# ENPM 809W
# Introduction to Secure Software Engineering

**Gananand Kini**

## Lecture 2

## Software Engineering & Security Principles


MITRE | SOLVING PROBLEMS FOR A SAFER WORLD™

# Outline

- Software Design and Engineering Principles
- Security Principles
- Security Risk Analysis & Threat Modeling

# Software Design and Engineering

# Design Problems

- **Wicked Problem – A problem that can be clearly defined only by solving it, or solving a part of it. (Horst Rittel and Melvin Webber, 1973)**
- **You essentially solve the problem first to define the problem, then solve it again to find a solution that works.**
- **Design is a "Wicked" problem.**
- **There are some guidelines for design…**


Sources:
1. XKCD: Software Development. https://xkcd.com/2021/. Retrieved July 10, 2021.

# Software Design (from Software Engineering Design by Carlos Otero 2012)

- **Abstraction: Deals with creation of conceptual entities that facilitates solving the problem by focusing on the *essential characteristics* of the entities themselves (procedural and data abstraction). Required for good modularization (Liskov and Guttag 2010).**
- **Modularization: *Decomposition* of the system until fine-grained components are created.**
- **Encapsulation: Principles that deals with providing access to the services of the conceptual entities (modules, components etc.) by *exposing* only essential information and *hiding* details on how those services are carried out.**
- **Coupling: The manner and degree of *interdependence* between software modules.**

# Software Design (from Software Engineering Design by Carlos Otero 2012) contd…

- **Cohesion: The manner and degree to which the tasks performed by a single software module are *related* to one another.**
- **Separation of Interface and Implementation: While encapsulation deals with exposing and hiding implementation details, this principle of separation goes further and allows for an interface with the implementation (separate) to be *swapped* for modified or new behavior.**
- **Completeness: Measures how well design units provide *required* services to achieve their intent.**
- **Sufficiency: Measures how well design units provide services that are *sufficient* to achieve their intent.**

# How do these design principles impact security?


- Abstraction and modularization help in being able to determine which of the modules are security relevant. If you don't need a module, then it is redundant and should be removed.
- For example, having a module to deal with users and their profile management can allow for security related functions for a user to be implemented easily.
- Encapsulation and Interface-Implementation Separation can be used to achieve security goals for modules as well by checking security related functions as part of the design. This design principle also helps in reducing the number of *surfaces* that need to be checked or secured by separating the implementation from the interface.
- For example, exposing administrative functionality for a web based application to authenticated and authorized users only where the authentication and authorization mechanisms are implemented independently of the Login.
- Reducing coupling is a general software design goal and helps in securing modules because it also reduces the cascade of failures through the system.
- Example, NPM repository down: March 2016, a disgruntled developer unpublished a very popular package called `left-pad` due to a naming dispute that many other packages had as a dependency causing widespread disruption.

Sources:

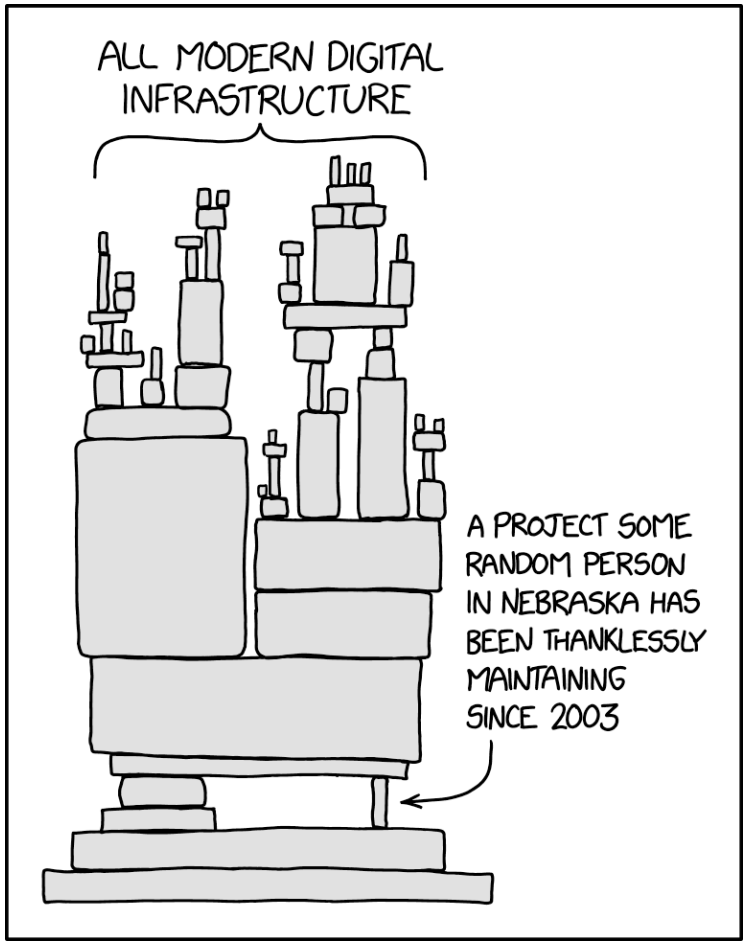
1. https://en.wikipedia.org/wiki/Npm_(software)#Notable_breakages

MITRE

# How do these design principles impact security?

- Data abstraction is a key principle that has implications on privacy and securing data flows within an application.
- The sufficiency principle similarly has implications on minimizing the surfaces that need to be secured.

# XKCD: Dependency




Sources:
1. XKCD: Dependency. https://xkcd.com/2347/. Retrieved July 10, 2021.

# NIST SP 800-160 Volume 1 Appendix F (Secure Design Principles) Topics

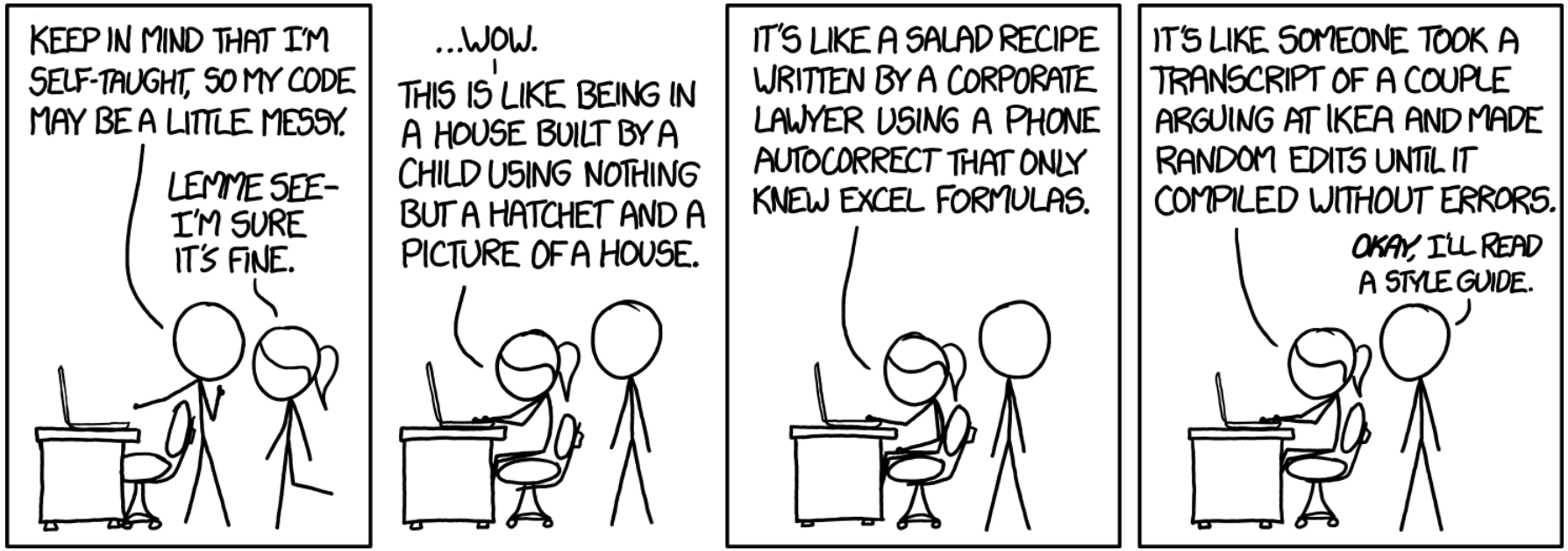
- Security Architecture and Design
  - Clear Abstractions
  - Least Common Mechanism
  - Modularity and Layering
  - Partially Ordered Dependencies
  - Efficiently Mediated Access
  - Minimized Sharing
  - Reduced Complexity
  - Secure Evolvability
  - Trusted Components
  - Hierarchical Trust
  - Inverse Modification Threshold
  - Hierarchical Protection
  - Minimized Security Elements
  - Least Privilege
  - Predicate Permission
  - Self-Reliant Trustworthiness
  - Secure Distributed Composition
  - Trusted Communication Channels
- Security Capability and Intrinsic Behaviors
  - Continuous Protection
  - Secure Metadata Management
  - Self-Analysis
  - Accountability and Traceability
  - Secure Defaults
  - Secure Failure and Recovery
  - Economic Security
  - Performance Security
  - Human Factored Security
  - Acceptable Security
- Life Cycle Security
  - Repeatable and Documented Procedures
  - Procedural Rigor
  - Secure System Modification
  - Sufficient Documentation

Sources:

1. Ron Ross, Michael McEvilley, Janet Corrier Oren. Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems. Vol 1. https://doi.org/10.6028/NIST.SP.800-160v1. pp 204-219.

