Security Design Principles: Economy of Mechanism Minimize Common Mechanism Isolation, Separation and Encapsulation

CS-3113: PRINCIPLES OF CYBER SECURITY

BENJAMIN R. ANDERSON

Review: The Eleven Design Principles

General/Fundamental Design Principles

1. Simplicity (related to Economy of Mechanism but not exactly the same)
2. Open Design
3. Design for Iteration
4. Least Astonishment

Security Design Principles

5. Minimize Secrets (not specifically identified by Saltzer and Schroeder)
6. Complete Mediation
7. Fail-safe Defaults
8. Least Privilege
9. ***Economy of Mechanism***
10. ***Minimize Common Mechanism (related to Least Common Mechanism)***
11. ***Isolation, Separation and Encapsulation***

Review: Saltzer and Schroeder's Security Design Principles

Many security issues are a result of poor coding techniques which lead to flaws in the program which can result in a security hole that can be exploited

Saltzer and Schroeder came up with a list of 8 security design principles that, if followed, would help programmers reduce the number of errors and design more secure software

1. ***Economy of mechanism – A simple design is easier to test and validate***
2. Fail-safe defaults –In computing systems, the safe default is generally "no access" so that the system must specifically grant access to resources
3. Complete mediation – Access rights are completely validated every time an access occurs
4. Open design –secure systems, including cryptographic systems, should have unclassified designs
5. ***Separation of privilege – A protection mechanism is more flexible if it requires two separate keys to unlock it, allowing for two-person control and similar techniques to prevent unilateral action by a subverted individual***
6. Least privilege – Every program and user should operate while invoking as few privileges as possible
7. ***Least common mechanism – Users should not share system mechanisms except when absolutely necessary***
8. Psychological acceptability – users won't specify protections correctly if the specification style doesn't make sense to them

Economy of Mechanism

There are many definitions, but one is:

- The design of the system should be small, simple, and straightforward. Such a system can be carefully analyzed, exhaustively tested, perhaps verified, and relied on.

Matt Bishop, in *Computer Security: Art and Science* defines ***Economy of Mechanism*** as:

- *The principle of economy of mechanism states that security mechanisms should be as simple as possible*

He goes on to state:

- *If a design and implementation are simple, fewer possibilities exist for errors. The checking and testing process is less complex, because fewer components and cases need to be tested. Complex mechanisms often make assumptions about the system and environment in which they run. If these assumptions are incorrect, security problems may result.*

There is a relation between Simplicity and Economy of Mechanism:

- ***Simplicity*** - Only do what you need to
- ***Economy of mechanism*** – Only include what you need to actually do that

Reducing the number of mechanisms necessary helps with verifying the security of a computer system – there are fewer things to test and verify

US-CERT on Economy of Mechanism

The US-CERT has an entire page dedicated to Economy of Mechanism:

- https://www.us-cert.gov/bsi/articles/knowledge/principles/economy-of-mechanism

It states:

- *One factor in evaluating a system's security is its complexity. If the design, implementation, or security mechanisms are highly complex, then the likelihood of security vulnerabilities increases. Subtle problems in complex systems may be difficult to find, especially in copious amounts of code. For instance, analyzing the source code that is responsible for the normal execution of a functionality can be a difficult task, but checking for alternate behaviors in the remaining code that can achieve the same functionality can be even more difficult. One strategy for simplifying code is the use of choke points, where shared functionality reduces the amount of source code required for an operation. Simplifying design or code is not always easy, but developers should strive for implementing simpler systems when possible.*

This includes a number of definitions (including Matt Bishop's)

Saltzer and Schroder:

- Economy of mechanism: Keep the design as simple and small as possible. This well-known principle applies to any aspect of a system, but it deserves emphasis for protection mechanisms for this reason: design and implementation errors that result in unwanted access paths will not be noticed during normal use (since normal use usually does not include attempts to exercise improper access paths). As a result, techniques such as line-by-line inspection of software and physical examination of hardware that implements protection mechanisms are necessary. For such techniques to be successful, a small and simple design is essential.

