FH JOANNEUM
Elektronik und
Computer Engineering

Security and Testing
Architectural Risk Analysis

FH JOANNEUM
Institute of Electronic Engineering
System Test Engineering
Egon Teiniker

Version: 1.2.0

CC BY SA

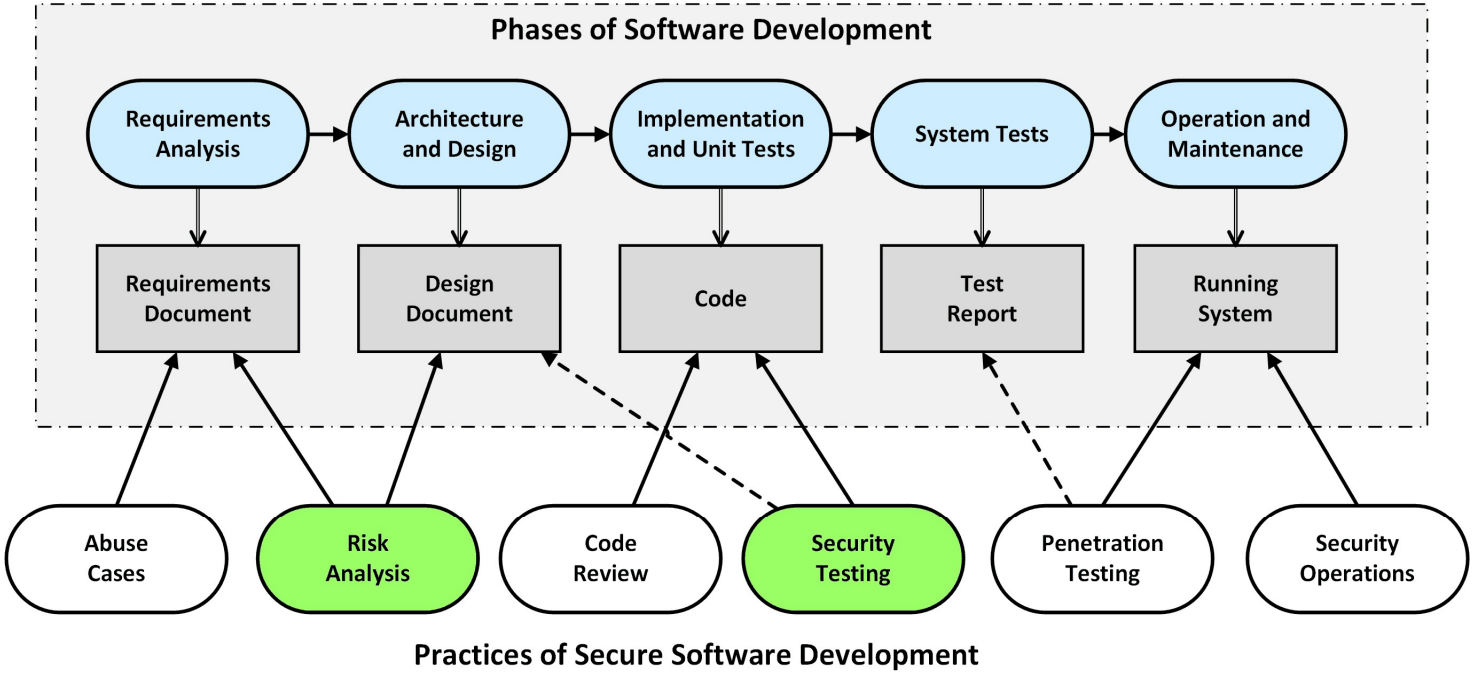
Architectural Risk Analysis

Outline

- Introduction
- Attack Surface Analysis
- Threat Modeling

Architectural Risk Analysis

Introduction



Architectural Risk Analysis

Introduction

Design flaws account for around **50% of security problems**.

At the design and architecture level, a system must be coherent and present a **unified security front**.

Designers, architects, and analysts should clearly document assumptions and **identify possible attacks**.

Security analysts **uncover and rank architectural flaws** so that mitigation can begin.

