# CIS 418/518 – Secure Software Engineering
# Threat Modeling

Jagadeesh Nandigam

School of Computing
Grand Valley State University
nandigaj@gvsu.edu

## Outline

1. Threat Modeling – What? and Why?
2. The Four-Step Framework for Threat Modeling
   - Step 1: What Are You Building?
   - Step 2: What Can Go Wrong?
   - Step 3: Managing and Addressing Threats
   - Step 4: Validating Threat Mitigations
3. Threat Modeling Tools

## What is Threat Modeling?

- Threat modeling is a process by which potential threats to the system you are building can be identified, enumerated, prioritized, and mitigated.
- Threat modelling can be applied to a wide range of things, including software, applications, systems, networks, distributed systems, things in the internet of things, business processes, etc.
- Two types of models are used during threat modeling:
  - A model of what you are building
  - A model of the threats

## Why Threat Model?

- Find security bugs early
- Understand your security requirements
- Engineer and deliver better products
- Address issues other techniques won't

# The Four-Step Framework for Threat Modeling

1. What are you building?
2. What can go wrong?
3. What should you do about those things that can go wrong?
4. Did you do a good enough job?

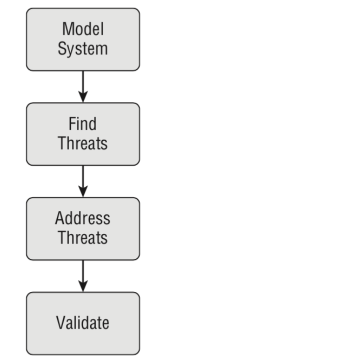


**Figure I-1:** The Four-Step Framework

# Step 1: What Are You Building?

- Making an explicit model of your software helps you look for threats without getting bogged down in details.
- Diagrams are a natural way to model software.
  - Data Flow Diagrams (DFDs)
  - Certain Diagrams from UML may be useful – Swim lane diagrams and state machine diagrams
- Data flow models are ideal for threat modeling – problems tend to follow the data flow, not the control flow.

# Step 1: What Are You Building?

- Traditional data flow diagrams consist of four elements: *data stores* and *processes* connected by *data flows*, interacting with *external entities*.

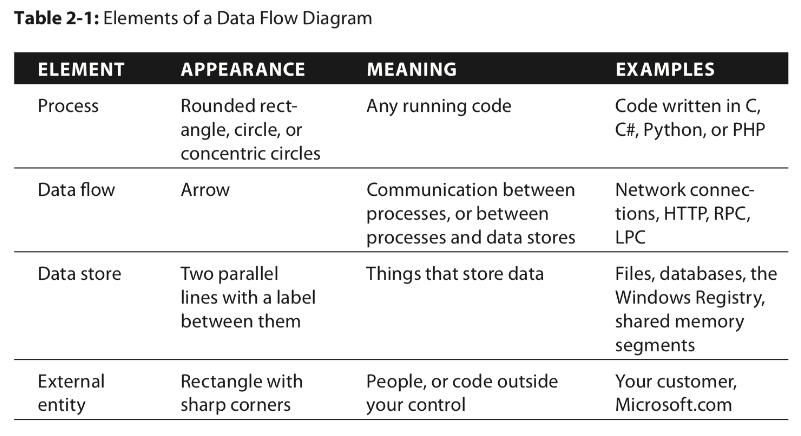


**Table 2-1:** Elements of a Data Flow Diagram

| ELEMENT | APPEARANCE | MEANING | EXAMPLES |
|---|---|---|---|
| Process | Rounded rectangle, circle, or concentric circles | Any running code | Code written in C, C#, Python, or PHP |
| Data flow | Arrow | Communication between processes, or between processes and data stores | Network connections, HTTP, RPC, LPC |
| Data store | Two parallel lines with a label between them | Things that store data | Files, databases, the Windows Registry, shared memory segments |
| External entity | Rectangle with sharp corners | People, or code outside your control | Your customer, Microsoft.com |

# Step 1: What Are You Building?

- DFDs used in threat modeling contain another element called *Trust Boundary* that is drawn where elements with different privileges interact.
- Trust boundaries should only cross data flows.

