Threat Modeling & Risk Assessment for Developers

Process Guide

Table of Contents

Introduction

Threat modeling and risk assessment is a structured approach that enables an organization to identify, quantify, and address the threats to a system based on risk to the business. It involves understanding the system from an attacker's perspective, which can significantly enhance the security measures. The primary goal of threat modeling is to provide the team with a systematic analysis of what controls or defenses need to be included, given the nature of the system, the data it must protect, and the potential threats to that data.

Threat modelling has traditionally been applied at the architecture level, looking at system components and data flows to identify attack pathways. When considering an application system, additional consideration should be given to requirements or user-stories, so abuse cases can be identified early on and considered during design. This will save the organization the additional cost and headaches of identifying flaws later when the system is in production.

The threat modeling process is iterative and should be repeated as necessary throughout the lifecycle of a system to reflect changes in threats and the environment. At a minimum, it should be applied early in the application life cycle when high level functionality and architecture is defined, also iteratively at the requirement or user-story level to determine abuse cases and design accordingly. It is also recommended that threat modelling is repeated at the architecture level periodically (at least once a year).

Architecture-level Threat Modeling

This is the traditional threat modelling process that can be applied to any system architecture. The following key steps are involved:

1. **Identifying assets, impact, and security objectives:** Clearly outline what digital assets need to be protected and why. Consider possible threat actors, and assets at risk, and impact to those assets because of a successful attack.
2. **Creating an application overview:** Map out the application's architecture, including data flows and external entities. Review the security design, and document what is in place, as well as any gaps.
3. **Decomposing the Application:** Break down the application into its components, interfaces, and data stores, then identify trust boundaries.
4. **Identifying and Prioritizing Threat Scenarios:** identify threat scenarios based on security gaps identified and assign risk rating based on impact and likelihood to assist with prioritization.
5. **Countermeasures and Risk Mitigation:** Develop strategies to mitigate identified threat scenarios and reduce risk to acceptable levels.

The process of threat modeling can be broken down into three primary stages, with each phase thoroughly documented throughout its execution. The culmination of this process is a comprehensive threat model for the application in question.

Step 1. Identifying assets, impact, and security objectives

When an attack on a system occurs, it often results in a breach of the core principles of information security: **confidentiality**, **integrity**, and **availability**, commonly known as the CIA triad. For instance, data exfiltration can lead to a breach of confidentiality as sensitive information is illicitly transferred from the system to an external location, falling into unauthorized hands. Data modification undermines data integrity, where unauthorized alterations to data can result in misinformation, corrupt processes, or flawed decision-making. Lastly, Data deletion attacks strike at availability, where critical data is removed, causing service disruption and potentially leading to significant downtime or loss of trust. Each of these represents a serious security incident that can have far-reaching consequences for the organization, its stakeholders, and its customers.

A business impact assessment should be performed to document the digital assets handled by the system (e.g. PII, payment data, health data, business critical data, etc.), any resources that incur cost (e.g. memory, CPU), and related impact to each asset when a potential threat scenario takes place in the future.

The impact of a threat scenario to an asset is the effect it would have on the business in terms of 3 factors: reputational damage, affected users, and financial damage. These are used along with likelihood values to determine the risk for a given threat scenario (see “Identifying and Prioritizing Threat Scenarios” section). Impact can be assigned based on the following general guidance (can be adjusted for each organization):

a) **Reputational Damage**
   0. **None**
   1. **Minimum:** Negative article on local news - Very limited number of people will care about this - Limited potential for customer loss.
   2. **Moderate:** Negative article on regional news - Might lose some customers and not able to easily attract new customers.
   3. **High:** Negative article on national/international news - Tarnish the brand permanently - Lose most customers, very difficult to attract any new customers. World ending situation including fines, contractual breach, lawsuits, etc.

b) **Affected Users**
   0. **No affected users**
   1. **Single user** can be exploited at a time.
   2. **Some users** of the system or application are impacted.
   3. **Majority of users** impacted.

c) **Financial Damage** (this is specific per company)
   0. **None**
   1. **Hundreds**
   2. **Thousands**
   3. **Millions**

Step 2. Creating an application overview

Creating an application overview is a foundational step in the threat modeling process, serving as a blueprint for understanding the application's architecture, data flow, and interaction with external entities. This comprehensive overview involves documenting the application's components, such as servers, databases, and third-party services, and mapping out how information is processed and stored.

