

Secure Software Design and Engineering Lab Manual



**Jazia.sajid Lab Engineer FCSE
Ghulam Ishaq Khan Institute of Engineering and Technology, Pakistan**

Objectives

Lab 03: Threat Modeling

- Learn how to perform threat modeling to identify potential security vulnerabilities.
- Analyze and evaluate various threat scenarios in software applications.
- Develop the skills to prioritize security risks and plan mitigation strategies.

Introduction

Threat modeling is a critical process in cybersecurity that helps organizations understand their security posture. It's a structured approach to identifying, quantifying, and addressing the security risks associated with an application or system. Think of it like a proactive security strategy - by understanding potential threats early in the development process, you can build more secure systems from the ground up. That's a lot better than scrambling to patch vulnerabilities later. And much less expensive!

Benefits:

I've seen firsthand how threat modeling can benefit an organization:

- It helps identify security threats early, when they're easier and cheaper to fix.
- It provides a systematic way to evaluate risk and prioritize security efforts.
- It encourages collaboration between security and development teams.
- It helps ensure that security is an integral part of the design process, not an afterthought.

Common Misconceptions About Threat Modeling

Let's talk about the elephant in the room – those myths about threat modeling that just won't go away:

1. **"Threat modeling is too time-consuming and expensive."** a. In reality, the cost of fixing security issues increases exponentially the later they're discovered. Threat modeling can actually save time and money in the long run.
2. **"Threat modeling is only for big companies with mature security programs."** a. False. Organizations of all sizes can benefit from threat modeling. It's about making security a priority, not the size of your security budget.
3. **"Threat modeling is a one-time activity."** a. Wrong. Threat modeling should be an ongoing process. As new threats emerge and systems evolve, threat models need to be regularly reviewed and updated.

Secure Development Lifecycle & Threat Modeling

Threat modeling is integrated into the SDLC as follows:

- 1. **Design Phase:** Define security requirements, create architectural diagrams
- 2. **Build Phase:** Follow secure coding guidelines, conduct code reviews
- 3. **Test Phase:** Perform security testing
- 4. **Production Phase:** Configure security settings, monitor security events

Threat Modeling Stages

- 1. **Diagram:** Identify system components, data flow, and trust boundaries
- 2. **Identify Threats:** Apply STRIDE methodology
- 3. **Mitigate Threats:** Implement security controls
- 4. **Validate & Report:** Ensure mitigations are effective and document the model

STRIDE Threat Classification

Threat	Description
Spoofing	Attacker gains unauthorized access using false identity
Tampering	Data modification during transmission or storage
Repudiation	Ability of an attacker to deny performing an action
Information Disclosure	Unauthorized access to sensitive data
Denial of Service (DoS)	Disrupting system availability
Elevation of Privilege	Attacker gains higher privileges than intended

Diagramming

Use **Data Flow Diagrams (DFDs)** to visualize system components and trust boundaries:

- **External Entity:** Interacts with the system (e.g., users, external systems)
- **Process:** Handles data within the application
- **Data Store:** Stores data but does not modify it

- **Data Flow:** Movement of data within the application
- **Trust Boundary:** Change of trust level in data flow

Identifying Threats with STRIDE

Apply STRIDE threats to each element in a diagram:

- **External Entity:** Spoofing, Tampering
- **Process:** Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege
- **Data Store:** Tampering, Information Disclosure, Denial of Service
- **Data Flow:** Tampering, Information Disclosure, Denial of Service

Addressing Threats

- **Authentication:** Prevent spoofing with strong authentication mechanisms
- **Integrity:** Use cryptographic hashing/signing to prevent tampering
- **Non-repudiation:** Implement logging and auditing mechanisms
- **Confidentiality:** Use encryption to protect data
- **Availability:** Implement redundancy and DoS mitigation techniques
- **Authorization:** Use role-based access control (RBAC) to prevent privilege escalation