More from Matt Bishop:

- *Interfaces to other modules are particularly suspect, because modules often make implicit assumptions about input or output parameters or the current system state; should any of these assumptions be wrong, the module's actions may produce unexpected, and erroneous, results. Interaction with external entities, such as other programs, systems, or humans, amplified this problem.*

They also quote Viega and McGraw:

- The KISS mantra is pervasive: "Keep It Simple, Stupid!" This motto applies just as well to security as it does everywhere else. Complexity increases the risk of problems. Avoid complexity and avoid problems.

  The most obvious implication is that software design and implementation should be as straightforward as possible. Complex design is never easy to understand, and is therefore more likely to include subtle problems that will be missed during analysis. Complex code tends to be harder to maintain as well. And most importantly, complex software tends to be far more buggy. We don't think this comes as any big surprise to anyone."

Citations:

- Bishop, Matt. Computer Security: Art and Science. Boston, MA: Addison-Wesley, 2003
- Saltzer, Jerome H. & Schroeder, Michael D. "The Protection of Information in Computer Systems," 1278-1308. *Proceedings of the IEEE 63,* 9 (September 1975).
- Viega, John & McGraw, Gary. *Building Secure Software: How to Avoid Security Problems the Right Way*. Boston, MA: Addison-Wesley, 2002.

Simplicity vs. Economy of Mechanism

These are related, but they are different

- Remember, a lot of cybersecurity is about nuance and detail
- This is one of the reasons it is very difficult

Economy of mechanism: A simple design is easier to test and validate

Simplicity: The more complex a system, the less assurance we may have that it will function as expected.

It is a subtle difference, and many professionals look at them the same thing

- However, this is a distinction that NSA makes
- Both address minimizing complexity, but in different ways

As an example of the distinction:

- You can have a simple design that only does one thing, but it could be a very complicated operation
  - Example: Display the location of a hypersonic missile on a radar screen
  - Given the speed, you would have to consider processing lag of the received signal, the refresh rate of the display, and the ability of the human eye
- Ideally the two will go hand-in-hand

Quote: *Fools ignore complexity. Pragmatists suffer it. Some can avoid it. Geniuses remove it.*

- Alan J. Perlis, "Epigrams in Programming" (1982)

