Threat Modeling Cheat Sheet

Introduction

Threat modeling is a structured approach of identifying and prioritizing potential threats to a system, and determining the value that potential mitigations would have in reducing or neutralizing those threats. This cheat sheet aims to provide guidance on how to create threat models for both existing systems or applications as well as new systems.

You do not need to be a security expert in order to implement the techniques covered in this cheat sheet. All developers, software and system designers, and architects should strive to include threat modeling in their software development life cycle. Optimally, you will create your threat models and determine which mitigations are needed during an early stage of the development of a new system, application, or feature. Assessing potential threats during the design phase of your project can save significant resources that might be needed to refactor the project to include risk mitigations during a later phase of the project.

When you produce a threat model, you will:

- Document how data flows through a system to identify where the system might be attacked.
- Document as many potential threats to the system as possible.
- Document security controls that may be put in place to reduce the likelihood or impact of a potential threat.

Note that throughout the document, the terms "systems" and "applications" are used interchangeably. The principles in the document apply equally to designing and building systems such as network infrastructures or server clusters as they do to designing or developing desktop, mobile, or web applications.

Threat Modeling Terminology

You should be familiar with the following terms that will be used throughout this cheat sheet.

A *threat agent* is an individual or group that is capable of carrying out a particular threat. It is fundamental to identify who would want to exploit the assets of a company, how they might use them against the company, and if they would be capable of doing so. Some threats require more expertise or resources, and thus raise the level of threat actor needed. For example, if a threat requires hundreds of thousands of dollars of computing power to implement, it is likely that only organized corporate, criminal, or government actors would be valid threat actors for such a threat. However, with the rise of cloud computing and the prevalence of attack software on the internet, other threats may be easy to implement with relatively little skill and few resources.

Impact is a measure of the potential damage caused by a particular threat. Impact and damage can take a variety of forms. A threat may result in damage to physical assets, or may result in obvious financial loss. Indirect loss may also result from an attack, and needs to be considered as part of the impact. For example, if your company's website were defaced this could cause damage to your company's reputation, which may in turn cause a loss of business because of the loss of confidence by your users. Depending on the business you are in, attacks that expose user information could potentially result in a physical threat of harm or loss of life to your users, greatly raising the impact of threats that would allow such exposure.

Likelihood is a measure of the possibility of a threat being carried out. A variety of factors can impact the likelihood of a threat being carried out, including how difficult the implementation of the threat is, and how rewarding it would be to the attacker. For example, if a threat required a skilled threat actor with tens of thousands of dollars of computing resources to implement, and the only reward was that they were able to gain access to information that is already public in some other form, the likelihood is low. However, if the threat is relatively easy to accomplish, or if the attacker were to gain valuable information from which they could profit, the likelihood may be higher.

Controls are safeguards or countermeasures that you put in place in order to avoid, detect, counteract, or minimize potential threats against your information, systems, or other assets.

Preventions are controls that may completely prevent a particular attack from being possible. For example, if you identify a threat that your users' personal information may be identified by certain application logging, and you decide to completely remove that logging, you have prevented that particular threat.

Mitigations are controls that are put in place to reduce either the likelihood or the impact of a threat, while not necessarily completely preventing it. For example, if you store your user's passwords as hashes in a database, two users who have the same password will have the same hash. Thus, if an attacker has access to the hashed passwords and is able to determine the password associated with one hash, he is easily able to find all the other users who share the same password simply by looking for the same hash. However, if you add salts to each user's password, the cost of this particular attack is greatly increased, as the attacker must crack each password individual. An increase in cost reduces the likelihood, and thus has *mitigated* the attack.

A *data flow diagram* is a depiction of how information flows through your system. It shows each place that data is input into or output from each process or subsystem. It includes anywhere that data is stored in the system, either temporarily or long-term.

A *trust boundary* (in the context of threat modeling) is a location on the data flow diagram where data changes its level of trust. Any place where data is passed between two processes is typically a trust boundary. If your application reads a file from disk, there's a trust boundary between the application and the file because outside processes and users can modify the data in the file. If your application makes a call to a remote process, or a remote process makes calls to your application, that's a trust boundary. If you read data from a database, there's typically a trust boundary because other processes can modify the data in the database. Any place you accept user input in any form is always a trust boundary.

In addition to the above terminologies, it is important to be familiar with the key threat modeling principles defined in the Threat Modeling Manifesto project. Those principles are considered throughout the following steps in this cheat sheet.