Creating a Data Flow Diagram for Threat Modeling

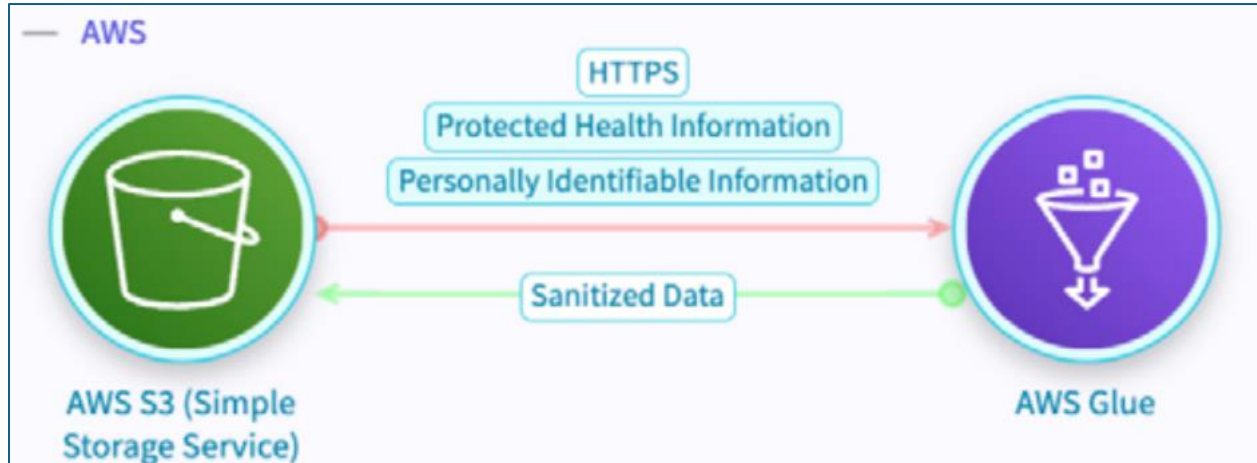
One of the most useful tools in the threat modeling practitioner's toolkit is the data flow diagram (DFD). A DFD is a graphical representation of how data moves through a system. Drawing up a DFD is an awesome way to get a bird's-eye view of your system and pinpoint any potential entry points that hackers might exploit. **Trust me, when you can see the whole.**

Elements of a Data Flow Diagram A DFD consists of four main elements:

1. External entities - these are outside actors that interact with the system, like users or third-party services.
2. Processes - these are the activities that manipulate data within the system.
3. Data stores - these are the places where data is stored, like databases or files.

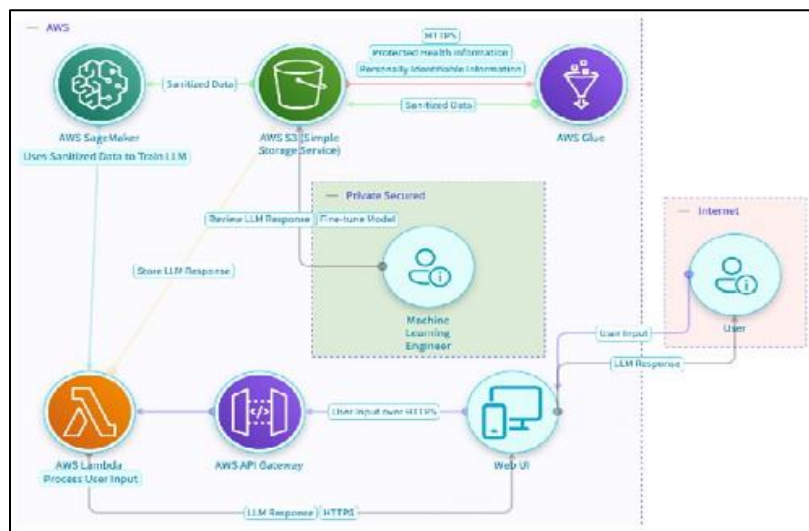
4. Data flows - these are the paths that data takes as it moves through the system. Visualizing the system's components and how they interact reveals the path data flows take, exposing potential weak points along the way.