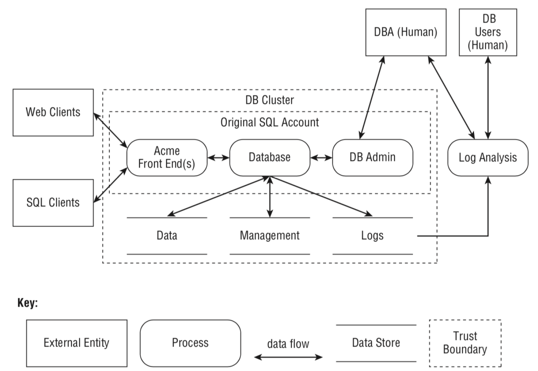


**Figure 2-1:** Data flow diagram of the Acme/SQL database

## Step 2: What Can Go Wrong?

- This step is about finding potential threats to your system.
- There are various approaches/models to finding threats:
  - STRIDE
  - Attack Trees (aka Threat Trees)
  - Attack Libraries
  - DESIST (Dispute, Elevation of Privilege, Spoofing, Information disclosure, Service denial, and Tampering)
  - PASTA (Process for Attack Simulation and Threat Analysis)
  - Trike
  - VAST (Visual, Agile, and Simple Threat modeling)

## Step 2: Finding Threats using the STRIDE model

- The goal of STRIDE is to help you find attacks/threats.
- STRIDE is a mnemonic for things that go wrong in security.
  - **Spoofing** is pretending to be something or someone you're not.
  - **Tampering** is modifying something you're not supposed to modify. It can include packets on the wire (or wireless), bits on disk, or the bits in memory.
  - **Repudiation** means claiming you didn't do something (regardless of whether you did or not).
  - **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 service, including by crashing it, making it unusably slow, or filling all its storage.
  - **Elevation of Privilege** is when a program or user is technically able to do things that they're not supposed to do.

## Step 2: Finding Threats using the STRIDE model

- The STRIDE threats are the opposite of some of the properties you would like your system to have: authenticity, integrity, non-repudiation, confidentiality, availability, and authorization.
- Using STRIDE model, walk through your model of the system (DFDs) and look for threats. Make a list of threats and the element(s) of the diagram affected.
- Threats tend to cluster around trust boundaries and complex parsing, but may appear anywhere that information is under the control of an attacker.

## Step 2: The STRIDE Threats

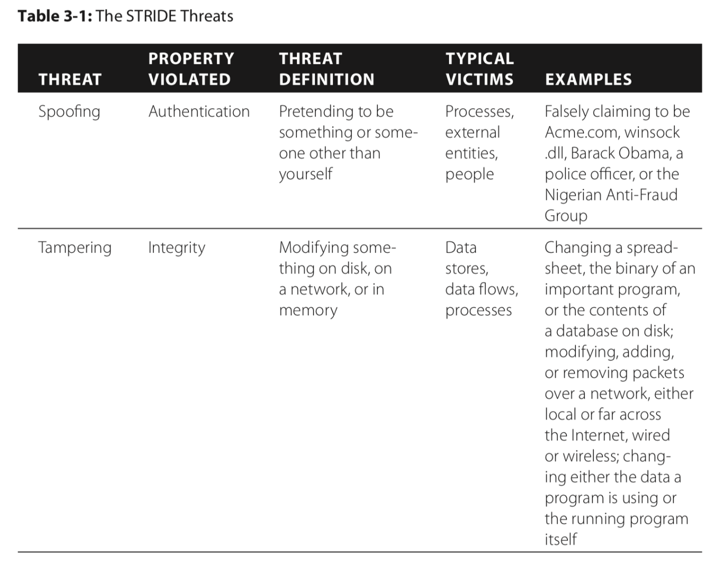
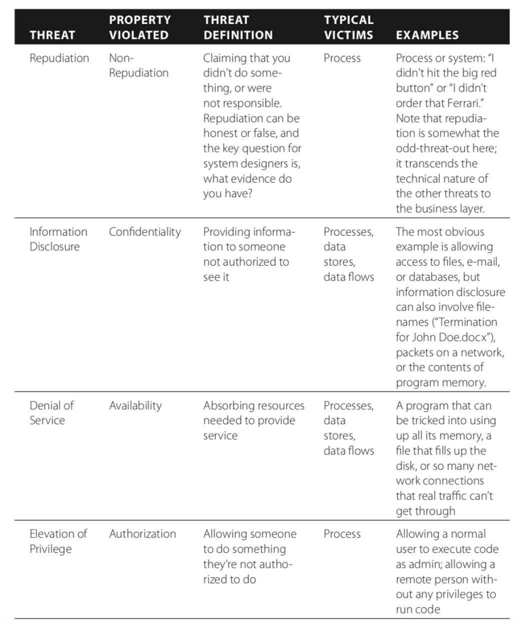


