Software Security

Threat Model

Computer Security & OS lab.

Cho, Seong-je (조성제)

sjcho at dankook.ac.kr

Fall, 2019

DANKOOK UNIVERSITY

Contents

References & Source (Credit):

- **Software Security**, Principles, Policies, and Protection, Mathias Payer, April 2019, v0.35
- **Threat Modelling - hacking the design**, Mustafa Kasmani, Senior Cyber Security Consultant, Worldpay, OWASP
- **Security Risk Analysis**, Prepared By: Ahmed Alkhamaiseh, Supervised By: Dr. Lo'a i Tawalbeh, Arab Academy for Banking & Financial Sciences (AABFS), 2007

Threat Model

Mathias Payer

Threat Modeling

- The process of enumerating and prioritizing all potential threats to a system
  - During the process, the system is evaluated from an attacker's view point.
  - Each possible entry vector is evaluated, assessed, and ranked according to the threat modeling system.

- Threat modeling evaluates questions such as:
  - What are the high value-assets in a system?
  - Which components of a system are most vulnerable?
  - What are the most relevant threats?

- Threat modeling
  - The first step consists of identifying individual components.
    - The interaction between components is best visualized by making any data flow between components.
  - Each component is then evaluated based on its exposure, capabilities, threats, and attack surface
    - For each identified threat, the necessary preconditions are mapped along with the associated risk and impact

Threat model

- A threat model defines the environment of the system and the capabilities of an attacker.
  - The model specifies the clear bounds of what an attacker can do to a system and is a precondition to reason about attacks or defenses.
  - Each identified threat can be handled through a defined mitigation or by accepting the risk if the cost of the mitigation outweighs the risk times impact.

A password-based authentication service

Let us assume we construct the threat model for the Unix "login" service

- Our application serves three use-cases:
  1) the system can authenticate a user through a username and password through a trusted communication channel,
  2) regular users can change their own password, and
  3) super users can create new users and change any password.

- The service must be privileged as arbitrary users are allowed to use some aspects of the service depending on their privilege level.
  - Our service therefore must distinguish between different types of users (administrators and regular users).
  - To allow this distinction, the service must be isolated from unauthenticated access.

- We identify the following components:
  - data storage,
  - authentication service,
  - password changing service,
  - user administration service

A password-based authentication service

- The **data storage component** is the central DB where all user accounts and passwords are stored.
  - The DB must be protected from unprivileged modification,
    - Only the administrator is allowed to change arbitrary entries while individual users are only allowed to change their entry.
  - This component relies on the authentication component to identify who is allowed to make modifications.
  - To protect against information leaks, passwords are encrypted using a salt and one-way hash function.
    - Comparing the hashed input with the stored hash allows checking equivalence of a password without having to store the plaintext (or encrypted version) of the password.

A password-based authentication service

- The authentication service takes as input a username and password pair and queries the storage component for the corresponding entry.
  - The input (login request) must come from the OS that tries to authenticate a user.
  - After carefully checking if the username and password match, the service returns the information to the OS.
  - To protect against brute-force attacks, the authentication service rate limits the number of allowed login attempts.

- The password changing service allows authenticated users to change their password, interfacing with the data storage component.
  - This component requires a successful prior authorization and must ensure that users can only change their own password but not passwords of other users.

- The administrator is also allowed to add, modify, or delete arbitrary user accounts.

A password-based authentication service

- Such an authentication system faces threats from several directions.
- an incomplete list of possible threats:
  - Implementation flaw in the authentication service allowing *either* a user (authenticated or unauthenticated) to authenticate as another user *or* privileged user without supplying the correct password.
  - Implementation flaw in privileged user management which allows an unauthenticated or unprivileged user to modify arbitrary data entries in the data storage.
  - Information leakage of the password from the data storage, allowing an offline password cracker to probe a large amount of passwords
    - Originally, the **/etc/passwd** file stored all user names, ids, and hashed passwords. This world readable file was used during authentication and to check user ids. Attackers brute forced the hashed passwords to escalate privileges. As a mitigation, Unix systems moved to a split system where the hashed password is stored in **/etc/shadow** (along with an id) and all other information remains in the publicly readable **/etc/passwd**.
  - A brute force attack against the login service can probe different passwords in the bounds of the rate limit.
  - The underlying data storage can be compromised through another privileged program overwriting the file, data corruption, or external privileged modification.