# A note about software quality

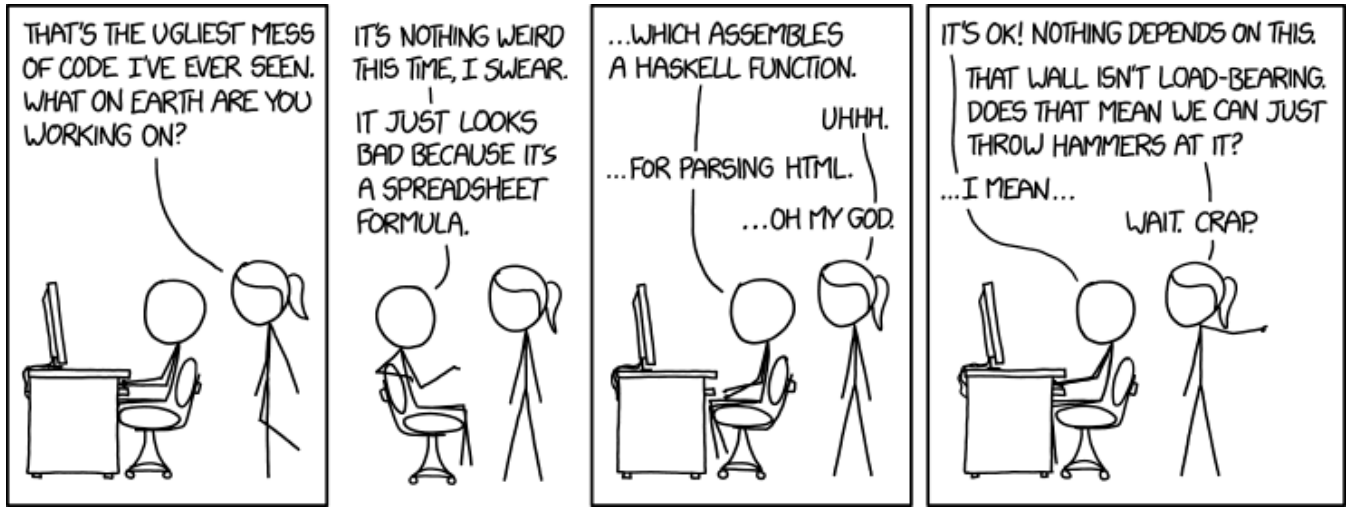
- There are several software quality metrics that can help assess the quality of the software and its design.
- High software quality can also lead to more secure software since it helps with the maintenance of software systems during its lifecycle.




Source:
1. XKCD: Software Quality. https://xkcd.com/1513/. Retrieved July 10, 2021.

# Why is software quality important?

- **Even if no other components depend on it, it is more than likely:**
  - That someone will re-use the code from the component
  - That someone will make it dependent on another software module
  - That an attacker will gain access to it
- **Also finding bugs becomes easier when software quality is maintained!**




Source:
1. XKCD: Bad Code. https://xkcd.com/1926/. Retrieved July 10, 2021.

# Security Principles

# Weakness (for this class)

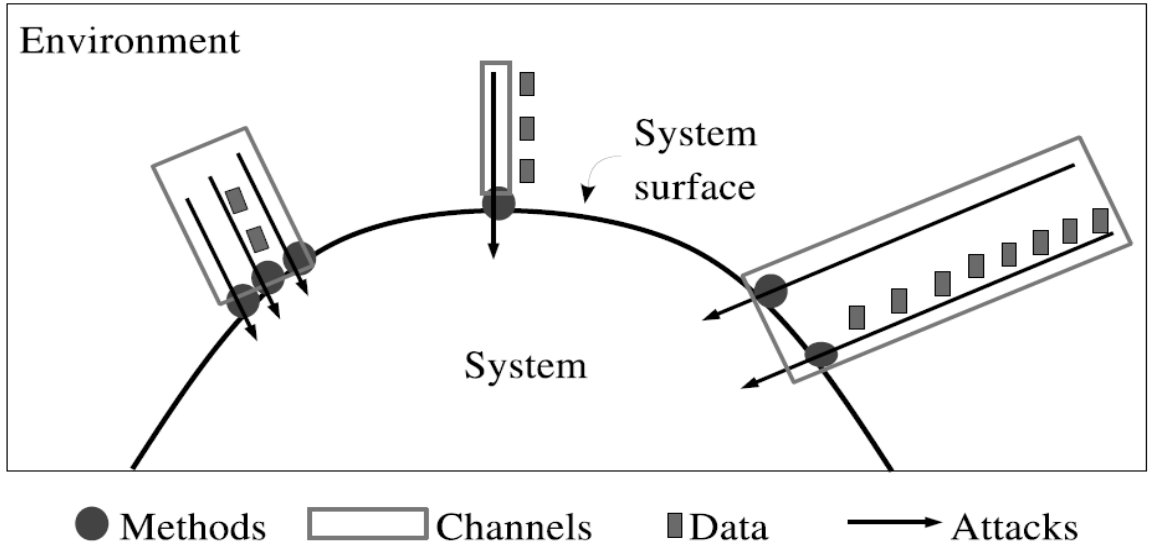
- Mistakes or flaws (intentional or unintentional) in the software system that can lead to serious system level impacts.
- You can use the CWE™ language to describe such flaws.
- Different from vulnerabilities, which involve using a weakness or set of weaknesses to achieve a desired effect or impact on the system.

# Attack Surface ( An informal definition)

- We have heard the concept of a “surface” in principles of software system design – an interface to the interaction between modules.
- An attack surface similarly provides attackers an interface to system resources that could be abused to access security vulnerabilities and to exploit the software system to achieve an attacker’s goals.
- For a home, what are some example attack surfaces?

# Attack Surface – Formal Definition

- Attacker can attack using channels (e.g., ports, sockets), invoke methods (e.g., API), and send data items (input strings or indirectly via persistent or stored data)
- A system's attack surface – Subset of the system's resources (channels, methods, and data) that can be used in attacks on the system.
- More attack surfaces likely means it is easier to exploit and cause more damage.



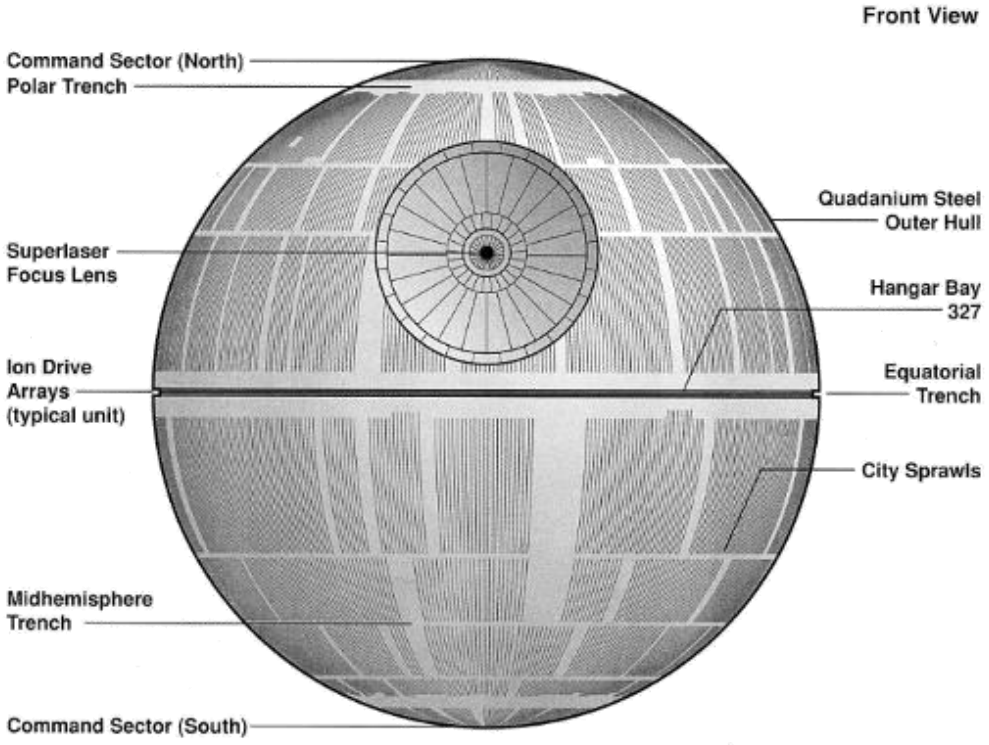

Source: *Attack Surface Metric*, Pratyusa K. Manadhata, CMU-CS-08-152, November 2008