Trade-offs

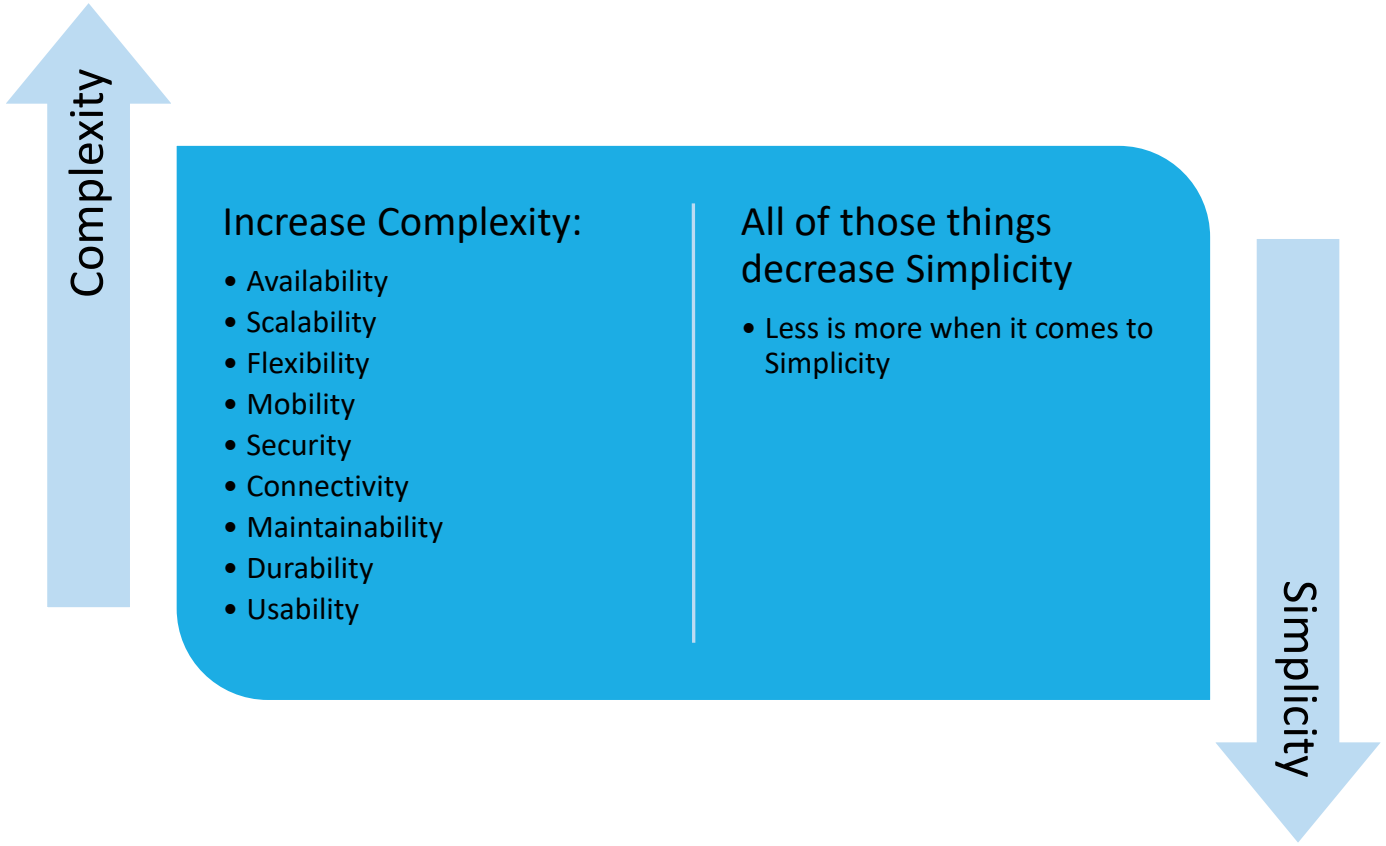
This is a situation where increasing one aspect of a system results in a decrease in another

- In the real world, there is only so much time and budget – which means you can't have all the quality that you want
  - This is captured in the "Good, fast, cheap – Pick 2" model
- To avoid wasting the "good" that the system can have, there are three rules:
  1. Maximize the goodness
  2. Avoid wasting it
  3. Allocate it to the places where it will help the most

Looking at things from a security design standpoint:

- The less there is, the more likely you will get it right
- Designing a secure system is difficult because every access path must be considered to ensure complete mediation, including ones that are not exercised during normal operation
  - Remember the errors in the code found in the previous lesson!
- As a result, techniques such as line-by-line inspection of software and physical examination of hardware implementing security mechanisms may be necessary
- For such techniques to be successful, a small and simple design is essential

Trade-offs


Complexity
Increase Complexity:
• Availability
• Scalability
• Flexibility
• Mobility
• Security
• Connectivity
• Maintainability
• Durability
• Usability
All of those things decrease Simplicity
• Less is more when it comes to Simplicity
Simplicity

Tradeoffs – Different Perspectives

Fire Chief Test:

What would you do if you went in an alley and saw a dumpster was on fire?

- Connect hose to spigot, turn water on, put out the fire.

What would you do if the dumpster was not on fire?

- Light the dumpster on fire, thereby reducing the problem to one I have already solved.

You need to keep in mind other perspectives

- What seems simple or easy from one perspective may be insanity from another
- Don’t get tunnel vision!

Tradeoffs

A quine is a computer program which takes no input and produces a copy of its own source code as its only output.

- https://en.wikipedia.org/wiki/Quine_(computing)