Getting Started

Define Business Objectives

Before starting the threat modeling process it is important to identify business objectives of the applications you are assessing, and to identify security and compliance requirements that may be necessary due to business or government regulation. Having these objectives and requirements in mind before the threat assessment begins will help you to evaluate the impact of any threat you find during the risk analysis process.

Identify application design

Early in the threat modeling process, you will need to draw a data flow diagram of the entire system that is being assessed, including its trust boundaries. Thus, understanding the design of the application is key to performing threat modeling. Even if you are very familiar with the application design, you may identify additional data flows and trust boundaries throughout the threat modeling process.

A thorough understanding of how the system is designed will also help you assess the likelihood and potential impact of any particular threat that you identify.

When you are assessing an existing system that has existing design documentation, spend time reviewing that documentation. The documentation may be out of date, requiring you to gather new information to update the documentation. Or, there may be not documentation at all, requiring you to create the design documents.

In the optimal case, you are performing your assessment during the design phase of the project, and the design documentation will be up-to-date and available. In any event, this cheat sheet outlines steps you can take to create design documents if they are needed.

Create design documents

There are many ways to generate design documents; the **4+1** view model is one of the matured approaches to building your design document.

Reference to **4+1** view model of architecture here.

Please note that the **4+1** is comprehensive, you may use any other design model during this phase.

The following subsections show the details about **4+1** approach and how this could help in the threat modeling process:

Logical View

Create a logical map of the Target of Evaluation.

Audience: Designers.

Area: Functional Requirements: describes the design's object model.

Related Artifacts: Design model

Implementation View

Audience: Programmers.

Area: Software components: describes the layers and subsystems of the application.

Related Artifacts: Implementation model, components

Please refer to the image in the appendix section for sample design for the implementation view.

Process View

Audience: Integrators.

Area: Non-functional requirements: describes the design's concurrency and synchronization aspects.

Related Artifacts: (no specific artifact).

Deployment View

Create a physical map of the Target of Evaluation

Audience: Deployment managers.

Area: Topology: describes the mapping of the software onto the hardware and shows the system's distributed aspects.

Related Artifacts: Deployment model.

Use-Case View

Audience: All the stakeholders of the system, including the end users.

Area: describes the set of scenarios and/or use cases that represent some significant, central functionality of the system.

Related Artifacts: Use-Case Model, Use-Case documents

Decompose and Model the System

Gain an understanding of how the system works to perform a threat model, it is important to understand how the system works and interacts with its ecosystem. To start with creating a high-level information flow diagram, like the following:

1. Identify the trusted boundaries of your system/application/module/ecosystem that you may want to start off with.
2. Add actors – internal and external
3. Define internal trusted boundaries. These can be the different security zones that have been designed
4. Relook at the actors you have identified in #2 for consistency
5. Add information flows
6. Identify the information elements and their classification as per your information classification policy
7. Where possible add assets to the identified information flows.

Define and Evaluate your Assets

Assets involved in the information flow should be defined and evaluated according to their value of confidentiality, integrity and availability.

Consider Data in transit and Data at rest

Data protection in transit is the protection of this data while it’s travelling from network to network or being transferred from a local storage device to a cloud storage device – wherever data is moving, effective data protection measures for in-transit data are critical as data is often considered less secure while in motion.

While data at rest is sometimes considered to be less vulnerable than data in transit, attackers often find data at rest a more valuable target than data in motion.

The risk profile for data in transit or data at rest depends on the security measures that are in place to secure data in either state. Protecting sensitive data both in transit and at rest is imperative for modern enterprises as attackers find increasingly innovative ways to compromise systems and steal data.

Create an information flow diagram

Whiteboard Your Architecture

It is important to whiteboard system architecture by showing the major constraints and decisions in order to frame and start conversations. The value is actually twofold. If the architecture cannot be white-boarded, then it suggests that it is not well understood. If a clear and concise whiteboard diagram can be provided, others will understand it and it will be easier to communicate details.

Manage to present your DFD in the context of MVC

In this step, Data Flow Diagram should be divided in the context of Model, View, Controller (MVC).

Use tools to draw your diagram

If you don’t like to manually draw your DFD; there are several tools available that could be used:

OWASP THREAT DRAGON

The OWASP Threat Dragon project is a cross platform tool that runs on Linux, macOS and Windows 10. Threat Dragon (TD) is used to create threat model diagrams and to record possible threats and decide on their mitigations using STRIDE methodology. TD is both a web application and a desktop application; refer to the project's GitHub repository for the latest release.

