Secure Software Design: Principles and Practices

A Report By

Mark Kardash

SDEV360: Secure Software Engineering

Professor David Gabello

15 November 2023

University of Maryland Global Campus

Software of any type must always be secure. However, security is especially important during the time a program or application is being developed. There is both a positive and a negative reason for this. If a critical flow is overlooked during development, the company may face significant backlash should it surface after release, resulting in product recalls, or even having legal consequences should user data be compromised. But here, the positive aspect comes in, as, by focusing on security early in development, the creators of an application or webpage have a chance to detect such flaws, preventing the grim scenario described. To ensure that their product is secure from the very beginning, companies follow a set of general principles for secure software design. This report will explain these principles in more detail, diving deeper into some of their key practices, and providing several examples of their functionality.

The first principle of secure software design is minimizing the number of high-consequence targets. According to Jackson et al (2017), “Minimization is the Principle of reducing needless size, complexity, and overly burdensome assets.” (Chapter 4: Minimization, para.1). In simpler terms, this principle involves reducing the number of an application’s parts that can bring significant consequences if attacked. The principle has several key practices within it, but, for the sake of conciseness, this paper will only focus on one called Principle of Least Privilege. Once again, its name is quite self-explanatory. Sometimes also called “least privilege access”, this practice only allows users to access parts of an application that are necessary for them. Such an approach ensures that, if the user’s account is hacked, or if they themselves have malicious intent, they would not be able to inflict significant damage. The fewer details are revealed to the world about a software product, the smaller the chance of a cyberattack (Cloudflare, n.d.). The practice may be understood better via the following example: In some cases, companies rely on third-party contractors to conduct work for them. While it may seem beneficial at first, this approach can turn out quite dangerous. For instance, if a company grants a third-party maintenance access to a server, and the employees of the company that sent them work in groups, dozens of people will now have access to credentials they were never meant to know. And out of these people, anyone may have malicious plans (Hoffman, n.d.). In any case, the credentials to the server of the “victim” company will now forever remain with the third-party company. This puts the former at serious risk. Since the privileges are unlimited and shared across servers, supervisors will not be able to track individual employee activities, and the possibility of lateral movement will become much greater. Additionally, if the credentials always remain the same, without an expiration date or regular updates, any employee who departs the company will still have access to the servers (Hoffman, n.d.).

There are several things that companies need to do to properly implement the Principle of Least Privilege. The first is performing a privilege audit. This is done to determine which accounts and employees have too many privileges. It also extends to any third parties involved with the software being audited. The audit will be performed both on those physically present at a company, and those working remotely (Okta, 2022). After this, “least privilege” can be made default at the company. This means revoking any unnecessary privileges found. Administrative privileges are the ones that give access to the most information, so only those needed most should be left. In general, employees must be left with the minimum privileges necessary for performing their tasks (Okta, 2022). The next step is separating accounts and privileges. This could mean separation of the standard accounts from administrative accounts, once again only granting the minimum privileges needed for a task, and restricting the ability of users to write in logs that are not monitored within the database (Okta, 2022). Other steps for implementation of the practice include temporarily allowing higher privileges for project completion, adjusting permissions based on role, analyzing and monitoring activities of privileged employees, and creating a schedule for regular review of the permissions given (Okta, 2022). The practice for implementing minimization of targets in general are not too different. Jackson et al (2017) recommend removing vulnerable and unused code from the system, deleting sensitive data after it is no longer needed, getting rid of unnecessary functions, and patching up a system as fast as possible, to not leave it vulnerable for long (n.d.).

The Principle of Minimizing High-Level Targets is all about eliminating as many vulnerable aspects of an application as possible. It keeps the system safe and secure by giving attackers minimal ground to operate on. It is, therefore, no wonder that companies use it to maintain a secure software design process.

The next principle of secure software design is not exposing vulnerable or high-consequence components. This means keeping a close eye on components of an application that are the easiest for attackers to infiltrate. Of course, as explained above, developers should keep components to a minimum. But occasionally, when a feature is so important it cannot be removed, it must be adequately protected. The principle has several best practices, two of whom will be discussed below.

Data segregation, also known as data separation, is “the process of separating certain sets of data from other data sets so that different access policies can be applied to those different data sets.” (NextLabs, para.1, 2023). This could mean separating program data from configuration data and executable parts. Data segregation is somewhat intertwined with the principle of least privilege, as it is done with the intention of allowing only certain employees to view certain datasets. There are two types of data segregation. Physical segregation involves using separate networks or servers to store data (NextLabs, 2023). It is an expensive practice, as it requires companies to install multiple networks or systems to implement it. It is most needed for data that must never leave the country it is stored in, as, in those cases, it is especially important that no unauthorized persons have access to it (NextLabs, 2023). The second data segregation type is logical segregation. This is the practice of storing data in different areas or logical partitions of the same device. In addition to being a much more feasible approach financially, logical segregation can make creating data access policies easier for companies, as they will now be able to update them virtually, thanks to the data being stored on one device (NextLabs, 2023).