Example:

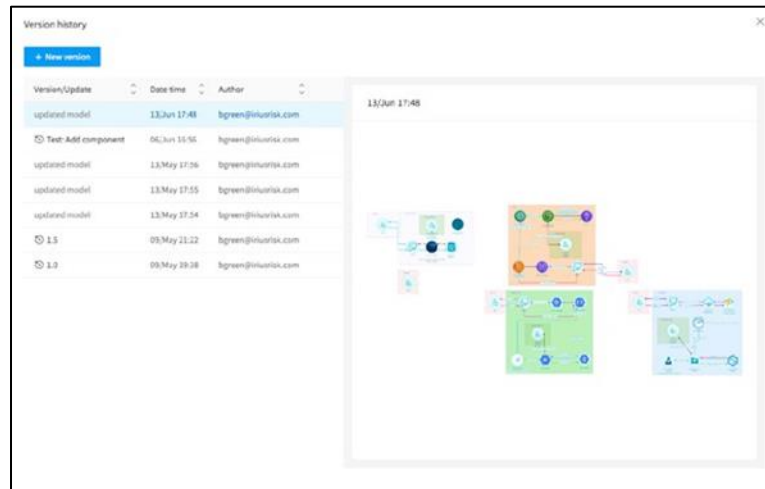


Best Practices for Creating Data Flow Diagrams Creating effective DFDs is an art, and I've got some insider tips from my experience to help you master it.

- **Keep it simple** - start with a high-level diagram and add detail as needed
- **Use consistent notation** - this makes the diagram easier to understand and maintain
- **Focus on the data** - the goal is to understand how data moves through the system, so don't get bogged down in implementation details
- **Collaborate with stakeholders** - DFDs are a great communication tool, so involve the right people in their creation and review them at least annually.



Remember, a DFD is a living document. As the system changes, so should the diagram. Regular updates will help ensure that your threat model stays accurate and relevant.



Create different versions of the diagram as changes are made so you can see the changes over time.

Identifying and Analyzing Potential Threats

Once you've got your data flow diagram created, it's time to put on your hacker hat and start thinking like an attacker. This is where the real fun begins - identifying potential threats at each point in your diagram. It's all about getting inside the mind of a malicious actor and considering all the creative ways they might try to exploit vulnerabilities and compromise your system.

The STRIDE threat modeling framework is great for this. STRIDE helps you break down threats into six main categories:

STRIDE Threat Model

STRIDE stands for:

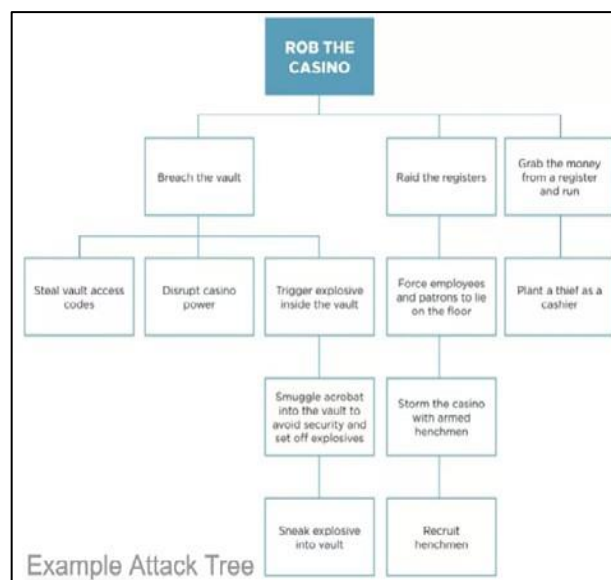
- Spoofing: Attackers pretending to be someone or something they're not to gain unauthorized access
- Tampering: Modifying data or code for malicious purposes

- Repudiation: Performing actions that can't be traced back, denying involvement
- Information Disclosure: Exposing sensitive info to unauthorized parties
- Denial of Service: Disrupting system availability and resources
- Elevation of Privilege: Gaining higher-level permissions than authorized

By systematically analyzing each element of your data flow diagram through the lens of STRIDE, you can uncover a wide range of potential threat scenarios to assess and prioritize.

