Secure Software Design

Daniel M Beck

UMGC – SDEV-360

July 13, 2020

Secure Software Design

Introduction

With every software development comes the need for security measures. The following principles apply to the general security of software. These guidelines are just a few of the many that influence the decisions made in developing software at every phase of the software life cycle.

Minimize the Number of High-Consequence Targets

"High-consequence targets are those that represent the greatest potential loss if the software is compromised, and therefore require the most protection from attack" (Enhancing the Development Life Cycle to Produce Secure Software, 2008). The software being developed should contain the minimal number of high-consequence targets as possible. Critical and trusted components are considered high consequence target since the impact of their compromising would be the most damaging.

Separation of Privileges, Duties and Roles

Separation of privilege, duties and roles is an information technology best practice applied by organizations to broadly separate users and processes based on different levels of trust, needs, and privilege requirements (Miller, 2019). This practice allows businesses to contain intruders as close to the point of compromise as possible, restricting lateral movement. It also ensures that certain roles within the development do not have access to more data than necessary. Separating the privileges and tasks associated with the software also provides the benefit of a simpler compliance and makes it easy to trace where an attack may have occurred.

An example of how this principle would be effective would be if a user, who is logged in as an administrator for an account, clicks on a phishing email, allowing their account to become

compromised. Consequentially, attacker will now have the same privileges of that account as the administrator. If the administrator has access to all aspects of the software, the damage done could be devastating.

Do Not Expose Vulnerable or High-Consequence Components

"The critical and trusted components the software contains should not be exposed to attack. In addition, known vulnerable components should also be protected from exposure because they can be compromised with little attacker expertise or expenditure of effort and resources" (Enhancing the Development Life Cycle to Produce Secure Software 2008). The exposing of these components can cause irreputable damage that may lead to a complete compromise of the software.

Minimize the Number of Entry and Exit Points

Although there is no such thing as invulnerable code, it is possible to make systems more secure. An effective method of doing this is to minimize the number of entry and exit points. "A system's attack surface is generally defined as all the points through which data can enter or exit that system" (Minimize Software Attack Surfaces for Stronger Security). Although it is a quite simple approach, adding up the entry and exit points that are in resources being used can show the developers what kind of risks they may face.

A tool that can be used to help minimize these risks was developed by Microsoft, called the Attack Surface Analyzer 2.0. This tool can help developers "identify potential security risks introduced by changes to an operating system's security configuration by identifying changes in key areas, including:

- File System
- User Accounts

- System Services
- Network Ports (listeners)
- System Certificate Stores
- Windows Registry" (Announcing the all new Attack Surface Analyzer 2.0, 2020)"

This tool can be vital to ensuring that the software being developed can withstand multiple security threats.

Deny Attackers the Means to Compromise

The software should not be susceptible to exploitable weaknesses, vulnerabilities, dormant code, or backdoors. The software should also have the ability to minimize damage, recover, and reconstitute as quickly as possible following an attack. This requires implementing a way to monitor, record, and react to how the software behaves.

Simplify the Design

"In the design of a system, one of the strongest influences on that system's security is the complexity of its design" (Perrin, 2011). Software that is more complex tends to generate more complex interactions between different sections of the software. As these interactions increase in complexity, an increase in the potential for unforeseen and unsafe effects also exists. Simplifying the design keeps the code easier to follow for other developers, optimizing readability with a more understandable design. Furthermore, the code becomes easier to read, maintain, improve, and debug.

The implementation of open source software presents many potential security benefits and design simplicity. The concept of dilettantism, otherwise known as the principle of "many eyes", becomes a viable option when considering the complexity of your software design. If there is a feature that has used commonly throughout many applications, there is a good chance

that, any developers have adjusted features so that there are minimal amount a security faults and a high chance of simplicity.

Design for Secure Failure

It is common that at some point in time, software will have a point of failure. It becomes very important that a program can fail securely when a successful attack occurs. When the system is hacked, the attacker should not be able to gain access that would allow critical damage to the system and the access to information should be expunged quickly.

Exception handling in designs for secure failure. “The exception handling logic should always attempt to take corrective action before a failure can occur, and to allow thresholds to be set to indicate "points of no return" beyond which recovery from a fault, vulnerable state, or encroaching failure is recognized to be unlikely or infeasible” (Krutz & Fry, 2009). When an attack occurs, the exception handler should allow the program to enter a secure failure state, protecting the software's program, control data and other sensitive data or resources to become exposed.

Hold All Actors Responsible

It is imperative that “all attacker actions are observed and recorded, contributing to the ability to recognize and isolate/block the source of attack patterns and thereby prevent attacks from succeeding” (Krutz & Fry, 2009). This involves recording all security-relevant actions of actors when interacting with the system (auditing) and collecting digital signatures to prevent actors from later denying responsibility for an action taken (non-repudiation measures). Having all of this recorded leads to the ability to isolate/block the source of future attack patterns.

In a software-intensive system, auditing and non-repudiation measures may have to expand beyond human “users to include semi- and fully-autonomous software entities, such as

agents and Web and grid services that operate without human intervention, and in some cases without human knowledge" (Enhancing the Development Life Cycle to Produce Secure Software 2008). The accurate collection of all software actors establishes a "paper trail" allowing responsibility for security violations or compromises to be traced back to the human agent that is responsible for the software entity that caused the violation/compromise.

Conclusion

As technology advances, security risks also increase, and it is important that businesses and software developers keep up with these new threats. Minimizing the number of high consequence targets, not exposing vulnerable or high consequence components, and denying attackers to mean to compromise are the foundations of a high-quality system. These general guidelines allow for developers not only to be prepared but also plan ahead.

References

Announcing the all new Attack Surface Analyzer 2.0. (2020, June 03). Retrieved July 14, 2020, from https://www.microsoft.com/security/blog/2019/05/15/announcing-new-attack-surface-analyzer-2-0/

Enhancing the Development Life Cycle to Produce Secure Software [PDF]. (2008). Data & Analysis Center for Software.

Krutz, R. L., & Fry, A. J. (2009). The CSSLP prep guide mastering the certified secure software lifecycle professional. Indianapolis: Wiley Pub.

Miller, M. (2019, March 15). How Separation of Privilege Improves IT Security. Retrieved July 13, 2020, from https://www.beyondtrust.com/blog/entry/how-separation-privilege-improves-security

Minimize Software Attack Surfaces for Stronger Security. (n.d.). Retrieved July 14, 2020, from https://www.computereconomics.com/article.cfm?id=1337

Perrin, C. (2011, January 10). Design simplicity is an important element of open source security. Retrieved July 14, 2020, from https://www.techrepublic.com/blog/it-security/design-simplicity-is-an-important-element-of-open-source-security/