Attack Surface Analysis

Outline

- Introduction
- Understanding the Attacker's View
- Mapping the Attack Surface

Attack Surface Analysis

Introduction

The focus is on protecting an application from external attack.

Attack Surface Analysis is about mapping out which parts of a system need to be reviewed and tested for security vulnerabilities:

- Identify which **parts of the system need to be reviewed or tested** for security vulnerabilities.
- Identify **high risk areas of code** that require defense-in-depth protection.
- Identify **when we have changed the attack surface**.

Attack Surface Analysis

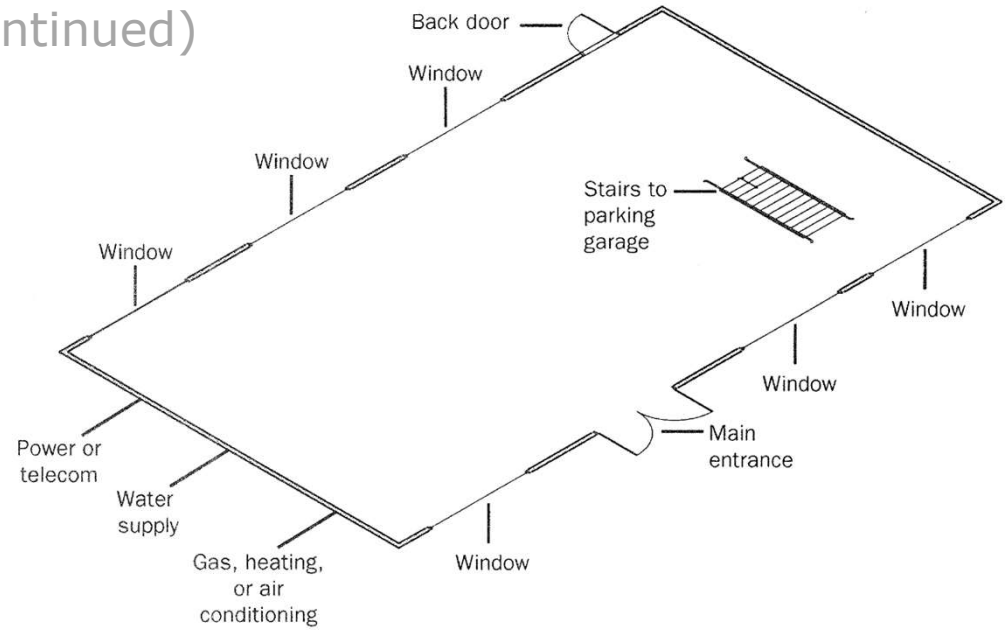
Understanding the Attacker's View

- **Entry Points**
  - Are any locations where data or control transfer between the system being modeled and another system.
  - Show all the places where an attacker can attack the system, including transfer points such as open sockets, remote procedure (RPC) interfaces, Web service interfaces, and data being read from the file system.
  - Entry points should be listed regardless of the privilege level required to interface with them.

Attack Surface Analysis

Understanding the Attacker's View

- **Entry Points** (continued)



(Swiderski & Snyder, 2004)

