**Secure Software Design**

**Student**: Patrick Walsh

**Date**: 4/10/2021

**School**: University of Maryland Global Campus

**Course**: SDEV 360 7380

**Professor**: Dr. Alla Webb

**Minimize the number of high consequence targets**

This first of the three General Principles of Secure Software Design is aimed at reducing the amount of damage any one individual can inflict upon a system. Just as the nuclear launch codes are not given to one individual but instead are distributed among several individuals, so one person should not have total control over a system. In a software design context, no single individual should be able to bring a system down or do irreparable damage. This reduces the potential for an insider wreaking havoc either intentionally or unintentionally, as well as the possibility of an outside attacker gaining inside privileges and doing harm. This General Principle can be implemented in the following Key Practices: The Principle of least privilege, and the Separation of privileges, duties, and roles.

1. *Principle of least privilege.*

This widely ascribed software design practice seeks to reduce the amount of harm any one person can inflict by granting that individual as little privileges over the system as possible. Microsoft emphasized the importance of granting the least privilege, pointing out that attackers are first and foremost concerned with gaining privilege over a system (Wolter, 2021). For example, database users have the potential to cause a huge amount of damage to an organization as databases typically store critical information like user login credentials, payroll and financial data, Personally Identifiable Information (PII) on customers and employees, and sensitive company data. For this reason, database administrators should grant users only the minimal access to the database required to carry out their duties. This means that the accounting department may only need SELECT or read access to certain financial tables, INSERT or write access to other tables, and no access to tables not related to finance. The accounting department probably has no need to have read or write access to user login credentials, so a security-minded database administrator will not grant accounting members access to those tables in the database. If users required temporary permission to alter a database table like adding a column to the table, administrators should grant users permission to only alter those specific tables, but not other permissions like the ability to drop a table altogether (Wolter, 2021). With these basic concepts in mind, administrators can greatly reduce the amount of damage done to systems when security breaches occur.

1. *Separation of privileges, duties, and roles.*

Similar to the Principle of least privilege, this practice seeks to reduce the amount of damage which can be inflicted on a system by separating user privileges based on that user’s duties or role. Using separation of privileges, a user could be given multiple levels of access, provided that these levels of access are separated according to the specific roles or job functions that the user performs at any given time (Miller, 2018). For example, a user could have a developer account which grants them access to read and make changes to code in a testing environment. The user could also have a project manager or senior developer account which grants them the ability to push code to the production environment. The developer account does not have access to the production environment, and the project manager or senior developer account does not have access to the testing environment. In this way, the practice of least privilege and separation of privileges based on duties and roles are both being followed. In the scenario when one person has both accounts, the principles are still being followed because a malicious user would have to gain access to both accounts in order to inflict maximum damage (Miller, 2018).

**Don’t expose vulnerable or high consequence components**

The second General Principle of Secure Software Design calls for separating the components of software into various pieces and ensuring that the most vulnerable of these pieces are not subject to attack. Just as separating privileges based on user roles helps prevent systems from extensive damage, so separating the software components helps to achieve a similar aim. In a secure software architecture, trusted entities should be isolated from untrusted entities, and a compromise on one component should not expose all other components to exploitation. There are various key practices that seek to implement this principle, but one important key practice is minimizing the number of entry and exit points in a system.

1. *Minimize the number of entry and exit points.*

As the name suggests, minimizing the number of entry and exit points is a key practice in building secure software. It is analogous to having a secure building with only one or two ways of entering or leaving the building. The more entry or exit points, the more difficult it is to protect against unauthorized access. Another way of expressing this concept is that it reduces the attack surface. In the context of software, an entry point is a component that accepts input from a user, while an exit is a response from the system back to the user (Kumar, 2016). Some common entry points are sockets, consisting of an IP address and port number, a registry like the hierarchical database found in the Windows operating system, a programmatic interface which enables communication between applications, an environment variable which stores path configuration of various applications, or a command line argument, protocol handler, or HTTP header (Kumar, 2016). For web applications, the primary point of entry will be an HTTP header since it allows for web traffic over the internet. The most basic and formidable way to control the entry and exit of HTTP traffic is through a firewall (Kumar, 2016). A firewall acts as a gatekeeper to the software program, preventing untrusted or unsecure traffic from passing through and exposing the system to malicious attacks. Firewalls can be configured to allow only traffic through certain ports. Port 443, for example, is the port which allows secure web traffic (Mukherjee, 2020). While firewalls are a powerful tool in securing entry and exit points, they also pose a potential for exploit by attackers who can pinpoint attacks against systems (Kumar, 2016). This is why monitoring traffic that passes through firewalls is another important practice to detect vulnerabilities or efforts to penetrate the system (Kumar, 2016).