It also includes detailing the user roles and their interactions with the application, as well as identifying entry and exit points that could potentially be exploited. This holistic view not only aids in pinpointing areas of vulnerability but also facilitates clear communication among team members and stakeholders regarding the application's structure and potential security risks. By establishing a clear picture of the application, developers and security professionals can more effectively strategize and prioritize their efforts to protect against threats.

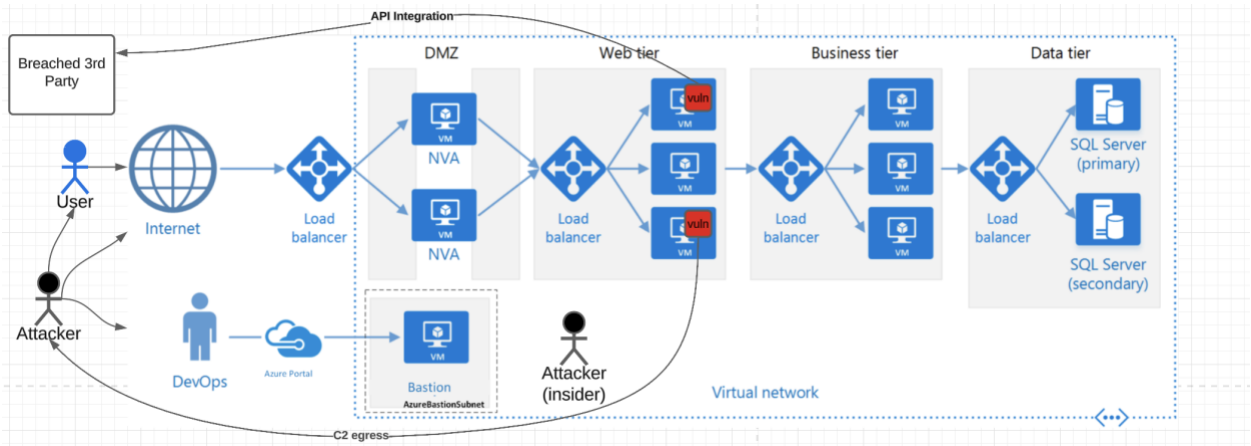

Figure 1: Example 4-Tier Application Architcture with Users and Threat Actors

Step 3: Decomposing the Application

This phase focuses on comprehensively understanding the application and its interactions with external systems. This phase includes:

- Developing high level use cases to grasp how the application is utilized.
- Identifying points of entry to determine potential interaction points for attackers.
- Pinpointing critical assets that may attract attackers.
- Establishing different levels of trust to define the access privileges for external entities.

The findings from this step are documented and used to create data flow diagrams that map out the application's pathways, emphasizing areas of privilege transition (see figure 1).

Step 4: Identifying and Prioritizing Threat Scenarios

Identifying threats is pivotal and involves employing a threat classification system. Architecture diagrams and data flows from the previous steps aid in pinpointing potential targets for threats, including storage devices containing digital assets, sensitive functions, and other assets as documented.

Models such as STRIDE provide mnemonics that can facilitate the identification of attacks that can form a threat scenario. These are as follows:

Attack	Desired property	Definition
Spoofing	Authenticity	Pretending to be something or someone other than yourself
Tampering	Integrity	Modifying something on disk, network, memory, or elsewhere
Repudiation	Accounting (audit)	Claiming that you didn't do something or were not responsible
Information disclosure	Confidentiality	Someone obtaining information they are not authorized to access
Denial of service	Availability	Exhausting resources needed to provide service
Elevation of privilege	Authorization	Allowing someone to do something they are not authorized to do

All data flows should be reviewed from an attacker to the target, and possible attacks should be considered at each step. Attacks are further explored through attack trees, with each tree representing a specific threat goal (generally one of C, I, or A).