To the right is an example in Java from the Wikipedia page

But, if we start adding constraints, the complexity can increase

- For example, if a restriction is: Make it as small as possible you could end up with something far less readable (also from Wikipedia):
- `a='a=%s%s%s;print(a%%(chr(39),a,chr(39)))' ;print(a%(chr(39),a,chr(39)))`

```
public class Quine
{
  public static void main(String[] args)
  {
    char q = 34;      // Quotation mark character
    String[] l = {    // Array of source code
    "public class Quine",
    "{",
    "  public static void main(String[] args)",
    "  {",
    "    char q = 34;      // Quotation mark character",
    "    String[] l = {    // Array of source code",
    "    ",
    "    };",
    "    for(int i = 0; i < 6; i++)           // Print opening code",
    "        System.out.println(l[i]);",
    "    for(int i = 0; i < l.length; i++)    // Print string array",
    "        System.out.println(l[6] + q + l[i] + q + ',');",
    "    for(int i = 7; i < l.length; i++)    // Print this code",
    "        System.out.println(l[i]);",
    "  }",
    "}",
    };
    for(int i = 0; i < 6; i++)           // Print opening code
        System.out.println(l[i]);
    for(int i = 0; i < l.length; i++)    // Print string array
        System.out.println(l[6] + q + l[i] + q + ',');
    for(int i = 7; i < l.length; i++)    // Print this code
        System.out.println(l[i]);
  }
}
```

Signs of Complexity

There are some signs that a system is going to have high complexity:

- Large number of components
- Larger number of interconnections
- Many irregularities
- A long description
- A team of designers, implementers, or maintainers

Remember: To combat this we can use the following techniques:

- modularity,
- abstraction,
- hierarchy, and
- layering

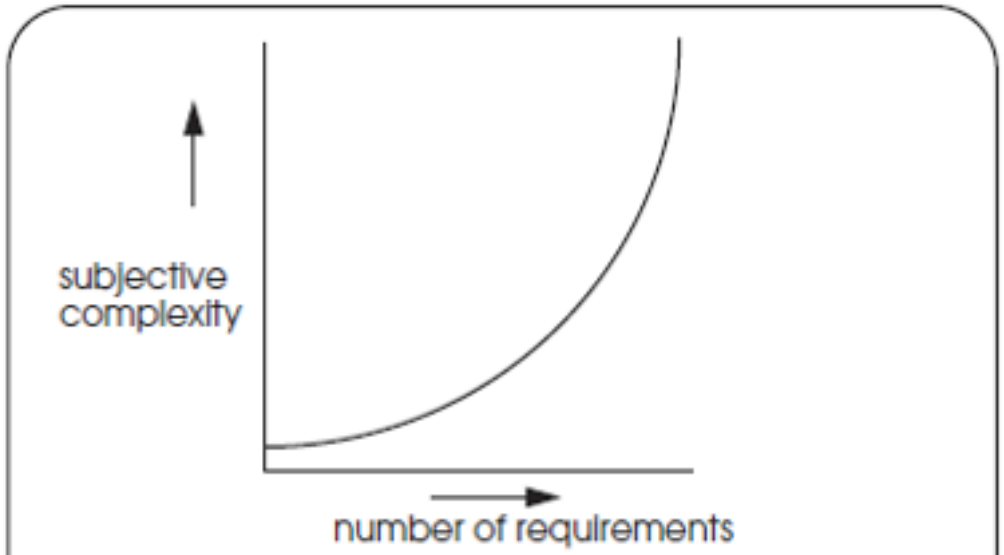


Figure 1.1: The principle of escalating complexity.

Coping With Complexity

"A camel is a horse designed by committee."

- Sir Alec Issigonis, designer of the original Mini

"When in doubt, make it stout, and of things you know about. When in doubt, leave it out."

- Saying from the automotive industry

Remember what was said previously about scope creep

Coping With Complexity

The Bradley Fighting Vehicle illustrates how a system can change when there are changing – and competing – requirement

(Optional) Watch: *The Pentagon Wars*

- This is the Bradley Fighting Vehicle Evolution scenes from the movie
- This is based on a true story
- ***Warning***: There is some inappropriate language so watching this clip is optional
- https://www.youtube.com/watch?v=aXQ2lO3ieBA&t=659s

Wikipedia article on the Bradley:

- https://en.wikipedia.org/wiki/Bradley_Fighting_Vehicle

The Space Shuttle also had a contentious design process

- There were a number of competing requirements from different government agencies
- Resulting in numerous, less than optimal, design decisions
- ***Read***:
  - https://en.wikipedia.org/wiki/Space_Shuttle_design_process
  - Read the sections:
    - Shuttle design debate
    - Air Force involvement
    - Final Design