**Deny attackers the means to compromise**

The third and final General Principle of Secure Software Design is meant to avoid software systems from being exploitable in the first place, rather than simply mitigating attacks once they occur. There are several key practices that make up this principle, but the guiding principle can be thought of in terms of designing and maintaining software that presents minimal risk to attackers. This concept is comparable to physical security principles where a concrete bunker surrounded with reinforced barriers and a barbed wire perimeter is by its very nature and design more secure than a typical home. In a software security context, denying attackers the means to compromise calls for simplifying the design to minimize the number of potential vulnerabilities as well as logging attack patterns of malicious actors. Both key practices are explained below.

1. *Simplify the design.*

As Perrin pointed out (2011), more complex software requires more complex interactions between different software components, increasing the chances for problems to arise and for attackers to exploit these interactions. Studies suggest that the larger and more complex a program is, the more susceptible it is to software bugs (Perrin, 2011). One safeguard against vulnerabilities in code which can be applied to open source software is known as dilettantism, or the “principle of many eyes” (Perrin, 2011). This principle states that bugs or security flaws are less likely to prevail in code when a large number of people review the code. The obvious drawback to this principle is of course that not all projects can be open source due to proprietary or security concerns, and thus the “principle of many eyes” cannot be applied.

1. *Hold all actors accountable.*

This final key practice seeks to enforce accountability through auditing and logging activities as well as non-repudiation measures. Logging and auditing are closely related, since an audit is difficult to carry out without proper logs being kept. Microsoft is one vendor that offers a robust set of tools in the Azure cloud suite that enable logging and auditing, including activity logs, resource logs, user login activity logs, data storage logs, system exceptions logs, diagnostics logs, and security alert logs (Microsoft, 2019). In the event of a data breach, for example, it is critical to understand how an attack was carried out, the source of the attack, the attack method, the attack vector, which systems were compromised, and the extend of the damage. Without secure logging of all this activity, organizations could be blind to the nature of the attack or how to prevent such an attack from occurring in the future. This analysis after an attack is referred to as an audit, but an audit could also take place as routine security measures to find and patch weaknesses before they become exploited. Good logging practices also allow for non-repudiation, which is a term used to describe validating the origin and integrity of data (Cryptomathic, n.d.). One form of non-repudiation is digital signatures which help to verify the identity of one or more parties in an online interaction (Cryptomathic, n.d.).

**References**

Cryptomathic. (n.d.). *What is non-repudiation?* Cryptomathic.Com. https://www.cryptomathic.com/products/authentication-signing/digital-signatures-faqs/what-is-non-repudiation

Kumar, A. J. (2016, October 15). *Discovering Entry Points*. Infosec Resources. https://resources.infosecinstitute.com/topic/discovering-entry-points/

Microsoft. (2019, October 31). *Azure security logging and auditing*. Microsoft Docs. https://docs.microsoft.com/en-us/azure/security/fundamentals/log-audit

Miller, M. (2018, March 15). *How Separation of Privilege Improves IT Security*. BeyondTrust. https://www.beyondtrust.com/blog/entry/how-separation-privilege-improves-security

Mukherjee, L. (2020, June 18). *Port 443 — Everything You Need to Know About HTTPS 443*. InfoSec Insights. https://sectigostore.com/blog/port-443-everything-you-need-to-know-about-https-443/

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

Wolter, A. (2021, February 3). Security: The Principle of Least Privilege (POLP). Microsoft Tech Community. https://techcommunity.microsoft.com/t5/azure-sql/security-the-principle-of-least-privilege-polp/ba-p/2067390