**Table 3-1:** The STRIDE Threats

| THREAT | PROPERTY VIOLATED | THREAT DEFINITION | TYPICAL VICTIMS | EXAMPLES |
|---|---|---|---|---|
| Spoofing | Authentication | Pretending to be something or someone other than yourself | Processes, external entities, people | Falsely claiming to be Acme.com, winsock .dll, Barack Obama, a police officer, or the Nigerian Anti-Fraud Group |
| Tampering | Integrity | Modifying something on disk, on a network, or in memory | Data stores, data flows, processes | Changing a spreadsheet, the binary of an important program, or the contents of a database on disk; modifying, adding, or removing packets over a network, either local or far across the Internet, wired or wireless; changing either the data a program is using or the running program itself |

## Step 2: The STRIDE Threats



| THREAT | PROPERTY VIOLATED | THREAT DEFINITION | TYPICAL VICTIMS | EXAMPLES |
|---|---|---|---|---|
| Repudiation | Non-Repudiation | Claiming that you didn't do something, or were not responsible. Repudiation can be honest or false, and the key question for system designers is, what evidence do you have? | Process | Process or system: "I didn't hit the big red button" or "I didn't order that Ferrari." Note that repudiation is somewhat the odd-threat-out here; it transcends the technical nature of the other threats to the business layer. |
| Information Disclosure | Confidentiality | Providing information to someone not authorized to see it | Processes, data stores, data flows | The most obvious example is allowing access to files, e-mail, or databases, but information disclosure can also involve filenames ("Termination for John Doe.docx"), packets on a network, or the contents of program memory. |
| Denial of Service | Availability | Absorbing resources needed to provide service | Processes, data stores, data flows | A program that can be tricked into using up all its memory, a file that fills up the disk, or so many network connections that real traffic can't get through |
| Elevation of Privilege | Authorization | Allowing someone to do something they're not authorized to do | Process | Allowing a normal user to execute code as admin; allowing a remote person without any privileges to run code |

## Step 2: Attack Trees to Find Threats

- Attack trees are multi-leveled conceptual diagrams (or textual outlines) that show how an asset or target system might be attacked.
- Attacks against a system are represented in a tree structure, with the goal (attack) as the root node and different ways of achieving that goal as leaf nodes.
- Attack trees can be drawn graphically or shown in textual outline form.
- Attack trees, as an alternative to STRIDE, work well for threat enumeration in the four-step framework.
- There are three ways you can use attack trees to enumerate threats:
  - Use an attack tree someone else created to help you find threats.
  - Create an attack tree to help you think through threats for a project you're working on.
  - Create attack trees with the intent that others will use them.

## Step 2: Attack Trees to Find Threats

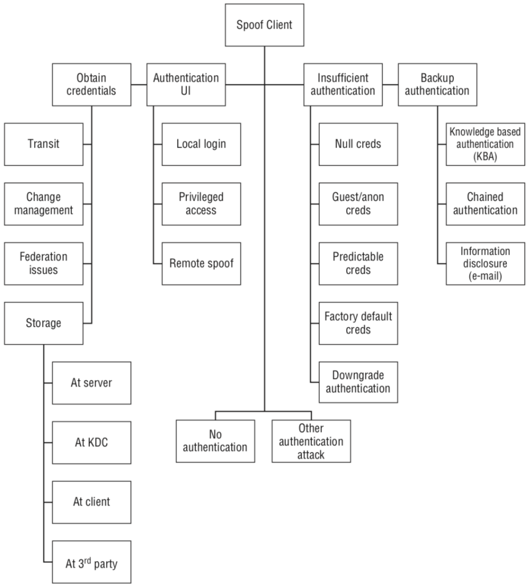


**Figure B-1:** Spoofing an external entity (client)

## Step 2: Attack Libraries to Find Threats

- A library of attacks can be a useful tool for finding threats against the system you are building.
- There are a number of attack libraries available:
  - CAPEC at `https://capec.mitre.org/`
  - CWE at `https://cwe.mitre.org/`
  - CVE at `https://cve.mitre.org/`
  - OWASP Top Ten Project at `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`

## Step 3: Managing and Addressing Threats

- Managing threats involves determining the most important threats (in terms of the risk the threat poses) so that you can prioritize your work of addressing the threats.
- Calculating Threat Risk – A Simple Method