Least Common Mechanism

Matt Bishop, in *Computer Security: Art and Science* defines ***Least Common Mechanism*** as:

- *The principle of least common mechanism states that mechanisms used to access resources should not be shared.*

He goes on to state:

- *Sharing resources provides a channel along which information can be transmitted, and so such sharing should be minimized. In practice, if the operating system provides support for virtual machines, the operating system will enforce this privilege automatically. Otherwise, it will provide some support (such as a virtual memory space) but not complete support (because the file system will appear as shared among several processes).*

US-CERT on Least Common Mechanism

As with Economy of Mechanism, the US-CERT has an entire page dedicated to Least Common Mechanism:

- https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/least-common-mechanism

It states:

- *Avoid having multiple subjects sharing mechanisms to grant access to a resource. For example, serving an application on the Internet allows both attackers and users to gain access to the application. Sensitive information can potentially be shared between the subjects via the mechanism. A different mechanism (or instantiation of a mechanism) for each subject or class of subjects can provide flexibility of access control among various users and prevent potential security violations that would otherwise occur if only one mechanism was implemented.*

More by Saltzer and Schroder:

- Least common mechanism: Minimize the amount of mechanism common to more than one user and depended on by all users.1 Every shared mechanism (especially one involving shared variables) represents a potential information path between users and must be designed with great care to be sure it does not unintentionally compromise security. Further, any mechanism serving all users must be certified to the satisfaction of every user, a job presumably harder than satisfying only one or a few users. For example, given the choice of implementing a new function as a supervisor procedure shared by all users or as a library procedure that can be handled as though it were the user's own, choose the latter course. Then, if one or a few users are not satisfied with the level of certification of the function, they can provide a substitute or not use it at all. Either way, they can avoid being harmed by a mistake in it.

Citations:

- Bishop, Matt. Computer Security: Art and Science. Boston, MA: Addison-Wesley, 2003
- Saltzer, Jerome H. & Schroeder, Michael D. "The Protection of Information in Computer Systems," 1278-1308. *Proceedings of the IEEE 63,* 9 (September 1975).

Least Common Mechanism

Distributed computing can also play a role in Least Common Mechanism

- In a Distributed Denial-of-Service (DDoS) attack, if services are distributed, only some aspects of the system will be down
- This was touched on in our discussion on cloud computing
  - If an attack took down an Amazon autonomous zone (AZ) other AZ's would still be up and able to provide services
  - This is because each AZ has its own servers, network infrastructure, power, etc.

Covert Channels

ISACA, formerly the Information Systems Audit and Control Association (they only use the acronym now – like KFC) defines a cover channel as:

- *A covert channel is any communication channel that can be exploited by a process to transfer information in a manner that violates the systems security policy. In short, covert channels transfer information using non-standard methods against the system design.*
- https://www.isaca.org/resources/news-and-trends/isaca-now-blog/2017/understanding-covert-channels-of-communication

There are many different methods for creating a covert channel, the most commonly used are:

- **Storage Channels** – modifying a "storage location" that the sender and the receiver can view/modify
- **Timing Channels** – perform operations that affect timing of a certain function
- **Network Channels** – transfer information "hidden" in network traffic
  - For example, DNS data exfiltration
  - ***Watch***: https://www.youtube.com/watch?v=fQ4Y8napHzw
- ***TCSEC (the Trusted Computer System Evaluation Criteria) defines only the first two kinds of channels***

Covet Channels

While the three methods on the previous slide are the generally accepted methods, anything can become a covert channel to a creative attacker

Recall the XKCD comic on workflow

Two isolated processes could communicate through the change in CPU temperatures

- A higher temperature (from executing computing-intense code) could be a ‘1’
- A lower temperature (from executing low-intensity code) could be a ‘0’
- It might be relatively slow – but it is still a covert channel

Image from XKCD, https://xkcd.com/1172/

Covert Channels

Covert channels are difficult to detect – which is why we use the term ***covert***

From Wikipedia:

- *Ordinary things, such as existence of a file or time used for a computation, have been the medium through which a covert channel communicates. Covert channels are not easy to find because these media are so numerous and frequently used. Two relatively old techniques remain the standards for locating potential covert channels. One works by analyzing the resources of a system and the other works at the source-code level.*
- https://en.wikipedia.org/wiki/Covert_channel

More from Wikipedia (same page):

- *The possibility of covert channels cannot be eliminated, although it can be significantly reduced by careful design and analysis.*
  The detection of a covert channel can be made more difficult by using characteristics of the communications medium for the legitimate channel that are never controlled or examined by legitimate users. For example, a file can be opened and closed by a program in a specific, timed pattern that can be detected by another program, and the pattern can be interpreted as a string of bits, forming a covert channel. Since it is unlikely that legitimate users will check for patterns of file opening and closing operations, this type of covert channel can remain undetected for long periods.
  A similar case is port knocking. In usual communications the timing of requests is irrelevant and unwatched. Port knocking makes it significant.

Data Loss Prevention

Watch: *Covert Channels* from CISSP Free by Skillset.com

- https://www.youtube.com/watch?v=YgZJ5z7Hmhw&t=164s

This talks about covert channels – and gets into Data Loss Prevention

Data Loss Prevention:

- *Data loss prevention (DLP) is a set of tools and processes used to ensure that sensitive data is not lost, misused, or accessed by unauthorized users. DLP software classifies regulated, confidential and business critical data and identifies violations of policies defined by organizations or within a predefined policy pack, typically driven by regulatory compliance such as HIPAA, PCI-DSS, or GDPR. Once those violations are identified, DLP enforces remediation with alerts, encryption, and other protective actions to prevent end users from accidentally or maliciously sharing data that could put the organization at risk.*
- Digital Guardian: https://digitalguardian.com/blog/what-data-loss-prevention-dlp-definition-data-loss-prevention

Data Loss Prevention

Categories of DLP Systems from Wikipedia:

- ***Standard measures***:
  - Standard security measures, such as firewalls, intrusion detection systems (IDSs) and antivirus software, are commonly available products that guard computers against outsider and insider attacks. The use of a firewall, for example, prevents the access of outsiders to the internal network and an intrusion detection system detects intrusion attempts by outsiders. Inside attacks can be averted through antivirus scans that detect Trojan horses that send confidential information, and by the use of thin clients that operate in a client-server architecture with no personal or sensitive data stored on a client device.
- ***Advanced measures***:
  - Advanced security measures employ machine learning and temporal reasoning algorithms to detect abnormal access to data (e.g., databases or information retrieval systems) or abnormal email exchange, honeypots for detecting authorized personnel with malicious intentions and activity-based verification (e.g., recognition of keystroke dynamics) and user activity monitoring for detecting abnormal data access.
- ***Designated DLP systems***:
  - Designated systems detect and prevent unauthorized attempts to copy or send sensitive data, intentionally or unintentionally, mainly by personnel who are authorized to access the sensitive information. In order to classify certain information as sensitive, these use mechanisms, such as exact data matching, structured data fingerprinting, statistical methods, rule and regular expression matching, published lexicons, conceptual definitions, keywords and contextual information such as the source of the data.
- https://en.wikipedia.org/wiki/Data_loss_prevention_software#Categories

Data Loss Prevention

Some examples:

- An email system can look for attachments that include sensitive markings
  - In government, this could include classified markings like SECRET, or FOR OFFICIAL USE ONLY
  - In industry, this could include PROPRIETARY, or TRADE SECRET, or HIPAA
- A DLP system could also look for data with a potentially sensitive format
  - This could include a 9-digit number (SSN) – especially if it has the format XXX-XX-XXXX
  - Tax documents such as a W-2
  - Credit card information – 16-digit number, with another number formatted XX/XX (expiration date)
- This can result in false-positives or legitimate operations being blocked
  - For example, communication with the Social Security Agency (SSA) should allow the transmission of an SSN
  - An employee may not be able to email their W-2 to themselves or their accountant
  - You can't have a 9-digit meeting ID for an online meeting with SSA employees
  - There may be problems emailing a person's medical record to them

Data Loss Prevention

DLP systems have to deal with structured and unstructured data

- Structured data resides in fixed fields within a file such as a spreadsheet
- Unstructured data refers to free-form text or media as in text documents, PDF files and video. (Requires natural language processing.)
- It is estimated that 80% of all data is unstructured and 20% structured.
- There are two approaches to determine categories of information
  - Content analysis is focused on structured data
  - Contextual analysis – which looks at the place of origin or the application or system that generated the data – is focused on unstructured data

Methods for describing sensitive content are abundant and can be divided into two categories: precise and imprecise methods.