Attack Trees

Imagine an attack tree as a roadmap used by cybercriminals to infiltrate your business. Like mind maps, these visual diagrams dissect every potential route an attacker might take to reach their goal, whether it's stealing sensitive data or disrupting your operations. Each branch of the tree represents a different attack scenario, detailing the specific steps and vulnerabilities an attacker could exploit to achieve their objective.



For instance, consider a simple attack tree targeting a company's financial data:

- **Goal:** Steal customer credit card information.
- Branch 1: Phishing attack

Step 1: Send a fraudulent email impersonating a trusted source.

Step 2: Trick employees into clicking a malicious link.

Step 3: Install malware on the employee's computer.

Step 4: Use malware to exfiltrate data from the company's network.

- **Branch 2: SQL injection attack**

Step 1: Identify a vulnerable web application.

Step 2: Craft a malicious SQL query.

Step 3: Inject the query into the web application's input field.

Step 4: Extract sensitive data from the database.

By mapping out these attack paths, you gain a comprehensive understanding of the various ways your system could be compromised. This insight allows you to prioritize your defenses, fortify vulnerabilities, and develop mitigation strategies for each scenario. Attack trees reveal the “attack surface” – the total area where your systems are exposed to potential threats – and empower you to proactively defend against them.

Threat Intelligence


Speaking of threat intelligence, it's an absolute must for a robust threat model. You've got to stay plugged into the latest threat data from reputable sources like US-CERT, your industry's ISAC, and commercial threat intel feeds. I've learned the hard way that you can't just rely on your own experiences and imagination when it comes to anticipating threats. Real-world threat data on active threats, vulnerabilities, and attack patterns needs to inform your threat modeling process.

Prioritizing and Managing Identified Risks

So you've uncovered a bunch of potential threats - now what? It's time to prioritize based on risk level and figure out your plan of attack (pun intended). Not all threats are created equal. You've got to assess the likelihood and potential impact of each one to determine which to tackle first.

This is where risk rating methodologies come into play.

Threat details

 AWS API Gateway / Networking

An attacker eavesdrops on the communication between the clien...


Medium Risk

Description:

Eavesdropping on communication is a network attack that captures network packets transmitted by other computers and reads the data content. This type of network attack is most effective when weak encryption services are used. An attacker could eavesdrop on the communication between the client and server and decrypt the encrypted data.


8 Recommended

☐

 AWS API Gateway


Enable request validation in API Gateway

☐

 AWS API Gateway


Use client-side SSL certificates for HTTP backend authentication

☐

 AWS API Gateway

Rotate Expiring SSL Client Certificates

☐

 AWS API Gateway

Enable Content Encoding for API Responses

Risk Rating Methodologies

There are a bunch of different risk rating models out there, like CVSS or a custom scoring system. Whichever one you choose, the key is to consistently evaluate each threat based on factors like:

1. How easy is it to exploit?
2. How much damage could it cause?
3. How many users would be impacted?

4. What's the risk to sensitive data?
5. Could it result in compliance violations?

Threat details

AWS API Gateway / Networking

An attacker injects new items into an existing command to execu...

High Risk

Description:

An adversary looking to execute a command of their choosing, injects new items into an existing command thus modifying interpretation away from what was intended. Commands in this context are often standalone strings that are interpreted by a downstream component and cause specific responses. This type of attack is possible when untrusted values are used to build these command strings. Weaknesses in input validation or command construction can enable the attack and lead to successful exploitation.

Countermeasures

- 1 Improper Neutralization of Special Elements used in a Command ('Command I...