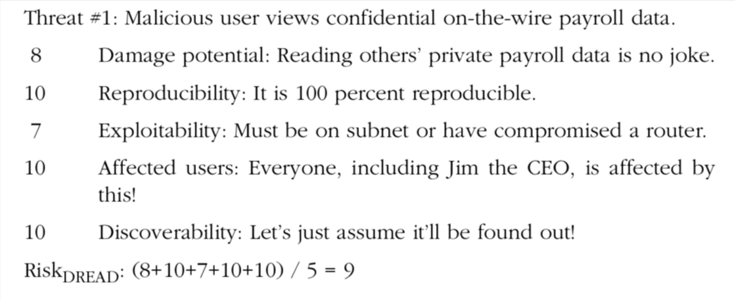
  Threat Risk = Criticality (damage potential) * Likelihood of Occurrence

  where 1 is low criticality or likelihood of occurrence and 10 is high criticality or likelihood of occurrence.

  The bigger the number, the greater the overall risk the threat poses to the system. The highest risk rating possible is 100.

## Step 3: Managing and Addressing Threats

- Calculating Threat Risk using the **DREAD** method
- The DREAD method considers five risk factors: Damage Potential (D), Reproducibility (R), Exploitability (E), Affected Users (A), and Discoverability (D).
- Each risk factor is given a value between 1 and 10.
- The threat risk is then determined by averaging the five numbers.



Threat #1: Malicious user views confidential on-the-wire payroll data.

| | |
|---|---|
| 8 | Damage potential: Reading others' private payroll data is no joke. |
| 10 | Reproducibility: It is 100 percent reproducible. |
| 7 | Exploitability: Must be on subnet or have compromised a router. |
| 10 | Affected users: Everyone, including Jim the CEO, is affected by this! |
| 10 | Discoverability: Let's just assume it'll be found out! |

Risk$_{DREAD}$: (8+10+7+10+10) / 5 = 9

## Step 3: Managing and Addressing Threats

- Once you've calculated the risk of each threat identified, sort all the threats in decreasing order of risk.
- Four types of action can be taken against each threat to address it.
  - **Mitigating threats** is about doing things to make it harder to take advantage of a threat.
  - **Eliminating threats** is almost always achieved by eliminating features.
  - **Transferring threats** is about letting someone or something else handle the risk. For example, transfer the threat handling to OS, firewall, customer, etc.
  - **Accepting the risk**
- Your "go-to" approach should be to mitigate threats. Mitigation is generally the easiest and the best for your customers. Mitigating threats can be hard work.