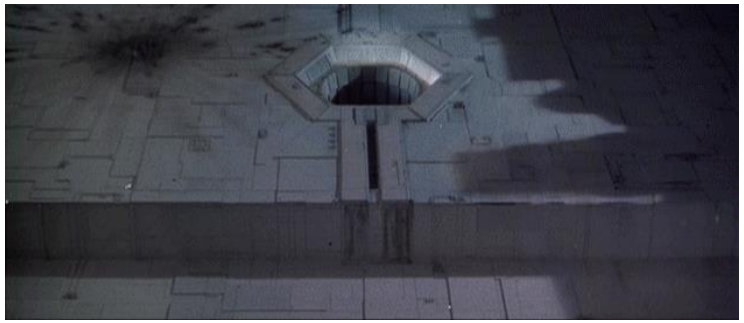
# What is an Attack Vector?

- **The set of inputs, actions and effects on the software system that when provided via the attack surface cause the exploitation of the system achieving the attacker/user's goal.**

- **Typically involves:**
  - Access to the software system
  - Ability to manipulate input
  - Ability and access to the configuration of the software environment

# Illustration



The reactor system at the core has a **weakness** that when it is hit with a blast, it causes a chain reaction that can be catastrophic. Intentionally hidden by Galen Erso, a scientist coerced to work for the Galactic Empire.

Exhaust port on the Death Star designed for exhaust (output) is an example of an **attack surface**.

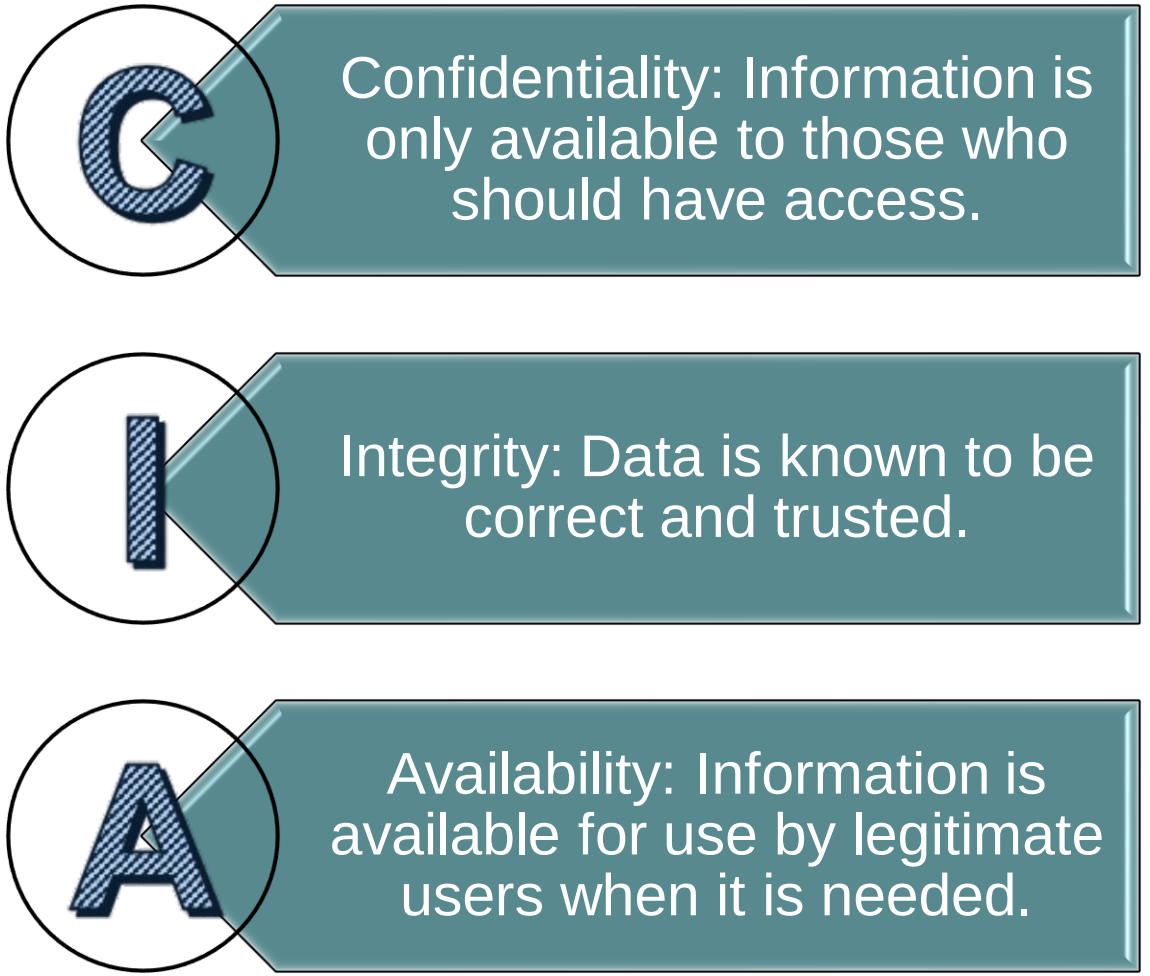
The Death Star has a **vulnerability** where anything that can cause a chain reaction in the reactor core can lead to a catastrophic loss of the system.

The **attack vector** is the two proton torpedoes used to access the exhaust port, reach the reactor core and initiate the chain reaction.


1. Johnson, S. (1995). In *Star Wars technical journal*. Boxtree.
2. David W, & Django Reinhardt. (1960, August 1). *Is there a canon explanation for how Proton Torpedoes were able to turn 90 degrees at the end of Star Wars: ANH*. Science Fiction & Fantasy Stack Exchange. https://scifi.stackexchange.com/questions/8543/is-there-a-canon-explanation-for-how-proton-torpedoes-were-able-to-turn-90-degre.

# Typical Security Objectives

- **Other security objectives:**
  - Non-repudiation – Transactional security property where sender of a message is provided with proof of delivery and recipient of the message is provided with proof of sender's identity
  - Privacy – Property where all disclosure of information is provided only by authorized consent.
  - Audit/Accountability/Logging – Transactional security property where all security relevant actions and events are recorded and can be examined by authorized parties.
  - Id-entity – Typically digital identity, a security property that allows unique identification of individual entities allowing for authentication, authorization and access to services.



# A set of security principles

- **A set of guiding principles for securing software systems need to be followed. The real "secret-sauce" behind most of the secure designs of software being followed in modern enterprises.**

- **The following security principles already have existed since the 1970s and are from the following sources:**
  - McGraw, Gary & Viega, John. "Keep It Simple." *Software Development.* CMP Media LLC, May, 2003.
  - Viega, John & McGraw, Gary. *Building Secure Software: How to Avoid Security Problems the Right Way*. Boston, MA: Addison-Wesley, 2002.
  - Saltzer, Jerome H. & Schroeder, Michael D. "The Protection of Information in Computer Systems," 1278-1308. Proceedings of the IEEE 63, 9 (September 1975).
  - Howard, Michael & LeBlanc, David. *Writing Secure Code*. 2nd. Redmond, WA: Microsoft Press, 2002.

# Trustworthniess vs. trust

- **“Trustworthiness implies that something is worthy of being trusted”**
- **“Trust merely implies that you trust something, whether it is trustworthy or not”**
  - Trust is a *decision*
  - You *should* only trust things that have *adequate* evidence of being trustworthy

Source: Definitions from “Principled Assuredly Trustworthy Composable Architectures“ by Peter Neumann, 2004

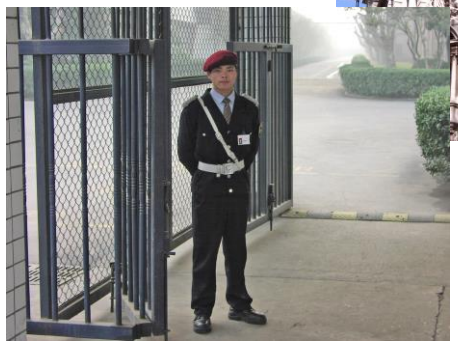
# Security Principle – Defense In Depth

- Do not rely on a single security method.
- Anticipate failures and layer basic security practices.
- Example: A Bank that manages money.