Risk summary

Inherent risk:	High
Current risk:	High
Projected risk:	High

Test state

Running through these factors helps you put numbers to the risk and rack and stack your priorities. In my experience, a solid risk rating process is crucial for getting everyone on the same page and making smart tradeoffs.

Determining Risk Appetite

You've also got to have a good handle on your organization's risk appetite - how much risk are you willing to accept? Every company has a different threshold based on their industry, regulations, and culture.

Threats that fall below that line can be deprioritized or accepted, while those that exceed it need to be mitigated asap.

And it's not a one-and-done deal - **risk appetite can shift over time**, so it needs to be revisited regularly.

Implementing Security Controls

Once you've got your prioritized list of risks, it's time to implement security controls to mitigate them.

Security controls can be preventative (stopping threats before they happen), detective (identifying threats in progress), or corrective (minimizing impact after the fact)

The screenshot shows the 'Countermeasure' interface in IriusRisk. At the top, there's a title 'Countermeasure' and a 'Recommended' status badge. Below the title is the countermeasure name: 'Use client-side SSL certificates for HTTP backend authentication'. A 'Medium priority' dropdown and a 'Create issue...' button are visible. A list of priority levels (Very high, High, Medium, Low) is shown, with 'Medium' selected. The main content area contains a detailed description of the countermeasure, a 'Note' about backend server compatibility, and an 'Impact' section. On the right, a 'Fields' panel shows 'Test state' with a 'Partially tested' result, 'Result source' set to 'Manual', and an 'Expiry date' field with a calendar icon.

Countermeasure

Recommended

Save

Use client-side SSL certificates for HTTP backend authentication

Medium priority

Create issue...

Calculated (Medium)

Very high

High

Medium

Low

is for HTTP backend authentication within AWS API Gateway.

can be used to verify that HTTP requests to your backend system are from API Gateway. This allows your HTTP backend to control and accept only requests that originate from Amazon API Gateway, even if the backend is publicly accessible.

You can use API Gateway to generate an SSL certificate and then use its public key in the backend to verify that HTTP requests to your backend system are from API Gateway. This allows your HTTP backend to control and accept only requests that originate from Amazon API Gateway, even if the backend is publicly accessible.

Note

Some backend servers might not support SSL client authentication as API Gateway does and could return an SSL certificate error. For a list of incompatible backend servers, see Amazon API Gateway important notes.

The SSL certificates that are generated by API Gateway are self-signed, and only the public key of a certificate is visible in the API Gateway console or through the APIs.

Impact:

None

Fields

Test state

Test result: Partially tested

Result source: Manual

Expiry date: dd/mm/yyyy

The key is striking the right balance. Try to bake security in as early as possible with secure design principles, rather than bolting things on later. This is referred to as “Shifting Left”.

Addressing risks early in the development process with the right security requirements and architecture decisions is way more effective than scrambling to patch things up post-release.

This is the benefit of Threat Modeling!

IriusRisk

IriusRisk is a powerful automated threat modeling tool designed to streamline and simplify the threat modeling process for teams of all sizes.

By leveraging a vast library of threat patterns and security standards, IriusRisk automates the identification and prioritization of threats based on your specific system architecture.