## Step 3: Managing and Addressing Threats

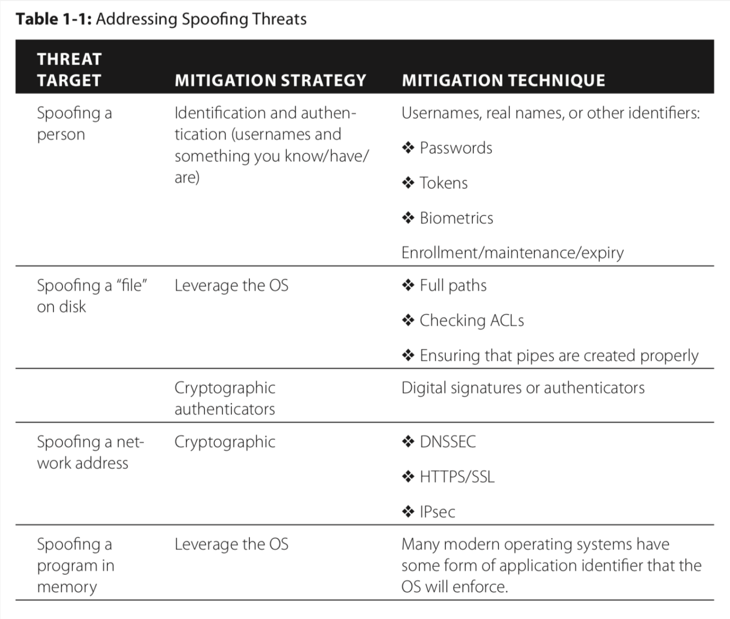


**Table 1-1:** Addressing Spoofing Threats

| THREAT TARGET | MITIGATION STRATEGY | MITIGATION TECHNIQUE |
|---|---|---|
| Spoofing a person | Identification and authentication (usernames and something you know/have/are) | Usernames, real names, or other identifiers:<br>❖ Passwords<br>❖ Tokens<br>❖ Biometrics<br>Enrollment/maintenance/expiry |
| Spoofing a "file" on disk | Leverage the OS | ❖ Full paths<br>❖ Checking ACLs<br>❖ Ensuring that pipes are created properly |
| | Cryptographic authenticators | Digital signatures or authenticators |
| Spoofing a network address | Cryptographic | ❖ DNSSEC<br>❖ HTTPS/SSL<br>❖ IPsec |
| Spoofing a program in memory | Leverage the OS | Many modern operating systems have some form of application identifier that the OS will enforce. |

# Step 3: Managing and Addressing Threats

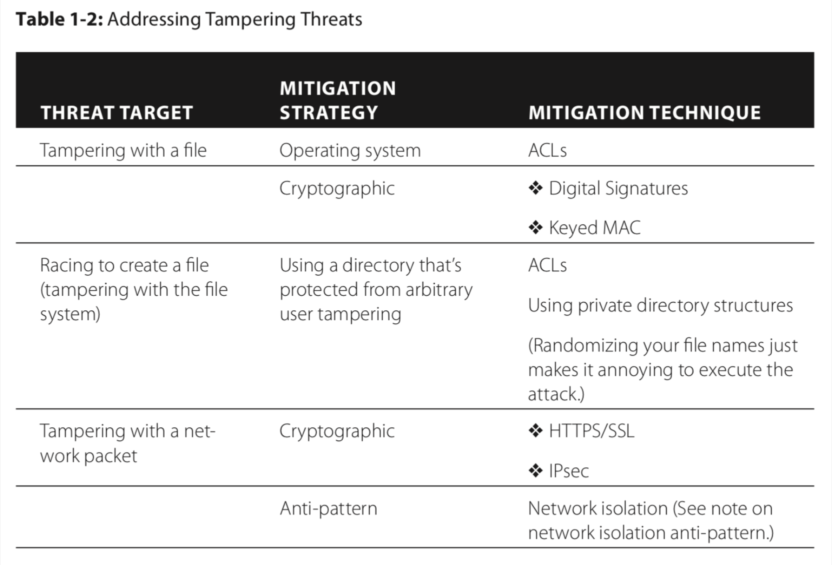


**Table 1-2:** Addressing Tampering Threats

| THREAT TARGET | MITIGATION STRATEGY | MITIGATION TECHNIQUE |
|---|---|---|
| Tampering with a file | Operating system | ACLs |
| | Cryptographic | ❖ Digital Signatures<br>❖ Keyed MAC |
| Racing to create a file (tampering with the file system) | Using a directory that's protected from arbitrary user tampering | ACLs<br>Using private directory structures<br>(Randomizing your file names just makes it annoying to execute the attack.) |
| Tampering with a network packet | Cryptographic | ❖ HTTPS/SSL<br>❖ IPsec |
| | Anti-pattern | Network isolation (See note on network isolation anti-pattern.) |

# Step 3: Managing and Addressing Threats

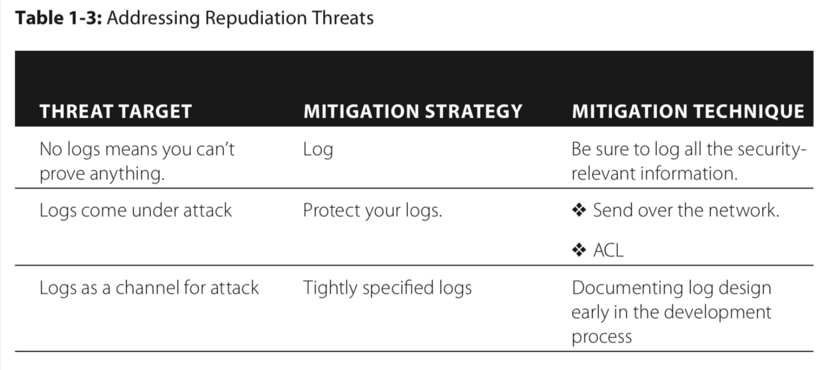


**Table 1-3:** Addressing Repudiation Threats

| THREAT TARGET | MITIGATION STRATEGY | MITIGATION TECHNIQUE |
|---|---|---|
| No logs means you can't prove anything. | Log | Be sure to log all the security-relevant information. |
| Logs come under attack | Protect your logs. | ❖ Send over the network.<br>❖ ACL |
| Logs as a channel for attack | Tightly specified logs | Documenting log design early in the development process |