Bank Vault

Bank Teller

Bank Building



Security Guard

Title: A private security officer at a Chinese factory in February 2004.
Creator: Robbie Sproule
Source: Flickr
License: CC-By-2.0

Title: Lloyds Bank Building
Creator: Pit-Yacker
Source: Wikimedia Commons
License: CC-By-2.5

Title: A teller in a branch of Bank Muamalat
Creator: Melwinsy
Source: Wikipedia
License: CC-By-SA-4.0

Title: Bank Vaults under Hotels in Toronto, Ontario
Creator: Jason Baker
Source: Flickr
License: CC-By-2.0

# Security Principle – Keep Security Simple

- Keep security or protection mechanisms simple.
- Makes it easy to understand, implement and analyze security
- Can also make it easy to use.

# Security Principle – Attack Surface Reduction

- **Actively address and reduce any surfaces exposed to unauthenticated or unauthorized actors that allow for the system to be compromised against its security goals.**
- **This can be done in a number of ways:**
  - From software design principles, the ideas of encapsulation or hiding functionality from entities (code, users, etc.) that don't require it, separation of interface and implementation can also be used to control which entities have access to what functionality etc.
  - In software systems, easiest way is to follow:
    - KISS principle: 'keep it simple stupid'. Keeping the system simple at design time tends to make it easy for entities to use the system and prevent misuse. It also helps in understanding and analyzing the system for attack surfaces and vulnerabilities.
    - Remove functionality that is not needed. The attack surface is essentially going to be all the code, functionality or data that is accessible by entities. Removing or reducing those can go a long way…

# Security Principle – Establish Secure Defaults

- The default configuration or state that ships with the software system should already include the most secure configuration. That means out of the box the software system should stop insecure actions.

- Do not rely on an entity to configure the software system in order to enable security features or to make it run securely.

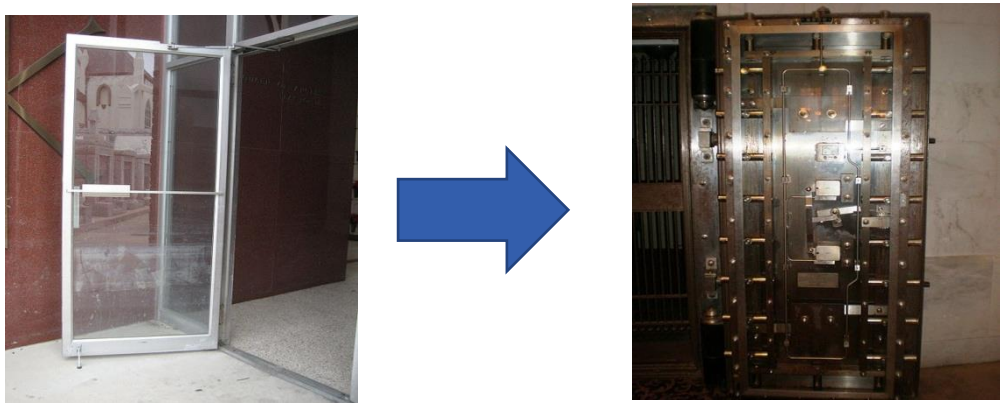
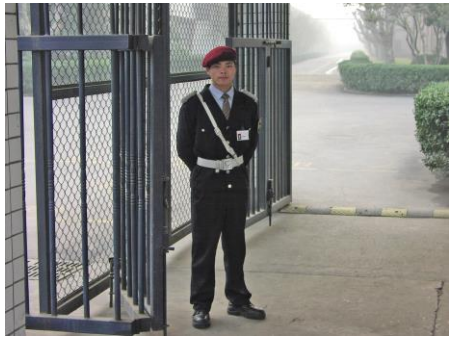


Image of Vault Door: © BrokenSphere / Wikimedia Commons. Retrieved September 20, 2011 from https://secure.wikimedia.org/wikipedia/commons/wiki/File:SF_City_Hall_South_Light_Court_vault_1st_door.JPG
Image of Glass Door: Under Creative Commons Attribution-Share Alike 3.0 license – taken by Infrogmation. Retrieved September 20, 2011, from https://secure.wikimedia.org/wikipedia/commons/wiki/File:StRochDec07GlassDoorFloodlines.jpg

# Security Principle – Principle of Least Privilege

- **Not everyone needs *access* or *privileges* to do everything.**
- **The entities (users, or programs) that use the software system should only be allowed to operate with the least privileges possible.**
- **This limits the damage from an accident, error or attack.**



Access?

Privilege?

Image of Gold and Coins: Licensed under Creative Commons Attribution 2.0 Generic License – Taken by Bullion Vault on June 3, 2009 from https://www.flickr.com/photos/bullionvault/3591732069.
Title: A private security officer at a Chinese factory in February 2004.
Creator: Robbie Sproule
Source: Flickr
License: CC-By-2.0

# Security Principle – Fail Securely

- **Security controls in an application should assume the application is under attack by default**
- **Apple SSLVerifySignedServerKeyExchange checks validity of SSL certificate:**
  - The second goto fail is not part of the `if` block:

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
  ... other checks ...
  fail:
    ... buffer frees (cleanups) ...
    return err;
```

- **Several issues here. But, one of them is the error code default was 0 which meant no error. This allowed attacker to skip other checks and return error of 0 as long as first statement returned no error. Here, failure cases should be checked explicitly and independently rather than stacking them one after another.**

Source:

1. https://us-cert.cisa.gov/bsi/articles/knowledge/principles/failing-securely
2. https://dwheeler.com/essays/apple-goto-fail.html

# Security Principle – Complete Mediation ("Non-by-passable")

- **Following from the earlier example, this principle involves making sure:**
  - To find **all** channels
  - Check **all** inputs from untrusted sources
  - Check as soon as possible and fail early
- **If the system has a client-server model, this means ensuring:**
  - Security checks are done on the server-side (client is not necessarily trusted, server may be trusted but not necessary)
  - Check trustworthiness of environment (client or server) to ensure checks can also return trustworthy results
  - Client-side checking can improve user-response and lower server load, however do client-side checks ***in addition to*** server-side checks.

# Security Principle – Don’t trust service interfaces

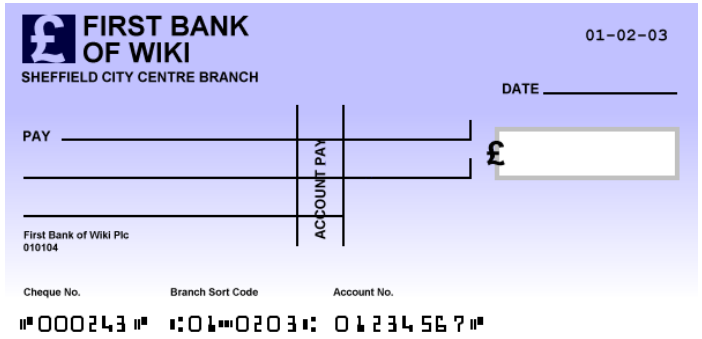
- **This principle means the software system should not trust any of the interfaces it *interacts* with especially with regard to:**
  - Its behavior
  - Its promises
- **The mere existence of an interface is not a contract that it will behave correctly.**
- **Example interfaces:**
  - Third-party services and libraries
  - Databases
  - File systems
  - Memory

Source:
1. https://movies.stackexchange.com/questions/42527/is-there-any-real-world-science-behind-r2-d2-s-computer-interfacing-arm. Retrieved July 10, 2021.

# Security Principle – Separation of Duties/Privilege

- This principle involves splitting the roles, duties and privileges that go along with those roles and duties into separate parts of the software system.
- Example: You would not want a bank to allow the same customer to change the address of an account and authorize a check against an account.


*Change of Address*




*Authorize a Check*

Check Image: Licensed under Creative Commons Attribution – Share Alike 3.0 Unported– Created by Sergio Ortega, modified by Trojan. Retrieved September 20, 2011, from https://secure.wikimedia.org/wikipedia/commons/wiki/File:BritishChequeEmpty.PNG

# Security Principle – Avoid Security Through Obscurity

- **This principle states:**
  - Avoid implementing obscure security mechanisms to “add” security.
  - Adversaries and attackers likely already have access to the software system, source code and perhaps even the data for reverse engineering given enough resources.
  - Relying on such mechanisms is very dangerous!
- **Assume the software system is already compromised and build security *in*.**
- **Example: The gate here really does not really add any security nor does it add deterrence.**

Image Source: https://www.syslog.com/~jwilson/pics-i-like/kurios119.jpg referenced by Bruce Schneier https://www.schneier.com/blog/archives/2005/02/the_weakest_lin.html on Feb 1, 2005.

# Security Principle – Avoid non-atomic security operations

- **This principles talks to how the software system should treat and implement security actions and operations atomically.**

- **Example:**
  - Fake check scheme of the past.
  - Both check verification and account balance update should be part of the same atomic transaction.
  - Check needs to be verified and then the deposit updated as part of the same transaction.

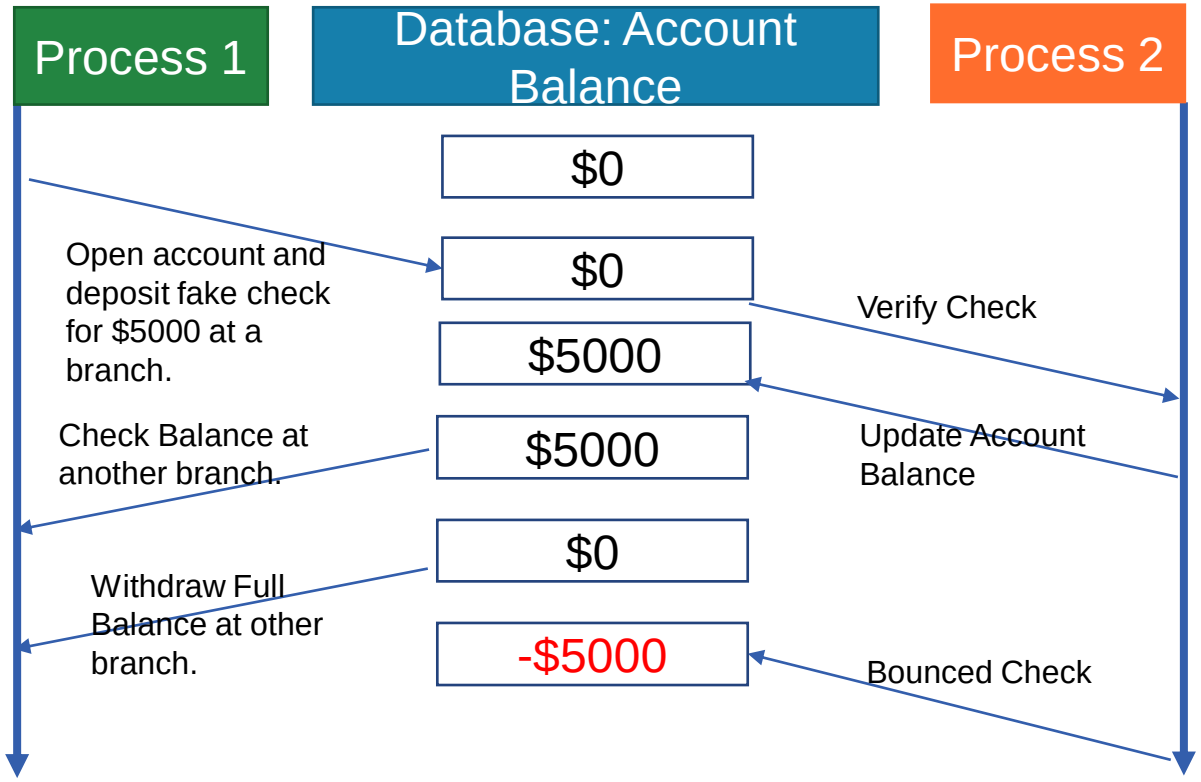
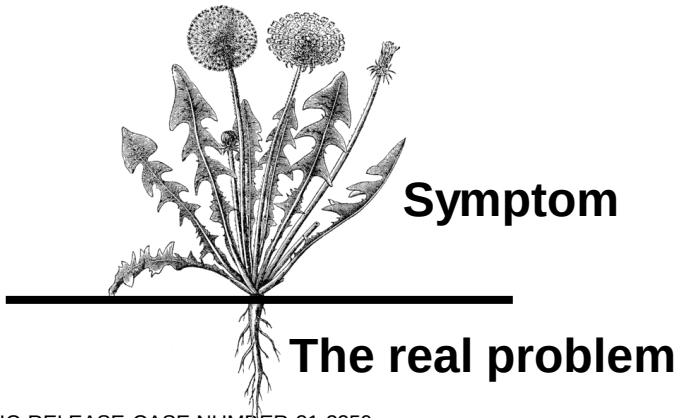



Image Sources:
https://commons.wikimedia.org/wiki/File:Criminal_Silhouette_L.svg Dingbat Fonts by Unknown. Licensed under Creative Commons 1.0 Universal Public Domain.
https://commons.wikimedia.org/wiki/File:Icon_bank_sentral.png Own Work by Bang Sen. Licensed under Creative Commons Share Alike 4.0 International.

# Security Principle – Fix security issues correctly

- Security issues need to be treated at the root cause of the issue rather than just addressing symptoms.
- The first gut response to an attack is to fix the issues responsible directly for the attack. You may only be addressing the symptoms of the problem.
- However, this principle states that the software system and processes should be analyzed for a deeper analysis on what allowed the issue to happen in the first place.
- As a software engineer you need to be able to fully understand the problem before you can design and implement a fix.



# Security Principle – Least Common 'Security' Mechanism

- This principle is actually a design principle where the system should be implemented in such a way as to reduce and minimize the use of shared mechanisms. For example, the use of /tmp or /var/tmp directories to store temporary files for users.
- This applies to security mechanisms as well.
- Shared objects provide potentially dangerous channels for information flow and unintended interactions.
- What is wrong with this code?

```
public class ApplicationUser : IdentityUser<int, ApplicationUserLogin,
                            ApplicationUserRole, ApplicationUserClaim>, User<int>
{
    public string Name { get; set; }
    public string Surname { get; set; }
    //code omitted for brevity
}

public class Student: ApplicationUser
{
     public int? Number { get; set; }

       [HttpPost]
       [ValidateAntiForgeryToken]
       public JsonResult Update([Bind(Exclude = null)] StudentViewModel model)
       {
           if (ModelState.IsValid)
           {
               ApplicationUser user = UserManager.FindById(model.Id);

               user = new Student
               {
                   Name = model.Name,
                   Surname = model.Surname,
                   UserName = model.UserName,
                   Email = model.Email,
                   PhoneNumber = model.PhoneNumber,
                   Number = model.Number, //custom property
                   PasswordHash = checkUser.PasswordHash
               };

               UserManager.Update(user);
           }
       }
}
```

Source: https://stackoverflow.com/questions/39304464/how-to-save-new-record-with-hashed-password-in-my-custom-table-instead-of-aspnet

# Security Principle – Limit resource dependencies

- **This principle involves limiting the amount of resources depended upon by the software system wherever possible.**
- **Fail2Ban – Limits excessive unauthorized login attempts over the network for an application based on logs and bans them when the limits are exceeded. Can be used for multiple applications including SSH.**
- **Google, 2020 – Denial Of Service by UDP amplification attack:**
- "In 2017, our Security Reliability Engineering team measured a record-breaking UDP amplification attack sourced out of several Chinese ISPs (ASNs 4134, 4837, 58453, and 9394), which remains the largest bandwidth attack of which we are aware."
- "The attacker used several networks to spoof 167 Mpps (millions of packets per second) to 180,000 exposed CLDAP, DNS, and SMTP servers, which would then send large responses to us. **This demonstrates the volumes a well-resourced attacker can achieve.** This was four times larger than the record-breaking 623 Gbps attack from the Mirai botnet a year earlier." – Damien Menscher, Security Reliability Engineer at Google

Source: https://www.a10networks.com/blog/5-most-famous-ddos-attacks/. Retrieved July 10, 2021.

# Security Principle – Ease of use for the software system

- This principle follows the previously discussed KISS principle.
- However, as applied to security, the software system's human interface must be designed for ease of use so users will *routinely* and *automatically* use the protection mechanisms correctly.
- Making it easy for users to secure their data and the software system, will make the system more secure.
- Mistakes are reduced when security mechanisms in the software system closely match the user's perception of their own protection goals.

Credit: Dr. David A. Wheeler

# Security Principle – Hardening the software system environment


- The software system is designed to run in an environment.
- Building an application securely + deploying it in an insecure environment ≠ Secure Software System
- The software system should also not assume anything about the environment wherever possible.

# The confused deputy problem

- A more privileged program (“a deputy”) that is tricked by a lower privileged program into transitively giving it authority and misusing the authority. This is a type of privilege escalation.

# Time-of-check-Time-of-Use (TOCTOU) problem


- A race condition where a program checks the state of a part of the system (like a security credential) and the use of the results of that check.

- Example: The `setuid` program bug

```
if (access("file", W_OK) != 0) {
    exit(1);
}

fd = open("file", O_WRONLY);
// Actually writing over /etc/passwd
write(fd, buffer, sizeof(buffer));
```

