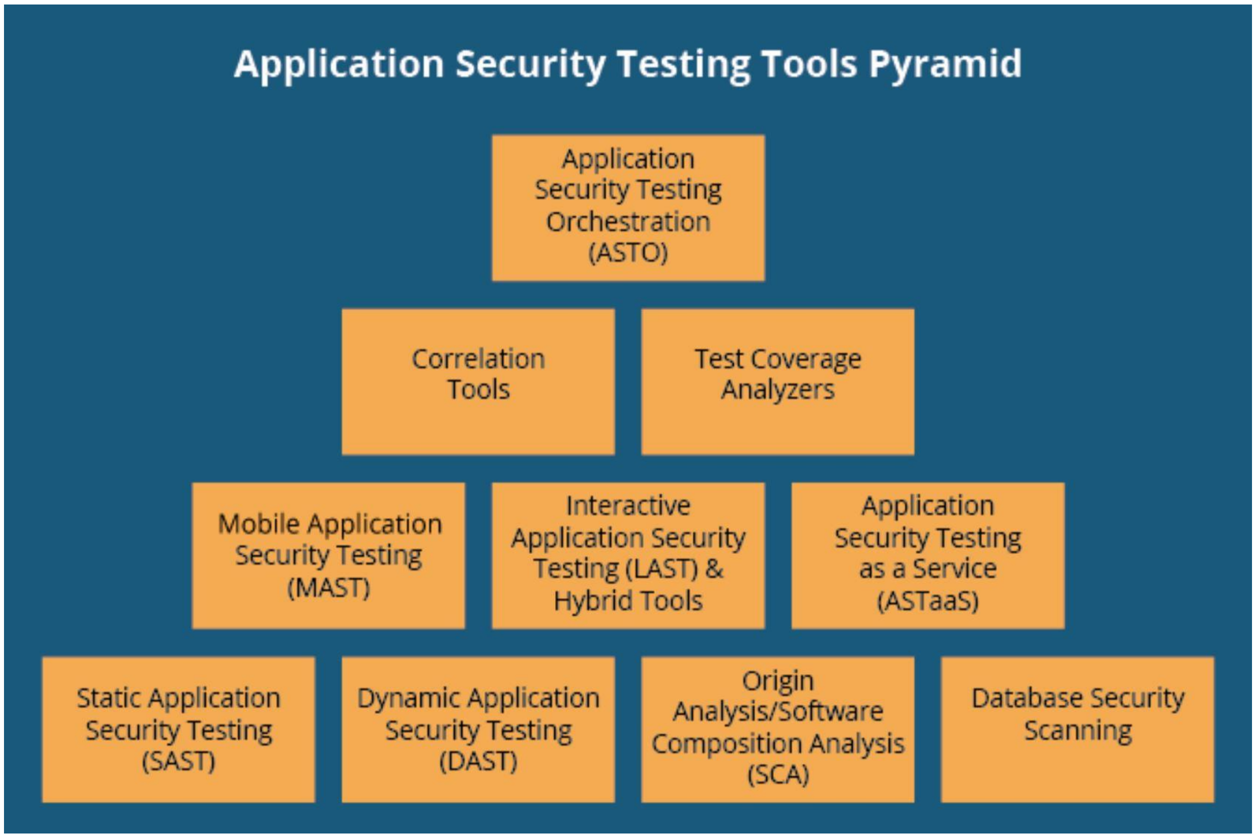
There are many different tools that can be used to ensure software is secure. Due to the fact that "84 percent of software breaches exploit vulnerabilities at the application layer", it is important to implement application security tools [3]. A benefit in these kinds of tools vs manual code reviews and the more traditional testing plans are that it is not as time consuming and is well-suited considering the continuous introduction of new vulnerabilities. They also increase speed, efficiency, and coverage paths. Tests are able to be repeated and used on many lines of code. A few examples of this form of tool is Static Application Security Testing, Dynamic Application Security Testing, and Mobile Application Security Testing.

Static Application Security Testing is sometimes referred to as "white-hat" or "white-box" testing. In this case, the tester has knowledge of the information about the system/software that is being tested and has access to everything including the source code, diagrams, etc. If someone is analyzing the source code, they can be checking for "defects such as numerical errors, input validation, race conditions, path traversals, pointers and references, and more" [3]. Other code analyzers such as for binary and byte-code will do the same except it will be for build/compiled code.

On the complete opposite spectrum, there is Dynamic Application Security Testing, which is sometimes referred to as "black-hat" or "black-box" testing. In this case, the test does not have any knowledge of the software they are testing. These tools will utilize fuzzing, which is the act of "throwing known invalid and unexpected test cases at an application, often in large volume" [3]. The goal is to locate any potential conditions of the software that may suggest security vulnerabilities of the application in its running state.