# Step 3: Managing and Addressing Threats

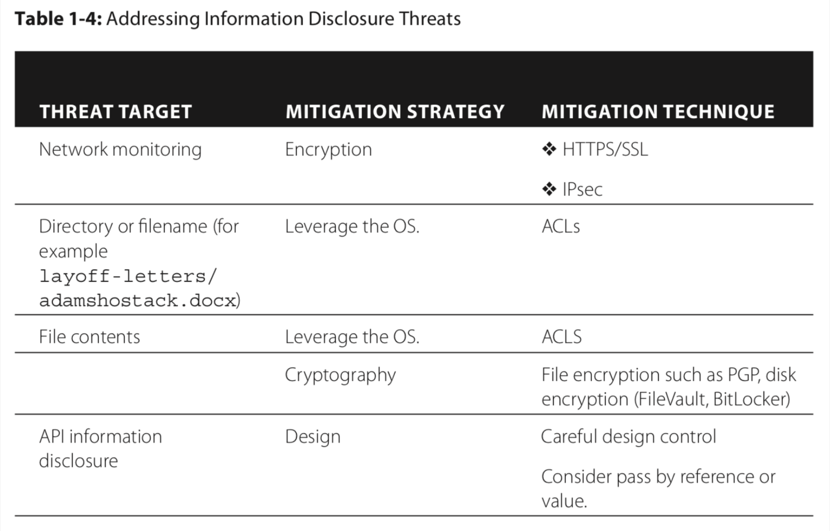


**Table 1-4:** Addressing Information Disclosure Threats

| THREAT TARGET | MITIGATION STRATEGY | MITIGATION TECHNIQUE |
|---|---|---|
| Network monitoring | Encryption | ❖ HTTPS/SSL<br>❖ IPsec |
| Directory or filename (for example `layoff-letters/adamshostack.docx`) | Leverage the OS. | ACLs |
| File contents | Leverage the OS. | ACLS |
| | Cryptography | File encryption such as PGP, disk encryption (FileVault, BitLocker) |
| API information disclosure | Design | Careful design control<br>Consider pass by reference or value. |

# Step 3: Managing and Addressing Threats

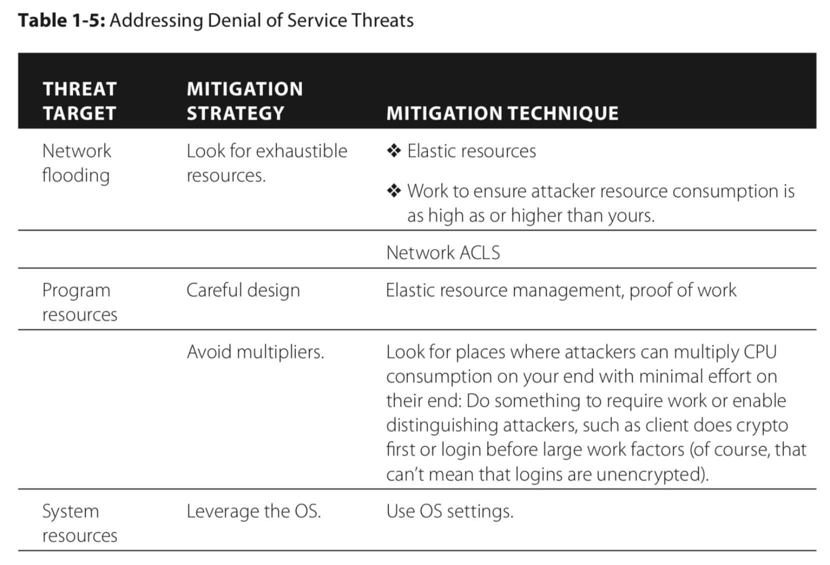


**Table 1-5:** Addressing Denial of Service Threats

| THREAT TARGET | MITIGATION STRATEGY | MITIGATION TECHNIQUE |
|---|---|---|
| Network flooding | Look for exhaustible resources. | ❖ Elastic resources<br>❖ Work to ensure attacker resource consumption is as high as or higher than yours. |
| | | Network ACLS |
| Program resources | Careful design | Elastic resource management, proof of work |
| | Avoid multipliers. | Look for places where attackers can multiply CPU consumption on your end with minimal effort on their end: Do something to require work or enable distinguishing attackers, such as client does crypto first or login before large work factors (of course, that can't mean that logins are unencrypted). |
| System resources | Leverage the OS. | Use OS settings. |