```
//
//
// After the access check
symlink("/etc/passwd", "file");
// Before the open, "file" points to the password database
//
//
```

Source: https://en.wikipedia.org/wiki/Time-of-check_to_time-of-use. Retrieved July 10, 2021.

# Covert/Side Channels

- **Surreptitious ways to send or “leak” data across data boundaries.**

- **This requires some sort of access, coordination and/or sharing.**


Source:
1. Wikipedia: Telefono Meccanico. https://it.wikipedia.org/wiki/Telefono_meccanico#/media/File:Tel%C3%A9fono_de_cordel_(1882).jpg. Retrieved July 10, 2021.
2. “The Eavesdropper” oil on canvas by Hubertus Van Hove. http://www.wikigallery.org/. Retrieved July 10, 2021.

# Impacts from security principle violations

- **Information leakage: Sensitive or Unauthorized information got exposed.**
- **Modification of data: Sensitive data was modified or unauthorized changes made to data.**
- **Denial of Service: Unreliable Execution: The software system became unreliable or went into a degraded mode of operation.**
- **Denial of Service: Resource Consumption: The software system ran out of resources.**
- **Unauthorized code or command execution: Unauthorized code or commands were able to be run or executed by external users.**
- **Escalation of privilege / Assumption of Identity: Malicious actors were able to hijack the identity of a privileged user or escalate their privileges within the software system or its environment.**
- **Bypass Protection Mechanism: Any protection mechanisms were easily bypassed or circumvented.**
- **Hide Activities: Activities performed on the software system or its environment were hidden from auditors or system evaluators or monitors.**
- **Loss of trust: Users lost trust in the system they were using.**

# Security Risk Analysis & Threat Modeling

# Security Risk Analysis

- After designing your software and architecture, you can apply the principles covered so far to do a risk analysis.
- Goal is to uncover any design flaws with a security focus.
- Identify any weak, missing, improper or inadequate security elements in the design/architecture or implementation planned. These kinds of issues can be expensive after implementation begins.
- Try to take the software system's final target running environment into account if possible. This is especially difficult if where the software system will end up is unknown during the design phase.

# What is Threat Modeling?

- **Threat – Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image or reputation). From NIST FIPS 200.**

- **Typically threats come with actors (usually entities with malign intent). Put yourself in their shoes, think like an attacker before implementation begins!**

- **Threat modeling is the activity or set of activities used to assess, identify security objectives and vulnerabilities, and then defining countermeasures to prevent or mitigate the effects of threats to a system. (OWASP)**

Source: https://csrc.nist.gov/glossary/term/threat. Retrieved July 1, 2021.

# Elements in a threat model

- Architecture
- Structure of the software system
- Assets
- Behaviors
- Trust boundaries
- Attack Surface

# Why do threat modeling?

- Build a secure design.
- Discover security issues early on in the lifecycle saving resources.
- Be able to make risk decisions early prioritizing security development.
- Bring security and development teams together helping increase synergy, shared understanding, informed development, maintenance and operating the software system.
- Give an idea of risk that organization/customer is taking on as part of fielding the software system.

# STRIDE approach (Microsoft, 2006)

- **Threat:**
  - Spoofing
  - Tampering
  - Repudiation
  - Information Disclosure
  - Denial of Service
  - Elevation of Privilege
- **Security Property:**
  - Authentication
  - Integrity
  - Non-Repudiation
  - Confidentiality
  - Availability
  - Authorization

Source: Uncover Security Design Flaws Using The STRIDE Approach | Microsoft Docs. Retrieved July 1, 2021.

# STRIDE approach (Microsoft, 2006)

| Approach | Description |
|---|---|
| Spoofing | The deliberate inducement of a user or resource to take incorrect action. Note: Impersonation, Masquerading, Piggybacking, and mimicking are forms of spoofing. |