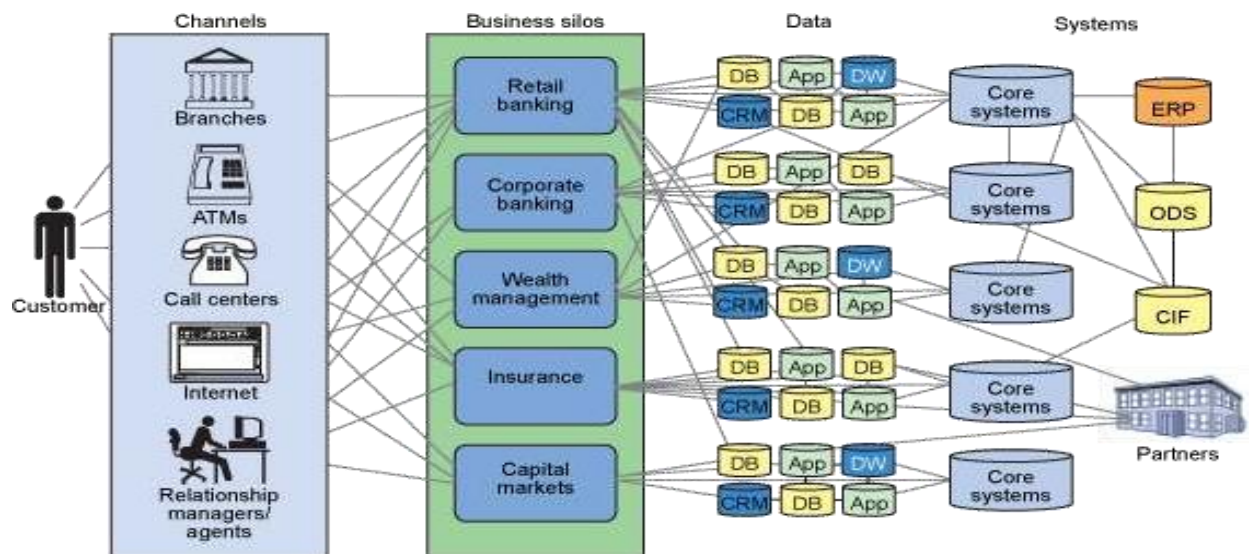
This significantly reduces the time and effort required for manual threat modeling, allowing you to focus on implementing effective countermeasures.

[illegible]

leading to mistakes in protection mechanisms. Code changes are often made without proper testing, making production systems as vulnerable as test environments.

To ensure security at every stage of an online bank's lifecycle, strong security processes must be in place. Implementing a Secure Software Development Lifecycle (SSDLC) helps prevent many security issues but does not replace the need for regular web application security assessments. White-box testing, which examines the source code, is more effective than gray-box and black-box methods. Additionally, using a Web Application Firewall (WAF) helps prevent attackers from exploiting vulnerabilities introduced through code changes.

In a Core Banking System, software applications record transactions, maintain customer information, calculate interest on loans and deposits, and more. Instead of using large physical ledgers, data is stored in backend databases digitally. The same software can be installed across multiple bank branches and interconnected via the internet or telephone lines to form a core banking network. This allows customers to access their accounts from any branch. If the bank offers Internet Banking or ATM facilities, customers can manage their accounts from virtually anywhere.



Key Observations:

1. Off-the-shelf online banking solutions often have better baseline security than custom in-house applications.
2. Developers frequently prioritize functionality over security, introducing vulnerabilities during code changes.
3. Unchecked code changes and inadequate testing exacerbate the risk of security breaches.

Recommendations:

- Implement a Secure Software Development Lifecycle (SDLC) to reduce errors.
- Conduct regular security assessments of web applications.
- Use white-box testing to analyze source code for vulnerabilities.
- Deploy a Web Application Firewall (WAF) to mitigate exploitation risks.

Task for Students

Students are expected to:

1. Identify potential threats and vulnerabilities in the core banking system, considering risks at different stages—data input, processing, storage, and output.
2. Analyze the impact and likelihood of each identified threat.
3. Prioritize threats based on severity.
4. Propose technical, procedural, and policy-based solutions.
5. Develop a mitigation plan for the top-priority threats.
6. Create a threat model using the IriusRisk.
7. Provide detailed report.

ACTIVITY:

Choose one case study from IriusRisk's website (<https://www.irusrisk.com/case-studies>) and analyze the organization's approach to threat modeling. Answer the following:

1. What security challenges did the organization face?
2. How did they implement threat modeling to address these challenges?
3. What were the key outcomes and benefits of their approach?
4. What lessons can be learned and applied to other organizations or projects?