Attack Surface Analysis

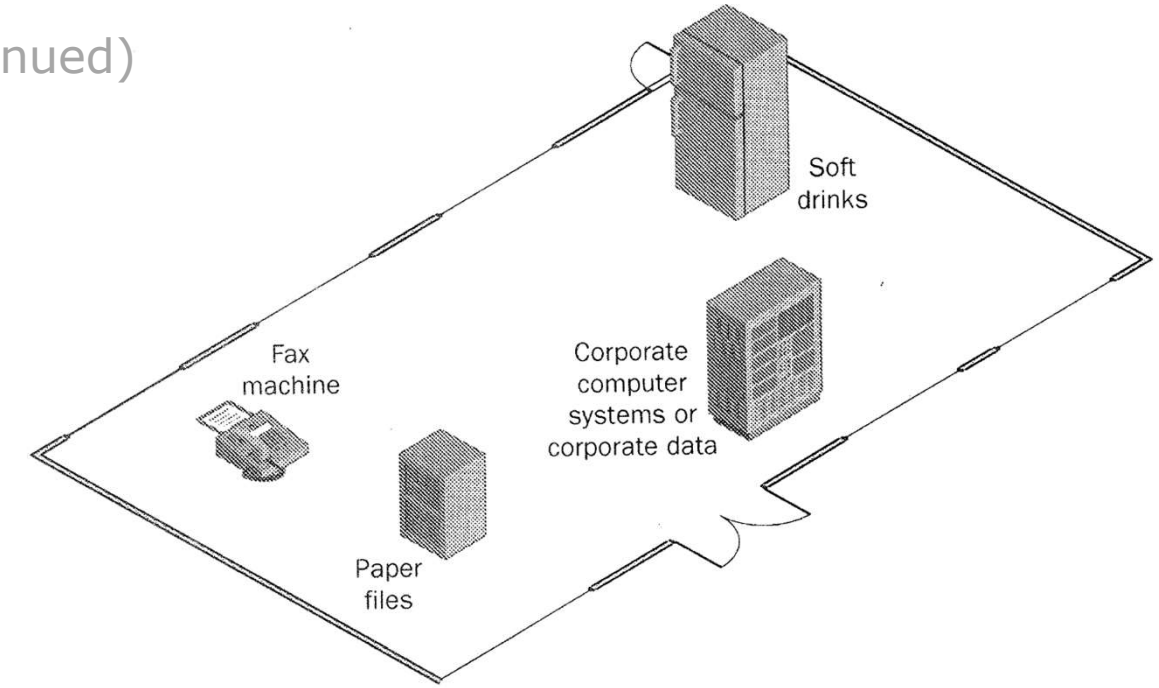
Understanding the Attacker's View

- **Assets**
  - Are the resources the system has that an attacker might try to modify, steal, or otherwise access or manipulate.
  - Can be concrete such as a session token, or more abstract, such as consistency of data.
  - Are the essentially threat targets.

Attack Surface Analysis

Understanding the Attacker's View

- **Assets** (continued)



(Swiderski & Snyder, 2004)

Attack Surface Analysis

Understanding the Attacker's View

- **Trust Levels (Roles)**

Always give just the minimum amount of privileges.

  - Are often broken down according to privileges assigned or credential supplied.
  - Define the privilege that an external entity should have to legitimately use an entry point or functionality at the entry point.
  - Dictate which assets external entities should normally be allowed to access or affect in some way.

Attack Surface Analysis

Understanding the Attacker's View

The Attack Surface describes all the different **Entry Points** and **Roles** were an attacker could get into a system, and where they could get **Assets** out.

Attack Surface of an application:

- The sum of all **paths for data and commands into and out** of the application.
- The **code that protects these paths** (including resource connection and authentication, authorization, activity logging, data validation and encoding).

Attack Surface Analysis

Understanding the Attacker's View

Attack Surface of an application: (continued)

- All **valuable data used in the application** (including secrets and keys, intellectual property, critical business data, personal data).
- The **code that protects these data** (including encryption and checksums, access auditing, data integrity, and operational security controls).

We overlay this model with the **different types of users** – roles, privilege levels – that can access the system. It is important to focus on two extremes: **anonymous users** and **highly privileged admin users**.

Attack Surface Analysis

Mapping the Attack Surface

We start building a baseline description of the Attack Surface in a **picture and notes**. We **review design and architecture documents** and **read through the source code** to identify attack points.

Examples of entry and exit points:

- User interface (UI) forms and fields
- HTTP headers and cookies
- APIs
- Files
- Databases

Attack Surface Analysis

Mapping the Attack Surface

Examples of entry and exit points : (continued)

- Other local storage
- Email and other kinds of messages
- Runtime arguments
- …

We also need to **identify the valuable data** in the application, by interviewing developers and users of the system and by reviewing the source code.