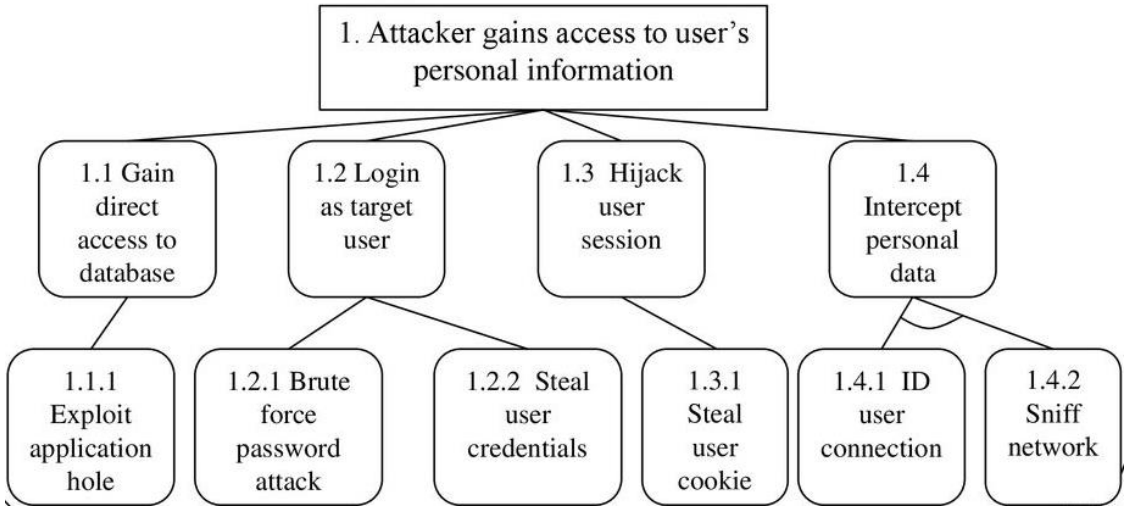

Figure 2: Example Application Attack Tree

A given branch in an attack tree forms a threat scenario. Risks posed by each threat scenario are assessed using models such DREAD for quantitative analysis or qualitative approaches based on general risk factors like likelihood and impact.

The likelihood that a threat scenario would take place is based on 3 factors: reproducibility, exploitability, and discoverability. These are measured as follows:

a) **Reproducibility:** Number of conditions that need to be true where the outcome of those conditions is not in control of the attacker or is random.
   0. **Too many factors** involved or quite a few conditions need to take place.
   1. **More than one** factor is required for the attack to be successful (e.g. user must be logged on + not be security aware)
   2. **Only one factor** is required for the attack to be successful (e.g. user must be logged on)
   3. **No factors** are required to be true that are random or not under control of the attacker.

b) **Exploitability:** this is about the required skills and expertise.
   0. **Next to impossible.** Even with direct knowledge of the vulnerability we do not see a viable path for exploitation (Only Neo could hack this)
   1. **Advanced techniques required**, custom tooling/moderate skills (DEFCON Presenter). Only exploitable by **authenticated** users.
   2. **Advanced techniques required**, custom tooling/moderate skills (Average Bug Bounty Participant). Exploitable **unauthenticated** users.
   3. **Trivial** - just a web browser or basic/publicly available tools (Script Kiddie)

a) **Discoverability**

0. **Very hard or impossible** to discover even given access to source code and privilege access to running systems.
1. **Inside knowledge or access to application internals** are required to discover the issue.
2. **Advanced tools and techniques** are required to discover the issue.
3. **Details of faults like this are already in the public domain** and can be easily discovered using a search engine. Weakness can be easily discovered by most computer users. The information is visible in the web browser address bar or in a form.

Step 5: Countermeasures and Risk Mitigation

Addressing vulnerabilities involves the implementation of appropriate countermeasures that can lower the risk in the most effective way. Controls should be identified to address each of the vulnerabilities that make up an attack chain (aka threat scenario).

After ranking threat scenarios according to their risk levels in the previous step, it becomes possible to prioritize them for mitigation based on risk. When deciding what controls to implement, those that impose the lowest effort and address the threat scenario as close to the target (or root of the attack tree) should be considered. The most effective way to mitigate a threat on the attack tree is to mitigate it as close to the root as possible. Although this is theoretically sound, it is not usually possible to simply mitigate a threat without other implications.