- Precise methods: Involve Content Registration and trigger almost zero false positive incidents (but is difficult and time-intensive)
- All other methods are imprecise and can include: keywords, lexicons, regular expressions, extended regular expressions, meta data tags, Bayesian analysis and statistical analysis techniques such as Machine Learning, etc.

The strength of the analysis engine directly relates to its accuracy

- It is important for a DLP system to be accurate to lower/avoid false positives and negatives
- Accuracy can depend on many variables, some of which may be situational or technological
- Testing for accuracy is recommended to ensure a solution has virtually zero false positives/negatives
- High false positive rates cause the system to be ignored – including legitimate alerts

Isolation, Separation, and Encapsulation

The Saltzer and Schroeder design principle that is most closely related to ***Isolation, Separation, and Encapsulation*** is ***Separation of Privilege***

To stay with Matt Bishop, from *Computer Security: Art and Science* defines ***Separation of Privilege*** as:

- *The principle of separation of privilege states that a system should not grant permission based upon a single condition.*

Keeping our format, we see goes on to state:

- *This principle is restrictive because it limits access to system entities.*
- *This principle is equivalent to the separation of duty principle discussed in Section 6.1 [of Computer Security]. Company checks for over $75,000 must be signed by two officers of the company. If either does not sign, the check is not valid. The two conditions are the signatures of both officers.*
- *Similarly, systems and programs granting access to resources should do so when more than one condition is met. This provides a fine grained control over the resource, and additional assurance that the access is authorized.*

Separation of Privilege

An example from Matt Bishop:

- *On Berkeley-based versions of the UNIX operating system, users are not allowed to change from their account to the root account unless two conditions are met. The first is that the user knows the root password. The second is that the user is in the wheel group (the group with GID 0). Meeting either condition is not sufficient to acquire root access. Meeting both conditions is required.*

US-CERT on Separation of Privilege

As with Economy of Mechanism and Least Common Mechanism, the US-CERT has an entire page dedicated to Separation of Privilege:

- https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/least-common-mechanism

It states:

- *A system should ensure that multiple conditions are met before granting permissions to an object. Checking access on only one condition may not be adequate for strong security. If an attacker is able to obtain one privilege but not a second, he or she may not be able to launch a successful attack. If a software system largely consists of one component, the idea of having multiple checks to access different components cannot be implemented. Compartmentalizing software into separate components that require multiple checks for access can inhibit an attack or potentially prevent an attacker from taking over an entire system.*

Even more by Saltzer and Schroder:

- *Separation of privilege: Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key. The relevance of this observation to computer systems was pointed out by R. Needham in 1973. The reason is that, once the mechanism is locked, the two keys can be physically separated and distinct programs, organizations, or individuals made responsible for them. From then on, no single accident, deception, or breach of trust is sufficient to compromise the protected information. This principle is often used in bank safe-deposit boxes. It is also at work in the defense system that fires a nuclear weapon only if two different people both give the correct command. In a computer system, separated keys apply to any situation in which two or more conditions must be met before access should be permitted. For example, systems providing user-extendible protected data types usually depend on separation of privilege for their implementation.*

Citations:

- Bishop, Matt. Computer Security: Art and Science. Boston, MA: Addison-Wesley, 2003
- Saltzer, Jerome H. & Schroeder, Michael D. "The Protection of Information in Computer Systems," 1278-1308. *Proceedings of the IEEE 63,* 9 (September 1975).

US-CERT on Separation of Privilege

US-CERT also states:

- *Separation of privilege is defined differently by Howard and LeBlanc [Howard 02].*
  An issue related to using least privilege is support for separation of privilege. This means removing high privilege operations to another process and running that process with the higher privileges required to perform its tasks. Day-to-day interfaces are executed in a lower privileged process.