Attack Surface Analysis

Mapping the Attack Surface

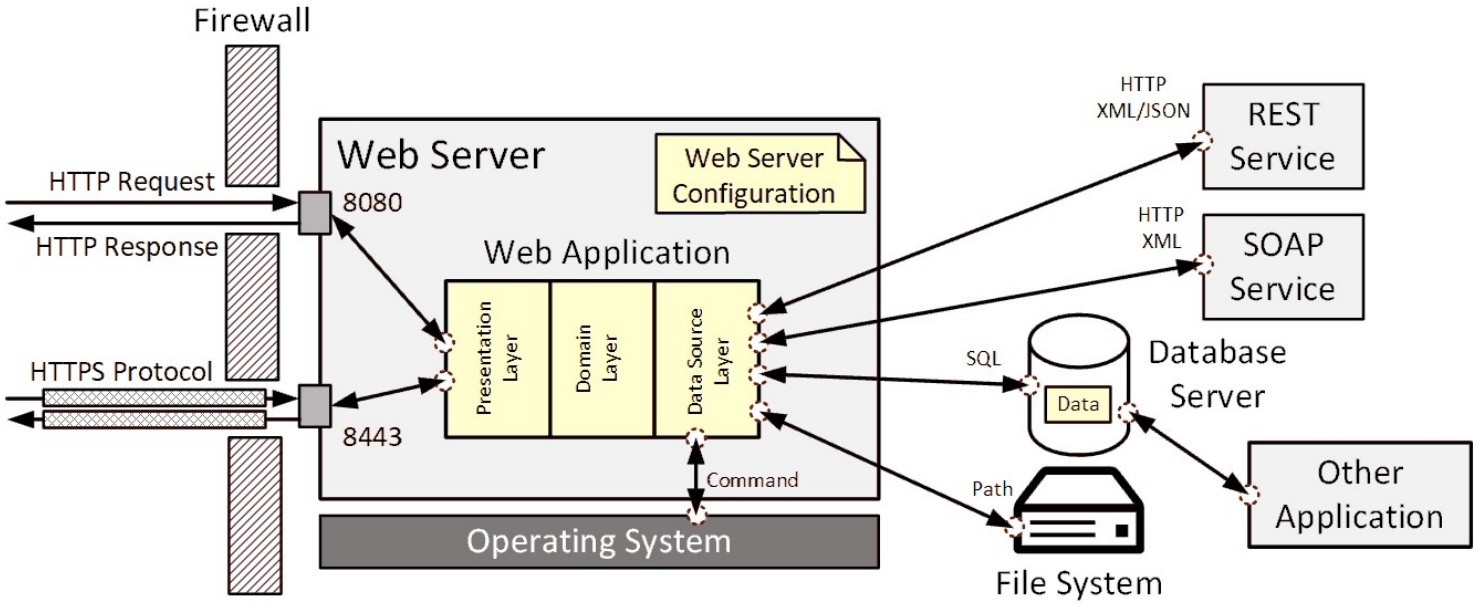
We can build up a picture of the Attack Surface by automatically **scanning the application**. For web applications we can use tools like OWASP ZAP or Burp Suite.

We can also walk through some of the main **use cases** in the system, following the **flow of control and data** through the system to see how where information is validated and where it is stored (see **Threat Modeling**).

Changes in the Attack Surface should trigger threat modeling, and threat modeling helps to understand the Attack Surface of the Application.

Attack Surface Analysis

Mapping the Attack Surface



HTTPs Protocol: Adds a Transport Layer Security to the communication on TCP/IP.

Attack Surface Analysis

FAQs

- How can we **define the Attack Surface** of an existing software system?
- How can we **map the Attack Surface** of an existing software system?

Threat Modeling

Outline

- Introduction
- Threat Modeling Process
- Threat Modeling in the Secure SDLC
- Data Flow Diagrams
- Finding Threats
  - STRIDE
  - Countermeasures

Threat Modeling

Introduction

Threat modeling is a method of **assessing and documenting** the **security risks** associated with an application.

Threat modeling looks at a system from a **hacker's perspective** to anticipate attack goals.