| Tampering | An intentional unauthorized act resulting in the modification of a system, components of systems, its intended behavior, or data. |
| Repudiation | The rejection of a claim involving an action or condition. Typically defined using non-repudiation or the inability to reject the claim by showing some proof. Example: Package delivery receipt. The receiver of the package can repudiate the claim that the package was delivered, while the package delivery company can show a proof of delivery to show it is not their fault thus making it non-repudiable. However, this property maybe desirable for privacy. A citizen may want to be able to repudiate the claim that they sent a private message under an authoritarian government with no ability to show proof showing that they sent the message. |
| Information Disclosure or Disclosure of Information | Data that is disclosed without authorization. |
| Denial of Service | Can be at the system, process, data store, data flow or resource level. The prevention of authorized access to resources or the delaying of time-critical operations. |
| Elevation of Privilege | The exploitation of a bug or flaw (read unauthorized) that allows for a higher privilege level than what would normally be permitted. |

Sources:
1. NIST Glossary. https://csrc.nist.gov/glossary. Retrieved July 10, 2021.

# STRIDE approach (Microsoft, 2006)

- **You apply each of the threats to your software system design's users, resources, data flows and logic.**
- **For example, with a web based application, you model:**
  - The users of the system
  - The resources in the system
  - The data flows that occur
  - The components and their logic
- **Apply STRIDE to each of the above. Examples:**
  - What would happen if I spoofed a user's identity, what is the impact on the software system? Is there loss of confidential data?, leaves the application in a bad state?, etc.
  - Is it possible for someone to tamper with a user's profile data? What is the impact of tampering with a user's profile within the context of the software system design?
- **Finally, mitigate the threats identified to have an impact in the software system.**

# MITRE ATT&CK™ Framework (2019-21): A Note

- A knowledge base of adversary tactics and techniques based on real world observations.
- Also threat modeling, but at a very high organizational level.
- More applicable to the organization hosting or deploying the software system.
- However, idea is to pool resources towards the risks that the organization needs more protection against.
- See Chapter 4 of https://www.mitre.org/sites/default/files/publications/mitre-getting-started-with-attack-october-2019.pdf.

# Threat Modeling Tools

- Microsoft Threat Modeling Tool (TMT7.0)
- Mozilla SeaSponge

- Use data flow diagrams to generate potential threats.

- Don’t go overboard. Add enough information needed for understanding threat surfaces.

- Start with a high level data flow picture and then you can create sub-diagrams to describe components further.

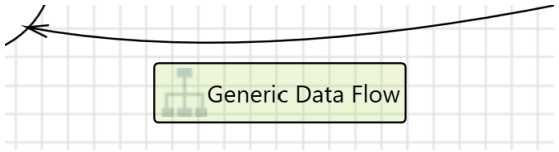
- Describe each part of the data flow diagram in sufficient detail, avoiding generic names, and avoiding too much detail.

Source:

1. https://owasp.org/www-community/Threat_Modeling. Retrieved July 10, 2021.
2. Use recommended tools to create a data-flow diagram - Learn | Microsoft Docs. Retrieved July 10, 2021.

# Data Flow Diagrams

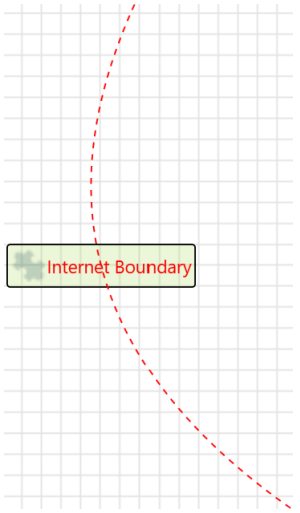

Processes are represented as circles.

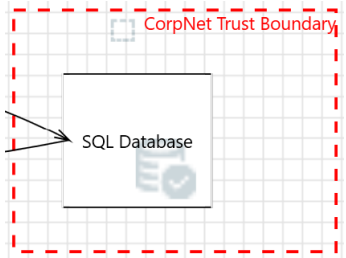



Data flows are represented by an arrow. If data flows are bi-directional, arrows point both ways.


External Entities to the system are represented as squares.


Data sources like files and databases are represented as two horizontal lines.




Trust Boundaries are indicated by these dashed lines.

Source:
1. Microsoft Threat Modeling Tool Example.

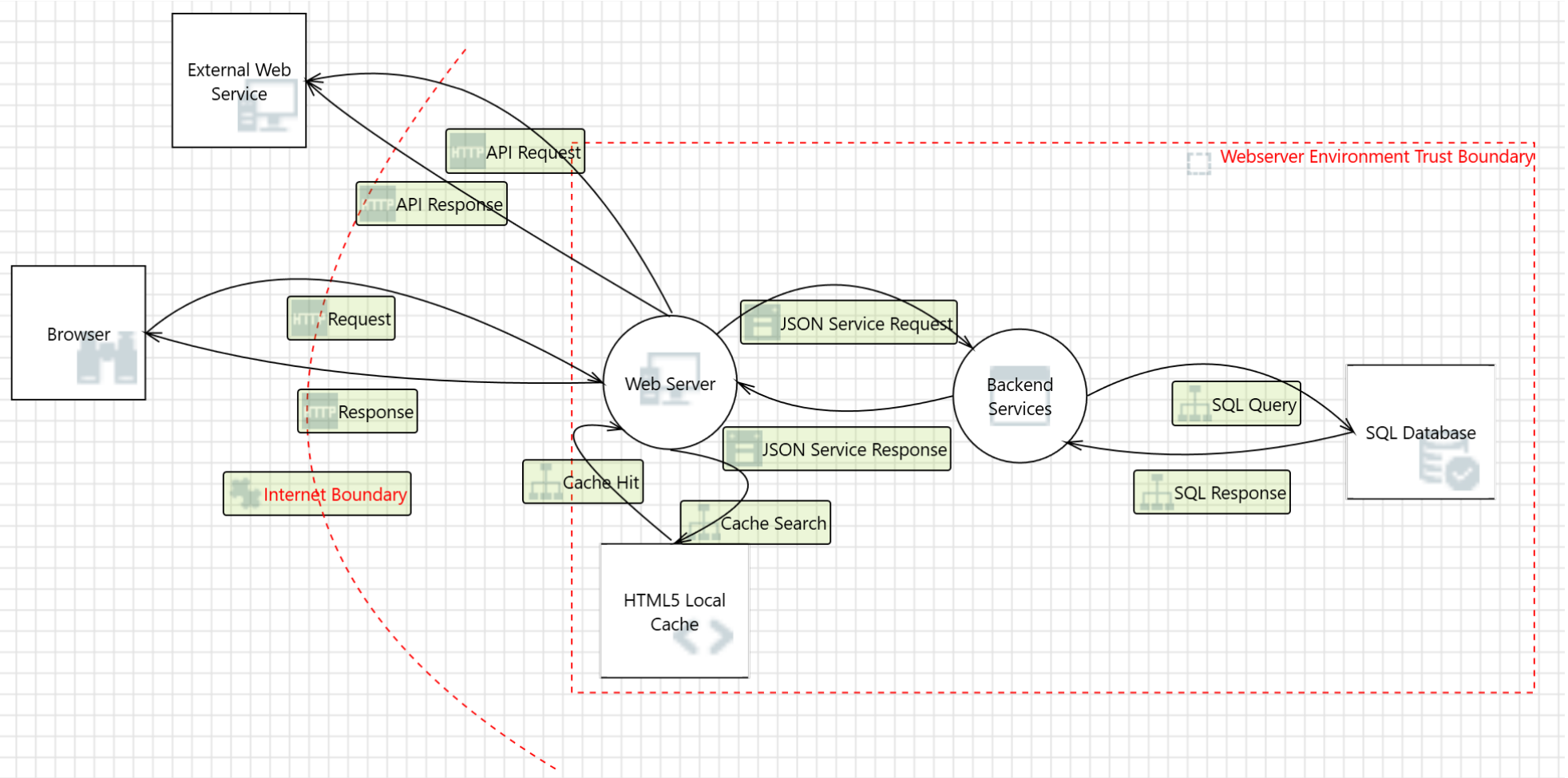
# Trust Boundaries

- A boundary where the trust environment for the data flow changes or shifts.

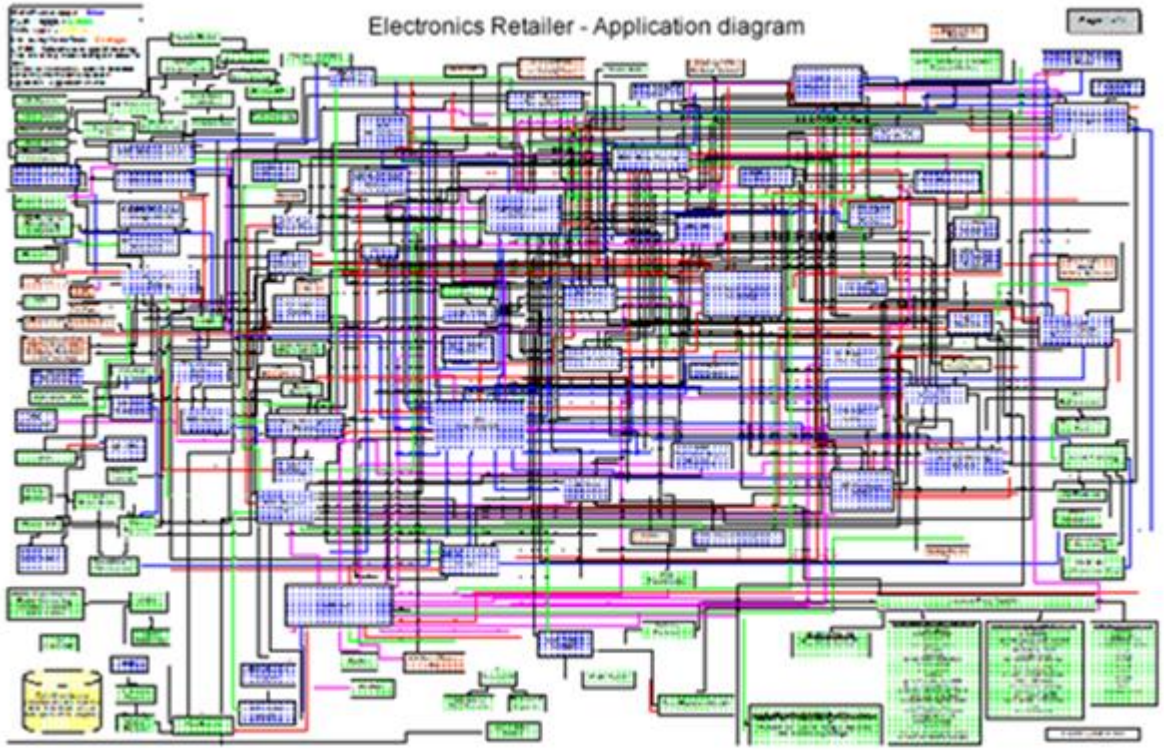
- An example: Once data flows from your computer, over the internet to the webserver, it probably crosses several boundaries: Your local network, the internet service provider network and finally the webserver's network. Even on the webserver network side, there may be a trust boundary separating the web server from 3rd party API based services for example.