POIROT

The Poirot tool isolates and diagnoses defects through fault modeling and simulation. Along with a carefully selected partitioning strategy, functional and sequential test pattern applications show success with circuits having a high degree of observability.

MS TMT

The Microsoft Threat Modeling Tool (TMT) helps find threats in the design phase of software projects. It is one of the longest lived threat modeling tools, having been introduced as Microsoft SDL in 2008, and is actively supported; version 7.3 was released March 2020. It runs only on Windows 10 Anniversary Update or later, and so is difficult to use on macOS or Linux.

OWASP PYTM

Pytm is a Python library to help you describe your system in terms of objects and attributes, able to generate a DFD in Graphviz (dot) format, a sequence diagram in plantuml format, and a list of threats (out of CAPEC and other threat libraries) to the system in a templated format. As of 2023 it is under active development. No Python knowledge is necessary for its use - if you can define objects and use .attribute notation, you should be able to use it.

Define Data Flow over your DFD

Define Data Flows over the organization Data Flow Diagram.

Define Trust Boundaries

Define any distinct boundaries (External boundaries and Internal boundaries) within which a system trusts all sub-systems (including data).

Define applications user roles and trust levels

Define access rights that the application will grant to external entities and internal entities.

Highlight Authorization per user role over the DFD

Highlight Authorization per user role, for example, defining app users' role, admins' role, anonymous visitors' role...etc.

Define Application Entry points

Define the interfaces through which potential attackers can interact with the application or supply them with data.

Identify Threat Agents

Define all possible threats

Identify Possible Attackers threat agents that could exist within the Target of Evaluation. Use Means, Motive, and Opportunities to understand Threats posed by Attackers. Then associate threat agents with system components they can directly interact with.

Work on minimizing the number of threat agents by:

- Treating them as equivalent classes.
- Considering the attacker's motivation when evaluating likelihood.
- Consider insider Threats

The user of this cheat can depend on the following list of risks and threat libraries sources to define the possible threats an application might be facing:

1. Risks with OWASP Top 10.
2. Testing Procedure with OWASP ASVS.
3. Risks with SANS Top 25.
4. Microsoft STRIDE.
5. Continuous Threat Modeling CTM.

Map Threat agents to application Entry points

Map threat agents to the application entry point, whether it is a login process, a registration process or whatever it might be and consider insider Threats.

Draw attack vectors and attacks tree

During this phase conduct the following activities:

- Draw attack vectors and attacks tree.
- Identify Use Cases/Abuse Cases.
- Re-Define attack vectors to consider multi-step attacks.

Mapping Abuse Cases to Use Cases

This is a very important step that can help identifying application logical threats. List of all possible abuse cases should be developed for each application use case. Being familiar with the types of application logical attack is an important during the mapping process. You can refer to OWASP Testing Guide 4.0: Business Logic Testing and OWASP ASVS for more details.

Re-Define attack vectors

In most cases after defining the attack vectors, the compromised user role could lead to further attacks into the application. For example, assuming that an internet banking user credentials could be compromised, the user of this cheat sheet has to then redefine the attack vectors that could result from compromising the user’s credentials and so on.

Write your Threat traceability matrix

Define the Impact and Probability for each threat

Enumerate Attacks posed by the most dangerous attacker in designated areas of the logical and physical maps of the target of evaluation.

Assume the attacker has a zero-day because he does. In this methodology, we assume compromise; because a zero-day will exist or already does exist (even if we don't know about it). This is about what can be done by skilled attackers, with much more time, money, motive and opportunity that we have.

Use risk management methodology to determine the risk behind the threat

Create risks in risk log for every identified threat or attack to any assets. A risk assessment methodology is followed in order to identify the risk level for each vulnerability and hence for each server.

Here we will highlight two risk methodology that could be used:

DREAD

DREAD, is about evaluating each existing vulnerability using a mathematical formula to retrieve the vulnerability’s corresponding risk. The **DREAD** formula is divided into 5 main categories:

- **D**amage - how bad would an attack be?
- **R**eproducibility - how easy it is to reproduce the attack?
- **E**xploitability - how much work is it to launch the attack?
- **A**ffected users - how many people will be impacted?
- **D**iscoverability - how easy it is to discover the threat?