The threat modeling process involves **understanding a hacker's goals** in attacking a system based on the system's **assets of interest**.

Threat Modeling

Introduction

Threat modeling looks at a system's **entry points** to determine the functionality that a hacker can exercise on the system and what **assets** he can affect.

This allows developers to enumerate attack goals (**threats**).

Vulnerabilities are discovered when a threat is investigated, and safeguards are proven insufficient.

Threat modeling produces a **threat model document**.
This is a living document - as the software that was modeled changes, the document should be updated to reflect this.

Threat Modeling

Threat Modeling Process

Threat modeling is comprised of three high-level steps:

- **Understanding that attacker's view (Attack Surface Analysis)**
  Threat modeling takes an outside-in approach because such an approach most closely models the attacker's view.
- **Characterizing the security of the system**
  Modeling the system helps the team identify design-specific and implementation-specific threats.
- **Determining Threats**
  Enumerating threats creates a threat profile for a system, describing all the potential attacks.

Threat Modeling

Threat Modeling Process

Characterizing the security of the system

- **Define use scenarios**
  Developers must specify how the system will be used, and also how the system will not be used.
  Use scenarios bound the threat model discussion, pointing out situations beyond the security architecture.

- **Identify assumption and dependencies**
  Developers should collect information such as external dependencies, external and internal security notes, and implementation assumptions.

Threat Modeling

Threat Modeling Process

Characterizing the security of the system (continued)

- **Model the system**
  - **Decompose the Application** using package-, component-, and class-diagrams. We model the internal structure of an application.
  - **Data flow diagrams (DFDs)** are used to understand the actions a system performs at a given entry point. DFDs are visual representations of how a system process data.

Threat Modeling

Threat Modeling Process

Determining Threats

To create a threat profile:

- **Identify threats**
  For each entry point, the developers determine how an attacker might try to affect an asset.

- **Analyze threats**
  Developers can decompose a threat into individual, testable conditions. Threats that are not mitigated become vulnerabilities (security bugs) which must be fixed.

Threat Modeling

Threat Modeling in the Secure SDLC

Threat modeling provides a basis for **assessing the security architecture** of a system, it also can be used as a **penetration test plan** and a tool to **direct security code reviews**.

Threat modeling **starts with the design** of a software system, but threat modeling is not strictly linear – it is **iterative**.

The **threat model document** is updated during development phases, when the architecture is done, and when code is complete.

Threat Modeling

Data Flow Diagrams

Data flow diagrams (DFDs) are used in threat modeling to better understand the operation of a system. They provide a **visual representation** of **how the system process data**.

Data flow diagramming focuses on **data** as it moves through the system and the **transforms** that are applied to that data.

Principles of the data flow approach:

- An application cannot be attacked unless the attacker has a way to interact with.
- An asset of interest to the attacker must exist.

Threat Modeling

Data Flow Diagrams

Shapes used in threat modeling:

- **Process**: Represents a task in the system that processes data or performs some action based on the data.

- **Multiple Processes**: Should be used whenever the process is further broken down in a lower-level DFD.

Threat Modeling

Data Flow Diagrams

Shapes used in threat modeling: (continued)

- **External Entity**: Represents an interactor that exists outside the system being modeled and which interacts with the system at an entry point. They can interact only with processes or multiple processes.

- **Data Store**: Represents a repository where data is saved or retrieved
(e.g. registry, file system, database)

Threat Modeling

Data Flow Diagrams

No way to automate it, needs to be manually drawn. Normally just for specific cases it is done.

Shapes used in threat modeling: (continued)

- **Data Flow**: Represents data being transferred between other elements. The direction of the flow is indicated with an arrow. Each data flow represents a logical set of data (e.g. data structure, byte stream, string).

- **Privilege Boundary** (**Trust Boundary**): Represents a boundary between nodes that have different privilege levels.

Boundaries are important!