- The boundaries can exist between machines, networks, components, processes etc. However, keep in mind when modeling to only include ones that are necessary to explain the security perspective.

# Examples of Data Flow Diagrams



# What Data Flow Diagrams should not look like…





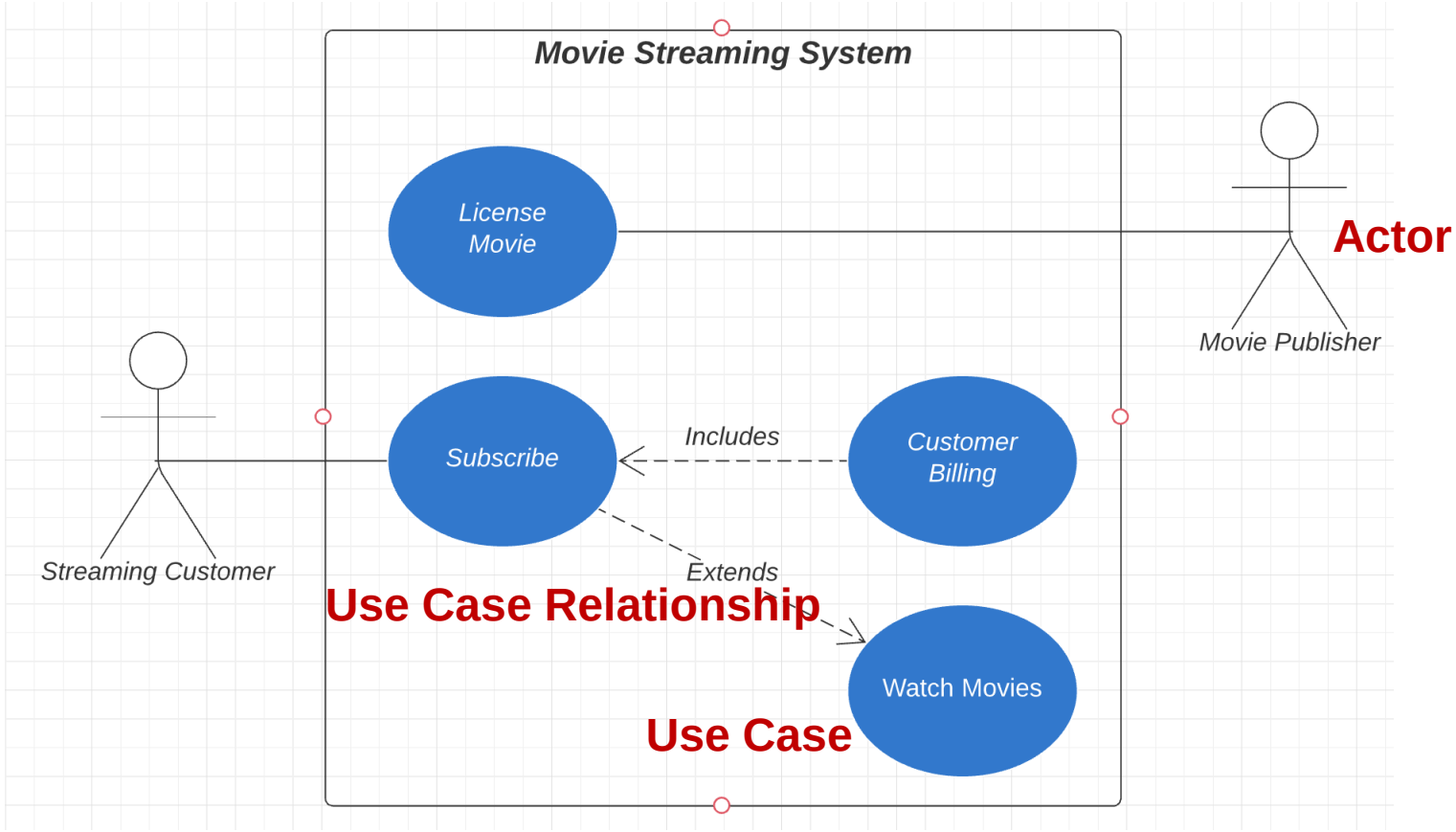
Source:

1. Kadiyala, Anant. How we got to now in enterprise software, and where we are headed – Part 3/6. https://medium.com/@akadiyala/how-we-got-to-now-in-enterprise-software-and-where-we-are-headed-part-3-6-c0ffe01f06b5 . Retrieved July 10,2021.
2. veiledcrit. Bingeclock. https://www.bingeclock.com/meme/s/tidying-up-with-marie-kondo/p/1/. Retrieved July 10, 2021.

# Use Case Diagrams

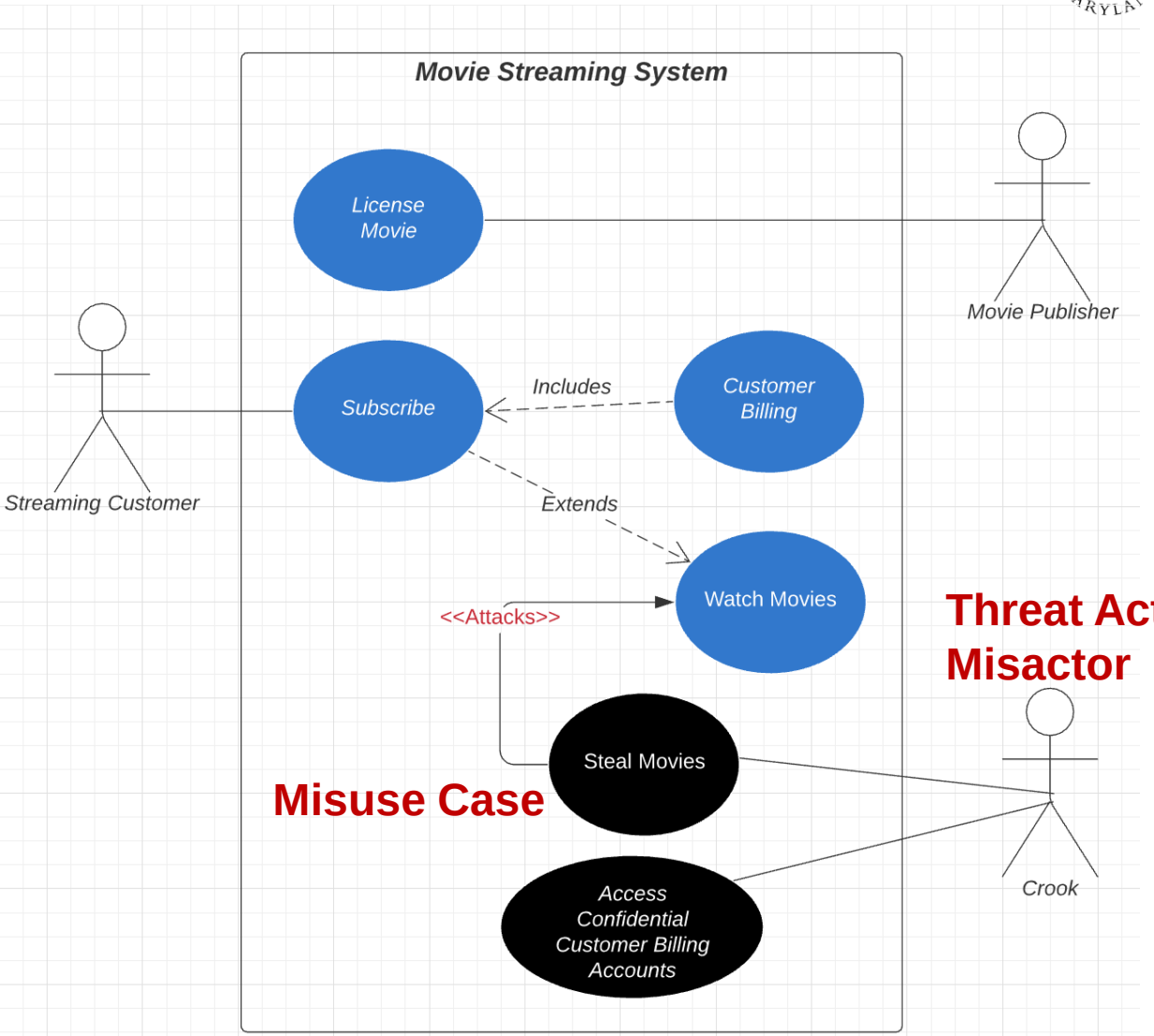
- **Highlights use cases for entities that use the software system.**

- **Includes the System boundary, Actors, Use Cases, and relationships like extend or include.**



# Misuse Cases

- Similar to use cases, however looking at misuse of the software system. Described by Guttorm Sindre and Andreas Opahl in 2001.
- Another threat modeling technique to think of all the ways a software system could be misused.
- Useful to analyze especially during the design phase of the Secure Development Lifecycle.



# Security Analysis Modeling and Testing – What misuse cases should you model and test for?

- **Test for the security objectives (covered earlier):**
  - Confidentiality
  - Integrity
  - Availability
  - Non-Repudiation
  - Identity (Authentication and Authorization)
- **Test with the assumption of STRIDE threats to your system (corollary to the above security objectives):**
  - Spoofing (Authentication)
  - Tampering (Integrity)
  - Repudiation (Non-Repudiation)
  - Information Disclosure (Confidentiality)
  - Denial of Service (Availability)
  - Elevation of Privilege (Authorization)

# How do you perform Security Analysis Testing?

- **Several ways including:**
  - Penetration Testing – Requires an outside party (other than developers) to test the software system usually with permission (perhaps from the organization's security team etc.)
  - Security Static Analysis Tools – Scan for security problems by **not** running the software system
  - Security Dynamic Analysis Tools – Scan for security problems when running the software system
  - Will cover the last two techniques later in the course.

# Next time …

- **Input related Security Bugs**

# Glossary of some security terminology for this course

- **Adversary or Attacker – An entity or actor that attacks or misuses the system, typically with malicious intent.**
- **Weakness – An intentional or unintentional mistake or bug in the software system that leads to a technical impact on the organization, systems, processes or resources.**
- **Vulnerability – A weakness or set of weaknesses that an adversary or user can exercise to achieve an effect on the system.**
- **Exploit – A vulnerability or set of vulnerabilities that allow an adversary or user to achieve their attack or system modification goal.**
- **Attack vector – A method or set of methods used to attack or make unauthorized modifications to a system.**
- **Attack surface – A component or part of the system that introduces attack vectors or exposes the system to exploitation.**
- **Authentication – Generally, the act of proving an assertion. However for identity security, verify the identity of the user, process, or device as a pre-requisite to allowing access to resources in an information system.**
- **Authorization – The right or a permission that is granted to a system entity to access a system resource.**

Source:
1. NIST Glossary. https://csrc.nist.gov/glossary. Retrieved July 10, 2021.

# Glossary of some security terminology for this course

- **OPSEC – Operations Security – A systematic and proven process by which potential adversaries can be denied information about capabilities and intentions by identifying, controlling, and protecting generally unclassified evidence of the planning and execution of sensitive activities. Process involves five activities:**
  - identification of critical information,
  - analysis of threats,
  - analysis of vulnerabilities,
  - assessment of risks,
  - and application of appropriate countermeasures.