Secure Design Principles

Revision

Version 5
11/15/21 8:13 AM

SME

Charles Wilson

Abstract

This document describes a set of design principles that are broadly applicable toward the development of secure software and systems.

License

Overview

It is insufficient to simply provide developers with a to-do list of security rules and expect that the result will be secure. To ensure the highest level of security, it is necessary to have a secure design. There are broadly applicable design principles which can greatly improve the security posture of a system.

Note: It’s important to keep in mind that there is no perfect security. Once we accept this, we can create designs that consider the eventuality of a security compromise.

Principles

The following design principles are presented in a mostly hierarchic manner. They are listed in order of conceptual import and implementation.

Modularize

Monolithic designs are harder to analyze, build, debug, test, and defend. Partition the system into logical functional elements that can be addressed separately.

Minimize Surface Area

It's easier to defend a doorway than an entire building. Keep the number of interfaces to each element as small as possible. Nothing is gained by leaving a currently unused interface active except to an adversary. If an interface isn't in use, it will probably not be properly updated.

Isolate Security

Whether used as a stage in a communications pipeline or to secure discrete blocks of data, the security elements should be isolated. There should be no cases where manual manipulation of security (or cryptographic operations) is done in non-dedicated code. Interaction with security elements should be via opaque objects managed entirely by the security element.

Isolate Privilege

In the same manner as security is isolated, activities which require elevated privilege should also be. The reasoning is similar in that we are reducing the possibility for abuse of privilege.

Use Least Privilege

Processes, even modular, isolated ones with minimal surface area need to have only the privileges necessary to accomplish their task and no more. Some may argue that you never know what you may need in the future or that without additional privileges, debugging is impossible. These assertions are indicative of failure to implement the previous principles.

Use Hierarchical Trust

The system should be architected is such a way as to establish a hierarchy of trust zones. This enables the minimization of controls necessary between elements at the same level of trust. It also means that only the higher-trust zone need establish additional cybersecurity measures.

Limit Lifetimes

This can refer to either data or code. The basic principle is the minimization of temporal surface area. Don’t leave resources available to possible abuse before and after they are needed.

Use Standards-based Cryptography

There is a temptation to use hand-rolled cryptography in cases believed to be too trivial or too complex for existing technology. In virtually every case, this is the wrong decision. Use standards-based cryptography that has been certified as appropriate to the intended task.

Use Adaptive Cryptography

The cryptographic algorithms and key lengths that were sufficient to protect information ten years ago are not considered so today. The same will hold for the cryptographic protections we consider sufficient today when viewed from the future. It is essential to use cryptographic mechanisms in a way that allows for clean upgrading. This means avoiding the use of fixed length structure allocations for keys. It also means that where multiple algorithms and key lengths are available during negotiation of communication channels, deprecated ones are removed from consideration to avoid compromise.

Use Trusted Communication Channels

When data is passing between processes at different trust levels, use appropriate security controls. This may be as trivial as standard user-system API-based communication, or as complex as encrypted payloads sent via authenticated and secured protocols.

Enable Auditing

As the overview states, “consider the eventuality of a security compromise.” Building an audit system parallel to the functional system enables both incident detection and post mortem analysis.

Note: This is not the same as data logging and should be kept separate.

Degrade as Necessary

To reduce the likelihood of a catastrophic failure, the system should be designed in such a way as to allow for non-critical elements to be disabled in order to maintain the core operation of the system.

References

1. **The Protection of Information in Computer Systems** https://www.cs.virginia.edu/~evans/cs551/saltzer/
2. **Industrial Cybersecurity** (Ackerman)
3. **NIST SP 800-160 v1 – Systems Security Engineering:** ***Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems*** **(Appendix F – Design Principles for Security)** https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v1.pdf