Threat Modeling

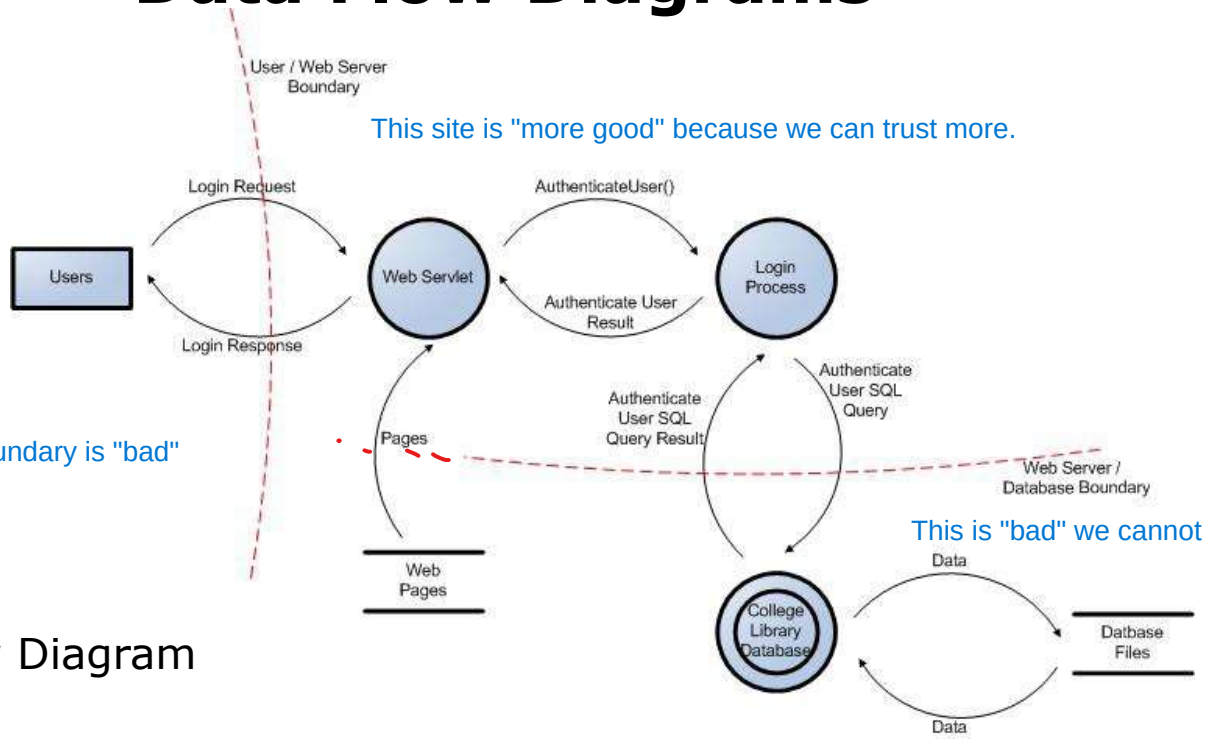
Data Flow Diagrams

DFD Example:

This site is "more good" because we can trust more.

This site of the boundary is "bad"

This is "bad" we cannot trust databases.