Options for risk management include:

- Acceptance: acknowledging the risk as tolerable.
- Elimination: removing elements that introduce vulnerabilities.
- Mitigation: implementing measures to lessen the risk's likelihood or impact.

Developer-centric Functional Threat Modeling

Developer-centric functional threat modeling is a critical approach in the secure development lifecycle that shifts the traditional perspective of threat identification and mitigation towards a more functional viewpoint. This approach emphasizes understanding and anticipating how an application can be misused or abused, directly involving the development team in the security conversation. By integrating abuse cases and "evil user stories" into the threat modeling process, developers are encouraged to think like attackers, considering not just how features should work but how they could be exploited.

In Agile or DevOps environments, threat modeling, including identifying abuse cases or "evil user stories," is a practice that is gaining traction. The goal of these activities is to anticipate and design against potential security threats by thinking from the perspective of an attacker. This approach helps in the early identification of security vulnerabilities that might be exploited and allows teams to implement countermeasures proactively.

Abuse Cases

These are scenarios that describe how the functionalities of a system can be misused or abused. They are essentially the opposite of use cases, which are intended to describe how a system should be used to achieve a positive outcome. Abuse cases help in understanding the system from an attacker's perspective. These are structured descriptions that outline how an application's features could be misused or abused. They are typically more formal and detailed, providing a comprehensive view of potential security threats from the perspective of an attacker. Abuse cases focus on identifying and understanding the negative actions that could be performed on a system, allowing developers and security teams to design and implement appropriate security controls to mitigate these risks.

Example: Account Takeover via Password Reset Functionality

- **Description**: An attacker exploits the password reset functionality to take over another user's account. The attacker uses social engineering or other methods to obtain or guess the email address associated with a user account. They then initiate a password reset request and intercept or fraudulently obtain the password reset link, allowing them to reset the user's password and gain unauthorized access to the account.
- **Mitigation**: Implement multi-factor authentication for password resets, requiring users to verify their identity through an additional channel (e.g., a code sent to a mobile phone) before allowing a password change. Use secure, token-based mechanisms for password reset links, and ensure that these tokens expire within a short timeframe.

Evil User Stories

Similar to abuse cases, evil user stories are a way to capture a requirement from the point of view of an attacker, rather than a legitimate user. They follow the user story format common in Agile development (e.g., "As an attacker, I want to inject SQL commands into input fields so that I can access the database and extract confidential data."). An easy way to do this is to add a comma (,) followed by “but” to the user story to describe the malicious actions that could be performed, aka “the evil story”.

Example: "As user, I want to be able to upload files that are required for my work**, but** as a malicious user, I want to exploit the application's file upload feature to upload and execute a malicious script, allowing me to gain unauthorized access to the server."

- **Narrative**: The attacker identifies a file upload feature in the web application intended for uploading profile pictures or documents. They craft a malicious script disguised as a

legitimate file type (e.g., an image) and upload it through the feature. The application fails to properly validate or sanitize the uploaded file, executing the malicious script and compromising the server.
- **Mitigation**: Strictly validate file types on both client and server sides, allow only specific, safe file extensions, and perform server-side checks to ensure that the uploaded files match the expected formats. Implement secure, server-side file handling practices and never execute uploaded files or scripts.

Incorporating these practices into the development process allows Agile and DevOps teams to integrate security considerations into the product design and development lifecycle early on. This is part of a broader strategy known as DevSecOps, which aims to build a culture and environment where building, testing, and releasing software can happen rapidly, frequently, and more securely by integrating security measures and testing throughout the development lifecycle, rather than as a final step.

Incorporating abuse cases and evil user stories into threat modeling enables developers to proactively identify and address security vulnerabilities within the design and implementation phases of software development. This proactive approach fosters a culture of security awareness and responsibility among developers, encouraging them to consider security implications throughout the development process. By understanding the mindset of potential attackers and the tactics they might use, developers can design and implement more robust security measures, ultimately enhancing the resilience of the application against malicious activities.

Top Attack Vectors for Threat Scenario Brainstorming

To assist individuals in the brainstorming process and in devising potential threat scenarios, it's crucial to consider a broad spectrum of attack vectors. These vectors represent various methods by which attackers could exploit vulnerabilities in both authenticated and unauthenticated contexts. Understanding the top attack vectors can help in identifying and mitigating potential threats to the system.