# Step 3: Managing and Addressing Threats

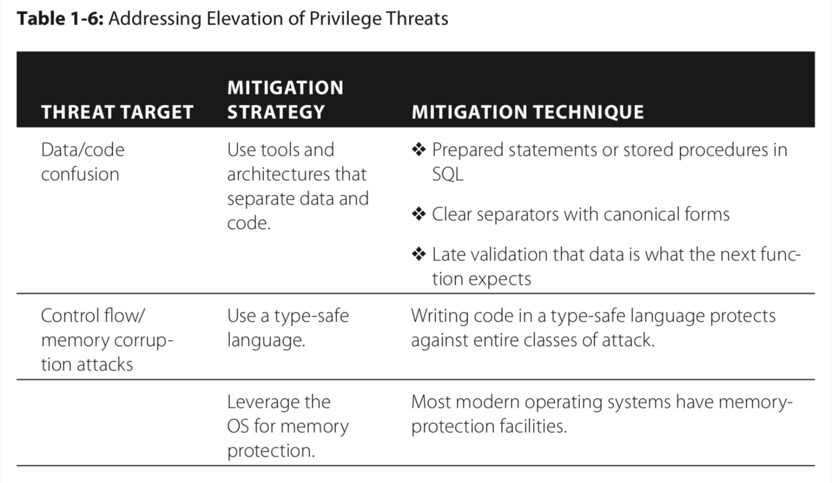


**Table 1-6:** Addressing Elevation of Privilege Threats

| THREAT TARGET | MITIGATION STRATEGY | MITIGATION TECHNIQUE |
|---|---|---|
| Data/code confusion | Use tools and architectures that separate data and code. | ❖ Prepared statements or stored procedures in SQL<br>❖ Clear separators with canonical forms<br>❖ Late validation that data is what the next function expects |
| Control flow/ memory corruption attacks | Use a type-safe language. | Writing code in a type-safe language protects against entire classes of attack. |
| | Leverage the OS for memory protection. | Most modern operating systems have memory-protection facilities. |

# Step 4: Validating Threat Mitigations

- It's important to "close the loop" and ensure that threats have been appropriately handled.
- If you think of threats as bugs, then testing threats is much like doing heavy testing on those bugs.
- Integrate thread model testing into your test process.

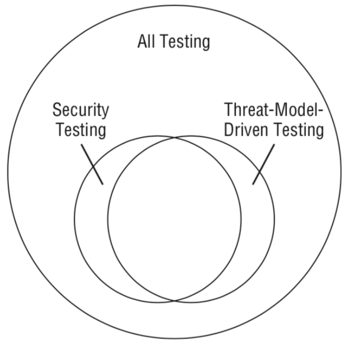


**Figure 10-1:** Different types of testing

# Step 4: Validating Threat Mitigations

- Validation tasks include:
  - Checking the model of the system – updating diagrams and diagram details.
  - Checking each threat – checking you did the right thing with each threat found and asking if you found all the threats you should find.
  - For each threat that you address, ensure you've built a good test to detect the problem and the threat has been properly mitigated.

# Bringing It All Together

- Determine the *threat targets* from functional decomposition.
- Determine types of threat to each component in functional decomposition by using STRIDE model.
- Use *threat trees* to determine how the threat can become a vulnerability.
- Apply a ranking mechanism, such as DREAD, to each threat to prioritize them.
- Mitigate threats and validate threat mitigations.

## Threat Modeling Tools

- General Useful Tools
  - Whiteboards
  - Office Suites – Word, Excel, Vision, etc.
  - Bug-tracking systems
- Open Source Tools
  - TRIKE (spreadsheet-based)
  - SeaMonster (an Eclipse-based tool, now abandoned but code available!!)
  - Elevation of Privilege (game of cards from Microsoft designed to help you threat model)
- Commerical and Free Tools
  - ThreatModeler
  - Corporate Threat Modeller
  - SecurITree
  - Little-JIL
  - MS SDL Threat Modeling Tool

## References

- Adam Shostack, *Threat Modeling – Desiging for Security*, Wiley.
- Michael Howard and David LeBlanc, *Writing Secure Code*, Chapter 4 on *Threat Modeling*, Microsoft.