A good example of data segregation would be a situation at a typical IT company: The average employee has access to some of the company’s data. For example, they may be able to search up fellow employees through the company’s website. They may even be able to view some employee information, such as full name and email. However, they should not be able to see other details, such as their colleagues’ salary or Social Security Number. This data is stored separately, and accessible only to higher-level officials, such as the Chief Financial Officer, the employer, and the individual employee themselves. In this way, companies ensure they remain fair to their employees, while also protecting their data from malicious use.

The implementation of data segregation is a multi-step process, beginning with the consideration of which data will undergo it. True to the principle of protecting vulnerable components, which this practice is a part of developers take a data-centric approach, focusing on the data that is most at-risk should an attack occur. From there, access to the application can be controlled at the data level, and using environment or user attributes are also techniques of implementing data segregation (NextLabs, 2023).

Another practice developers use to avoid exposing their product’s vulnerable components is using only trusted interfaces to access resources in an environment. According to Guillemet (2020), “A trusted user interface (UI) provides a path between the user and the application that is unambiguously free from any form of manipulation.” (What is a Trusted User Interface, para.1). Using only trusted interfaces for interactions with an application therefore ensures that when a user validates or displays sensitive information on the interface, it will not be exploited by attackers. This way, they will feel safe entering their password, address, or even Social Security Number (Guillemet, 2020). Attacks through interfaces are quite common and can be performed in a variety of ways. A very popular way is the spoofing of email addresses on both ends of a transaction. For example, an employee may want to transfer funds (with the CEO’s approval) to purchase new supplies for the company, from a trusted seller. They confirm all the details they entered, before confirming the transaction. Once this is done, the hacker changes the email address where the costs are being sent. Although the hacker has now entered their own email, the unsuspecting employee is still seeing that of the seller. The results of this scenario are disastrous, as company funds are now in the hands of criminals.

Cybercriminals can also attempt to leverage the rights of users to gain the trust of an organization. One of the most popular forms of this is clickjacking (Guillemet, 2020). It occurs when a hijacker either takes over, or creates a user interface (UI) for the victim to use. Since the interface is not secure, the user can click on a button or link that will perform malicious actions on their computer (ex. Installing a virus) (Guillemet, 2020).

The road to implementing the above techniques is slightly different from the others described here, in that they don’t necessarily have an order in which to be performed. To determine whether they can trust an interface, developers can check how well it protects confidential data. For example, whether the data is encrypted after entering it. Developers can also check whether an interface’s internal code is correctly written, without any signs of corruption or tampering (Guillemet, 2020).

Keeping vulnerable parts of the application out of reach for attackers is perhaps the most obvious way to protect software in its design phase. The practices of data segregation and using trusted interfaces both support this principle, defending the system from potential breaches.

The final principle of secure software design is denying attackers the means to compromise. This means depriving them of any potential ways they could compromise an application’s integrity. For example, by engaging in the previously discussed practice of using trusted interfaces, a developer denies compromise for hackers, since any data coming from an untrusted interface could be intentionally harmful for the software (National Cyber Security Centre, n.d.). Other ways of denying compromise would be either validating data against certain preconditions, or performing data transformation to ensure its authenticity. Data transformation is the conversion of a data file from one type into another, for easier detection of malicious code. Rendering is yet another technique of denying compromise, in which the data is rendered in an external environment to check for any dangers (National Cyber Security Centre, n.d.). Like the two principles before it, denying attackers the means to compromise has a few practices recommended for use, two of which, once again, will be examined here.

One of the most important things in upholding this principle is holding all those involved in an attack accountable for their actions. This involves not only the cybercriminals facing consequences, but also the actions of every employee being monitored for safety. As the National Cyber Security Centre of the United Kingdom puts it, “The ability to *attribute actions to individuals rather than to groups* will be important when it comes to establishing accountability. It will also aid incident response.” (2.6, para.2, n.d.). Therefore, this practice is helpful because it traces the cause of an attack to individual people, rather than abstractly attributing it to a group. For example, if a company suffers a leak of its customers’ financial information, they can investigate and analyze the specific actions of the developer or employee who was in charge of that information and determine if any vulnerabilities were exposed. It is also recommended that an access control function be used to verify user identity, before permission is granted for privileged actions (National Cyber Security Centre, n.d.). In this way, supervisors in the example given can ensure that no unauthorized persons had access to the customer finances from within the company when the attack occurred.