Citations:

- Bishop, Matt. Computer Security: Art and Science. Boston, MA: Addison-Wesley, 2003.
- Howard, Michael & LeBlanc, David. Writing Secure Code, Second Edition. Redmond, WA: Microsoft Press, 2002.
- Saltzer, Jerome H. & Schroeder, Michael D. "The Protection of Information in Computer Systems," 1278-1308. *Proceedings of the IEEE 63,* 9 (September 1975).

More Definitions

From Todd Merritt:

Separation Privilege Design Principle

- *The separation privilege design principle requires that all resource approved resource access attempts be granted based on more than a single condition. For example a user should be validated for active status and has access to the specific resource.*
- https://dzone.com/articles/9-software-security-design

From Professor Hossein Saiedian's class given at KU:

- *Separation of privilege: multiple privileges should be needed to achieve access (or complete a task)*
- https://people.eecs.ku.edu/~saiedian/710/Lectures/ch01.ppt

Separation of Duties

A related concept, this ensures a single person can't perform malicious activities

- For example, ordering and receiving should be handled by two separate people
  - This prevents someone from placing fake orders, paying themselves, and claiming they were received
  - This helps reduce the insider threat, and helps reduce fraud

From OWASP:

- *The rule of thumb for separation of duties is that the entity that approves an action, the entity that carries out an action, and the entity that monitors that action must be separate. Separation of duties is a prevalent control in both the accounting and IT communities. The goal is to eliminate the possibility of a single user from carrying out and concealing a prohibited action. By separating the authorization, implementation and monitoring roles, fraud can only be successfully carried out through the collusion of multiple parties. Support for segregation of duties should be found in both application features (e.g., role-based access), and should be built into the System Development Lifecycle of the application (e.g., restricting developer access to production libraries). Certain roles have different levels of trust than normal users. In particular, Administrators are different from normal users. In general, administrators should not be users of the application.*
- https://wiki.owasp.org/index.php/Separation_of_duties

Self-Reliant Trustworthiness

Essentially, this means a system should rely on itself – and its own resources – to determine trustworthiness (or at least minimize its reliance on other systems)

For example, an enterprise workstation relies on a Domain Controller to verify credentials

- However, there is the chance that the system will be offline
- A user might need to log into a laptop before they can establish a VPN to the organization
- The solution is to cache the necessary information to verify a user in case the connection is down

When Facebook had a DNS outage, employees were no longer able to access conference rooms

- The conference room relied on access to the physical access control system (PACS) to determine access rights to the room and unlock the door
- Access to the PACS was reliant on DNS working properly
- The chain of reliance meant the physical door required DNS to work properly
  - This is also a good example of the problems with complexity
- https://www.businessinsider.com/facebook-employees-no-access-conference-rooms-because-of-outage-2021-10

Isolation and Encapsulation

From Professor Hossein Saiedian's class given at KU:

- Isolation:
  - Public access should be isolated from critical resources (no connection between public and critical information)
  - Users files should be isolated from one another (except when desired)
  - Security mechanism should be isolated (i.e., preventing access to those mechanisms)"
- https://people.eecs.ku.edu/~saiedian/710/Lectures/ch01.ppt

Encapsulation for security purposes is related to the concept in object-oriented programming, but it is different

- Encapsulation for security means ensuring that all execution paths for code go through the appropriate security checks
- Return-oriented programming (ROP) is an example of an encapsulation failure
  - ROP is an advanced attack technique that was published in 2007 by Hovav Shacham
  - Read more on Wikipedia: https://en.wikipedia.org/wiki/Return-oriented_programming

Summary

The design of the system should be small, simple, and straightforward

Reducing the number of mechanisms necessary helps with verifying the security of a computer system

Such a system can be carefully analyzed, exhaustively tested, perhaps verified, and relied on

Users should not share system mechanisms except when absolutely necessary

Avoid having multiple subjects sharing mechanisms to grant access to a resource

A covert channel is a type of computer security attack that creates a capability to transfer information objects between processes that are not supposed to be allowed to communicate by the computer security policy