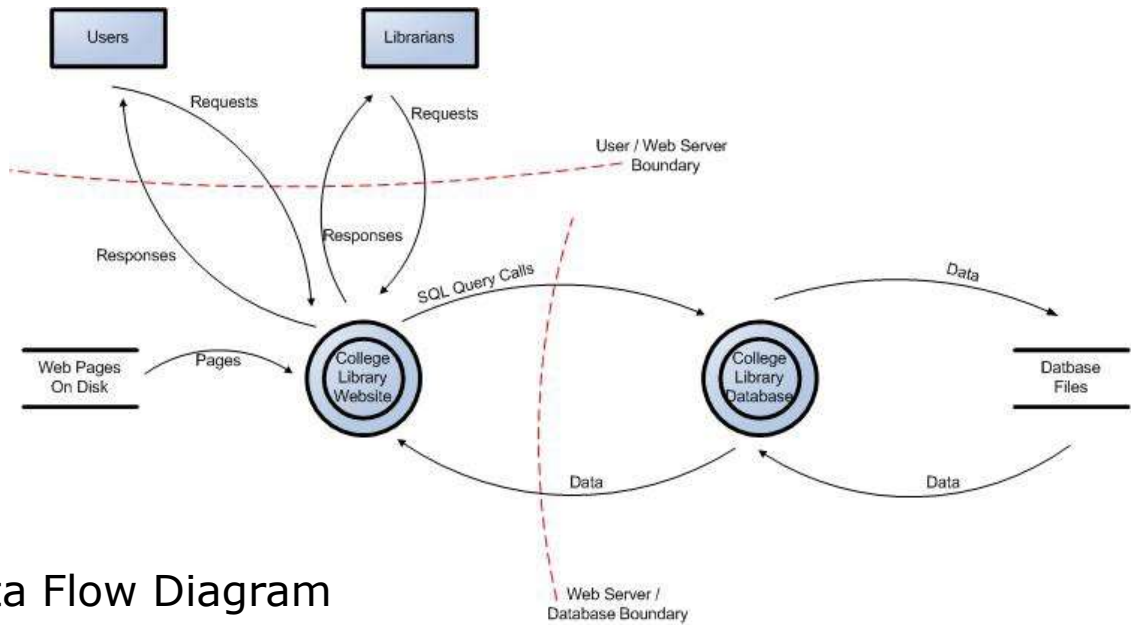
User Login Data Flow Diagram

(Shostack, 2014)

Threat Modeling

Data Flow Diagrams

DFD Example:



College Library Website Data Flow Diagram

(Shostack, 2014)

Threat Modeling

Finding Threats

There are many approaches to finding threats:

- **Brainstorming** is the most traditional way to enumerate threats. Take a set of experienced experts in a room and let them analyze the system. The quality of the brainstorm is bounded by the experience of the brainstormers and the amount of time spent.

- **STRIDE** is a mnemonic for things that can go wrong in security: **S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, **E**levation of Privilege.

Threat Modeling

Finding Threats

There are many approaches to finding threats: (continued)

- **Attack Libraries** are libraries constructed to track and organize threats. They can be very useful to those new to threat modeling (e.g. **OWASP Top10**).

Each of them has advantages and disadvantages, and different approaches may work in different circumstances.

Threat Modeling

STRIDE

We can find threats using the simple mnemonic STRIDE.

STRIDE stands for:

- **Spoofing** is pretending to be something or someone we are not.
- **Tampering** is modifying something we are not supposed to modify. It can include packets on the wire, bits on disk or in memory.
- **Repudiation** means claiming we did not do something (regardless of whether we did or not).

Threat Modeling

STRIDE

STRIDE stands for: (continued)

- **Information Disclosure** is about exposing information to people who are not authorized to see it.
- **Denial of Service** are attacks designed to prevent a system from providing services, including by crashing it, making it unusably slow, or filling all its storage.
- **Elevation of Privilege** is when a user is technically able to do things that she is not supposed to do.

Threat Modeling

Countermeasures

Threat	Countermeasure
Spoofing	Authentication
Tampering	Hashes, Signatures
Repudiation	Audit Logging
Information Disclosure	Encryption
Denial of Service	Resource Pooling
Elevating of Privilege	Authorization

Threat Modeling

FAQs

- Describe the three high-level steps of the **Threat Modeling Process**.
- Explain the terms **Entry Point**, **Exit Point**, **Asset**, **Trust Level**, and **Trust Boundary**.
- How can we **characterize the security** of a system?
- Describe the usage of **data flow diagrams** in the context of threat modeling.
- How can we use the **STRIDE** mnemonic to find threads?
- What are common **countermeasures** for STRIDE threats?

References

- *Frank Swiderski, Window Snyder*
  Threat Modeling
  Microsoft Press, 2004

- *Adam Shostack*
  Threat Modeling – Designing for Security
  Wiley, 2014

- *Jim Bird, Jim Manico*
  Attack Surface Analysis Cheat Sheet
  OWASP
  https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Attack_Surface_Analysis_Cheat_Sheet.md