Implementation of this practice would require reasonable measures to monitor employee activity. This may include things like procedural and physical controls for sharing credentials from a control form, as well as physical access controls for individuals like supervisors. One of the most common methods is the use of CCTV (National Cyber Security Centre, n.d.). Having CCTV cameras to monitor employees can allow supervisors to rewind footage, returning to a particular datetime, and analyze the events that occurred to understand the exact circumstances that led to a data breach.

Denying attackers the means to compromise also involves designing for secure failure. According to Spacey (2018), designing for failure “is the practice of designing things to retain their quality in the face of failures and stresses” (para.1). In other words, it involves preparing the system to handle anything that comes its way. Using this practice, developers must approach their product with a certain “mindset”. They must consider every single error, vulnerability, shortcoming, and attack scenario that their application may face. In addition to this, they must operate with the philosophy that it will inevitably face them all, in their most damaging forms. This prediction of failures is done with the goal of designing elements that will prevent these failures from happening (Spacey, 2018). The details of this practice are also the steps needed for its implementation. To implement designing for failure, developers can enhance their product in several ways. Redundancy is very popular in the practice, with developers creating parts of an application that may seem redundant and needlessly time-consuming, but actually serve an important purpose. For instance, a platform or page that runs on hundreds of servers instead of two or three may be able to still function relatively smoothly, even if a few of the servers are compromised (Spacey, 2018). This leads into the concepts of error tolerance and cold standby. The former is a program’s ability to still run despite existing errors. The former is the existence of backups that will store sensitive data if needed. Despite potential complications, developers prefer not to make a program shut down after a single error and allow it to continue with its intended operations for as much as possible. They also may implement backups such as generators that power a company’s devices in the case of an emergency, so that critical data will not be lost (Spacey, 2018). Other things done by developers in designing for failure are monitoring errors to eliminate them, deactivating things gradually, rather than shutting off an entire program in the case of failure, making applications durable to stress, and correcting mistakes via an “undo” feature (Spacey, 2018).

The principle of denying compromise to attackers is just as important for secure software design as the previous two. It not only allows developers to make their products as secure as possible, but also requires them to use their skills to the maximum, coming up with ideas, defenses and solutions that propel software development into the future.

The concepts of software design and security cannot, and should never, be viewed separately. Designing programs without regard to security will do nothing but harm developers and companies. Data breaches, digital robberies, and identity theft will lead to loss of customer trust, lawsuits, and, if not dealt with swiftly, bankruptcy. For this reason, the three principles of secure software design must be religiously followed by developers, while research for new methods of protecting systems must also continue.

References

*2. Make compromise difficult*. (n.d.). National Cyber Security Centre. <https://www.ncsc.gov.uk/collection/cyber-security-design-principles/making-compromise-difficult#:~:text=Make%20compromise%20difficult%201%202.1%20External%20input%20can%27t,administrators%20to%20manage%20access%20control%20...%20More%20items>

Casey, J. (2018, March 3). *16 Types of Design for Failure*. Simplicable. <https://simplicable.com/design/design-for-failure>

Guillemet, C. (2020, April 26). *The importance of trusted user interfaces and secure execution*. HackerNoon - read, write and learn about any technology. <https://hackernoon.com/the-importance-of-the-trusted-display-and-secure-execution-1l7e3249>

Hoffman, B. (2023, August 10). *Principle of least privilege examples | With diagrams*. Delinea. Retrieved November 15, 2023, from <https://delinea.com/blog/principle-of-least-privilege-examples>

Jackson, C., Russell, S., & Sons, S. (2017). Chapter 4.Minimization. In *Security from first principles: A practical guide to the information security practice principles*. O'Reilly Media, Inc. <https://www.oreilly.com/library/view/security-from-first/9781491996911/ch04.html>

*Principle of least privilege: Definition, methods & examples*. (2022, July 21). Employee and Customer Identity Solutions | Okta. Retrieved November 15, 2023, from <https://www.okta.com/identity-101/minimum-access-policy/>

*What is data segregation?* (2023, October 24). NextLabs. Retrieved November 15, 2023, from <https://www.nextlabs.com/what-is-data-segregation/>

*What is the principle of least privilege?* (n.d.). Cloudflare - The Web Performance & Security Company | Cloudflare. <https://www.cloudflare.com/learning/access-management/principle-of-least-privilege/>