Fuzzing
Testing the application with invalid, unexpected, or random data to uncover vulnerabilities.

SQL Injection (SQLi)
Exploiting vulnerabilities to execute malicious SQL commands within a database system.

Denial of Service (DoS)
Overloading the system resources to disrupt service to legitimate users.

Creating Fake Users
Registering fake user accounts to bombard the system.

Cross-Site Scripting (XSS)
Injecting malicious scripts into content viewed by other users to steal data or impersonate users.

Insecure Direct Object References (IDOR)
Accessing unauthorized data by manipulating input values that reference objects directly.

Password Reset
Exploiting password reset functionalities to gain unauthorized access to user accounts.

Server-Side Request Forgery (SSRF)
Tricking the server into making requests to unintended locations or services.

Spear Phishing
Targeted phishing attacks designed to deceive specific individuals into divulging sensitive information.

Privilege Escalation (Users)
Exploiting vulnerabilities to gain higher-level permissions than originally assigned.

File Upload (Direct Shell Upload)
Uploading malicious files to gain unauthorized access or execute arbitrary code on the server.

XML External Entity (XXE)
Exploiting XML processors to execute unauthorized commands or access files.

Remote File Inclusion/Local File Inclusion (RFI/LFI)
Exploiting vulnerabilities to include files from a remote or local file system.

Multi-Factor Authentication (MFA) Bypass
Circumventing multi-factor authentication mechanisms to gain unauthorized access.

Verification Email Bypass
Exploiting vulnerabilities to bypass email verification processes during account creation or modification.

Server-Side Template Injection (SSTI)
Injecting malicious templates into server-side templates to execute arbitrary code.

Directory Traversal
Exploiting insufficient security controls to access files and directories stored outside the web root folder.

HTTP Request Smuggling

Manipulating HTTP requests to bypass security controls or access unauthorized information.

Log Poisoning
Injecting malicious content into log files to exploit vulnerable log processing.

Null Injection
Inserting null bytes to manipulate the application's logic or access unauthorized resources.

OAuth-Based Attacks
Exploiting vulnerabilities in OAuth authentication processes to gain unauthorized access.

Deserialization
Exploiting insecure deserialization to execute arbitrary code or commands on the application server.

WebSocket Attacks
Exploiting vulnerabilities in WebSocket implementations to intercept or manipulate messages.

CRLF Injection/Parameter Pollution
Injecting carriage return and line feed characters to manipulate HTTP headers or exploit web applications.

Cache Deception
Tricking the application into caching sensitive information that can be accessed by unauthorized users.

Cache Poisoning
Injecting malicious content into a web cache to spread to other users.

CORS-Related Attacks
Exploiting misconfigured Cross-Origin Resource Sharing (CORS) policies to bypass access controls.

Host Header Injection
Manipulating the host header in HTTP requests to spoof websites or bypass security controls.

OS Command Injection
Executing arbitrary system commands on the server through vulnerable application inputs.

Web Cache Entanglement
Exploiting cache mechanisms to serve malicious content or steal sensitive information.

JSON Interoperability Vulnerabilities
Exploiting differences in JSON parsers to bypass input validation or security controls.

Man-in-the-Middle (MitM) Attacks
Intercepting and possibly altering the communication between two parties who believe they are directly communicating with each other.

Clickjacking
Tricking users into clicking something different from what they perceive, potentially revealing confidential information or allowing system control.

Race Condition Exploits
Taking advantage of a system's process sequence to perform malicious actions while a program is being executed.

Subdomain Takeover
Exploiting a subdomain that points to a service not in use (orphaned DNS entry) to serve malicious content.

Credential Stuffing
Using stolen account credentials to gain unauthorized access to user accounts, relying on password reuse across services.

Session Hijacking
Exploiting the web session control mechanism to steal or manipulate session tokens and take over a legitimate user's session.

Typo-Squatting and Domain Phishing
Registering domains that closely resemble legitimate ones to deceive users into visiting malicious sites or divulging sensitive information.

This comprehensive list of attack vectors serves as a foundation for identifying potential vulnerabilities within a system and formulating strategies to mitigate associated risks.