DREAD formula is:

Risk Value = (Damage + Affected users) x (Reproducibility + Exploitability + Discoverability).

Then the risk level is determined using defined thresholds below.

PASTA

PASTA, Attack Simulation & Threat Analysis (PASTA) is a complete methodology to perform application threat modeling. PASTA introduces a risk-centric methodology aimed at applying security countermeasures that are commensurate to the possible impact that could be sustained from defined threat models, vulnerabilities, weaknesses, and attack patterns.

PASTA introduces a complete risk analysis and evaluation procedures that you can follow to evaluate the risk for each of the identified threat. The main difference in using PASTA Approach is that you should evaluate the impact early on in the analysis phase instead of addressing the impact at the step of evaluating the risk.

The idea behind addressing the impact earlier in PASTA approach is that the audience that knows impact knows the consequences on a product or use case failures more than participants in the threat analysis phase.

Application security risk assessments are not enough because they are very binary and leverage a control framework basis for denoting risks. It is recommended to contextually look at threats impacts, probability and effectiveness of countermeasures that may be present.

R = (TVP*I) / Countermeasures

For more details about PASTA.

Rank Risks

Using risk matrix rank risks from most severe to least severe based on Means, Motive & Opportunity. Below is a sample risk matrix table, depending on your risk approach you can define different risk ranking matrix:

- Risk Value: 01 to 12 → Risk Level: **Notice**
- Risk Value: 13 to 18 → Risk Level: **Low**
- Risk Value: 19 to 36 → Risk Level: **Medium**
- Risk Value: 37 to 54 → Risk Level: **High**

Determine countermeasures and mitigation

Identify risk owners and agree on risk mitigation with risk owners and stakeholders. Provide the needed controls in forms of code upgrades and configuration updates to reduce risks to acceptable levels.

Identify risk owners

For the assessors: After defining and analyzing the risks, the assessor should be working on the mitigation plan by firstly identifying risk owners which is the personnel that is responsible for mitigating the risk. i.e. one of the information security team or the development team.

For the designers or the architects: they should assign the risk mitigation to the development team to consider it while building the application.

Agree on risk mitigation with risk owners and stakeholders

After identifying the risk owners, it is important to review the mitigation controls for each of the identified risks. Some controls might be inapplicable, you should propose other mitigation controls or discuss with the risk owners the possible compensation controls.

Build your risk treatment strategy

- **Reduce:** building controls if the form of code upgrades, confirming a specific design for the application or building a specific configuration during the deployment phase to make sure that application risk is reduced.
- **Transfer:** For a specific component in the application the risk can be transferred to an outsourced third party to develop that component and making sure that the third party is doing the right testing for the component; or during the deployment phase, outsourcing a third party to do the deployment and transferring that risk to that third party.
- **Avoid:** an example of avoiding the risk is disabling a specific function in the application that is the source for that risk.
- **Accept:** if the risk is within acceptable criteria set earlier, in that case, the designer risk owner can accept that risk.

For the assessor, this is considered as the last step in the assessment process. The following steps should be conducted by the risk owner, however, the assessor shall engage in 6.5 (Testing risk treatment) to verify the remediation.

Select appropriate controls to mitigate the risk

Selecting one of the controls to reduce the risk, either by upgrading the code, or building a specific configuration during the deployment phase and so on.

Test risk treatment to verify remediation

Mitigation controls will not vanish the risk completely, rather, it would just reduce the risk. In this case, the user of this cheat sheet should measure the value of the risk after applying the mitigation controls. The value of the risk should be reduced to the acceptable criteria set earlier.

Reduce risk in risk log for verified treated risk

After applying the mitigation and measuring the new risk value, the user of this cheat sheet should update the risk log to verify that risk has been reduced.

Periodically retest risk

Application threat modeling is an ongoing process, in addition to the changes that might be happened to the application that may require re-evaluating the expected threats, it is also important to do periodic retest for the identified risks and the implemented risk treatments.

Appendix

- Mikko Kontio, Architectural manifesto: Designing software architectures, Part 5.
- Hui LM, Leung CW, Fan CK and Wong TN, "modeling agent-based systems with UML". Proceedings of the Fifth Asia Pacific Industrial Engineering and Management Systems Conference.
- References for the “4+1” View Model of Software Architecture: Kruchten, Philippe. Architectural Blueprints — The “4+1” View Model of Software Architecture. IEEE Software 12 (6), pp. 42-50.