Another important tool is Mobile Application Security Testing. These tools are a combination of static, dynamic, and forensics analysis. They are specifically enabled to have the ability to analyze issues that are specific to mobile applications, such as jail-breaking, spoofing of WI-FI connections, handling/validation of certificates, attempting to avoid data leakage, etc.

Different projects will require different tools so it is important to consider what is specifically appropriate each time. Unfortunately, there are cases where security is not taken into account as it should be. A study performed by Microsoft in 2013 found that 76 percent of developers in the United States were not using secure application-program processes. This study also found that 40% of software developers globally did not feel that security was their top priority.


Application Security Testing Tools Pyramid
Application Security Testing Orchestration (ASTO)
Correlation Tools
Test Coverage Analyzers
Mobile Application Security Testing (MAST)
Interactive Application Security Testing (LAST) & Hybrid Tools
Application Security Testing as a Service (ASTaaS)
Static Application Security Testing (SAST)
Dynamic Application Security Testing (DAST)
Origin Analysis/Software Composition Analysis (SCA)
Database Security Scanning

There are some things that can be kept in mind to aid in the build of secure software. For starters, it is crucial that input is tested rigorously. A potential attacker may try to find a path into your software by sneaking through the input. One way this can be done is through "buffer overflow" [4]. Suppose the program allows for any input of string characters up to the point of the string hitting zero, as it is the official C symbol meaning last character, then all an attacker has to do is send an "arbitrarily long packet of data" and then they are able to write over the programming stack and memory. They are able to do this as long as they do not send zero, as it would cause termination. Depending on how they write it, they may even be able to take complete control and rewrite whatever they wish. It is a good idea to only store what are considered to be absolute musts. For example, in cases where you may be asking your customer/user for personal information such as a mailing address, phone number, email, or any other form of vulnerable data, it is important to ask yourself if you will truly be utilizing that data. Reasons for this include that the extra data will also mean extra time in processing, taking up space, as well as putting customers in a more vulnerable position in the case of someone successfully attacking the software. An excellent practice of keeping attackers at bay is to not rely solely on a password as a means of proving right of access. N-factor authentication can make attacking more difficult. Different methods can be utilized such as having a code sent to the user's phone number on file, keeping track of IP addresses so as to recognize when an unfamiliar IP address is being used, using hardware to lock up cryptographic keys, etc. With each requirement added to a project, a team should consider how that requirement may open the door to attacks. Implementing delays in code can assist in ensuring that bots will face more struggle. For example, if for each incorrect password entered there is a doubled delay, the successes of attacking will not be so easily achieved. When going from one area to another, especially in cases where one will be moving to more sensitive data, a good security measure to implement is creating a wall. Only requiring one login can give greater ease of access to the wrong people, but forcing someone to continue to verify their rights to have access to information will help to safe guard that information from those who should not have it. When designing and considering what information you will hold, it is wise to consider who your potentially threats may be, and from there you can decide the best ways to fight them off.

In conclusion, secure software should be able to withstand attacks and maintain its functionality. Practicing software security is essential in all stages of a developmental process. A good way to think of software security is by comparing it your car [2]. An unlocked car will be very easy to break into, of course, and a locked car will hold off an intruder. A car that uses a manual key will be easier to steal than a car that requires a smart key. Regardless of the key's form, the security it provides cannot necessarily completely evade the car of being in harm's way or maybe even being broken into or stolen, but measures can be taken to help avoid and prevent it. There are many tools, practices, and suggestions available to ensure that your software is as protected as possible. When deciding on which to use, it is vital to consider who your potential attackers may be so that your methods are as successful as possible. It is worth the extra effort to implement such preventative measures for the sake of your integrity as well as protection of important data.

Works Cited

[1] company, The first software maintenance, and APIBEST LLC © 2013 – 2021. "The Importance of Security in Software Development: APIBEST™." *APIBEST*, apibest.com/blog/importance-security-software-development#:~:text=This%20is%20why%20software%20security,prevention%20is%20better%20than%20cure.&text=And%20most%20importantly%2C%20after%20the,new%20type%20of%20malicious%20attack.

[2] Griffin, Lyna. "Secure Software: Definition & Characteristics." *Study.com*, study.com/academy/lesson/secure-software-definition-characteristics.html.

[3] Scanlon, Thomas. "10 Types of Application Security Testing Tools: When and How to Use Them." *SEI Blog*, 9 July 2018, insights.sei.cmu.edu/blog/10-types-of-application-security-testing-tools-when-and-how-to-use-them/.

[4] Wayner, Peter. "Safeguard Your Code: 17 Security Tips for Developers." *InfoWorld*, InfoWorld, 4 Feb. 2013, www.infoworld.com/article/2078701/safeguard-your-code--17-security-tips-for-developers.html.