Threat Model

Mustafa Kasmani

What is Threat Modelling ?

- Threat modelling is a process by which potential threats can be identified, enumerated, and prioritized – all from a hypothetical attacker's point of view.

- The purpose of threat modelling is to provide defenders with a systematic analysis of the probable attacker's profile, the most likely attack vectors, and the assets most desired by an attacker.

- Threat modelling answers the questions "Where are the high-value assets?" "Where am I most vulnerable to attack?" "What are the most relevant threats?" "Is there an attack vector that might go unnoticed?"

— Wikipedia – (https://en.wikipedia.org/wiki/Threat_model)

4 key questions

What are you building ?

- Model system —> DFD’s, sequence flows, API contracts, etc.
  - An API contract is the documentation of the API

What can go wrong ?

- Identify threats —> STRIDE threat analysis

What should be done about it ?

- Address threats —> Risk analysis

Is the threat analysis correct ?

- Validate analysis —> Testing of controls

Why should it be done ?

- Analyze the system from an attackers point of view, threat actors & motives, and enumerate assets to protect.
- Find flaws in the design and remediate when easiest & cheapest to do so.
- Create a common understanding of the system design amongst the architects, designers, developers, testers & security folk.
- Culture over Process over Tools: Security Maturity & Worldpay experiences

Risk Assessment

- The more perspectives you get into your threat model means better protection can be designed to the system.
  - Perspective: (문제 해결을 위한 사고에서의) 균형감, 관점, 시각, (멀리 바라보이는) 전망
- Certain features can become vulnerabilities when used by people with malicious intent.
- Balance between security -vs- usability -vs- cost -vs- other competing resources (opportunity cost).
- Build up library of patterns for which risks are known, understood & accepted by the stakeholders.
- Avoid technical debt being built up through better understanding prior to new features being added
  - Technical debt (also known as design debt or code debt) is a concept in software development that reflects the implied cost of additional rework caused by choosing an easy (limited) solution now instead of using a better approach that would take longer.

Who should be involved ?

- Architects, Designers, Developers, Testers, Security, + Anyone who has an interest in it:
  - Different perspectives - business fraud (operational processes / external entities), not just technical threats
    - **Software architect**: a software developer expert who makes high-level design choices and dictates technical standards, including software coding standards, tools, and platforms.

- Security Champions in the team: Link between Development & Security:
  - scale AppSec capabilities, understand the system, maintain risk log, point of contact.
    - AppSec = Application Security

Use-cases

As a security architect,

- I want to do a threat model of …
- So that I can design effective security controls mitigate the threats identified in the threat model.
    - A **security architect** is the individual who is responsible for maintaining the security of a company's computer system. They must think like a hacker would, because they must anticipate all of the moves and tactics that hackers will use to try and gain unauthorized access to the computer system.

As a security tester,

- I want to create a library of security tests for …
- So that I can validate that the security controls in place are mitigating the threats identified in the threat model.

When ?, How ? & In reality

- **When should it be done ?**
  - As early as possible !
  - Influence direction, technology choice, system design
  - Iterative - can re-visit once further details are known
  - "The best time to plant an oak tree was 20 years ago. The next best time is now." — wise words
- **How …?**
  - STRIDE - Microsoft Methodology (c.1999)
  - PASTA - (Process for Attack Simulation and Threat Analysis)
  - VAST - (Visual Agile and Simple Threat Modelling)
- **In reality**
  - use a methodology for structure,
  - But focus on how to find good threats, rather than the merits of one approach over another
    - each has its own strengths & weaknesses
  - appropriate to what is being built, who is building it (skill-set), the prevalent risk appetite & culture

What are you building ?

- Model the system - (appropriate level of detail)
- Trust boundaries -vs- Attack surface
  - The **attack surface** of a software environment is the sum of the different points (the "attack vectors") where an unauthorized user (the "attacker") can try to enter data to or extract data from an environment.
    - Examples of attack vectors include user input fields, protocols, interfaces, and services.
  - The attack surface of a system is the complete set of vulnerabilities that exist within that system.
  - A **vector (malware)** is the method that this code uses to propagate itself or infect a computer.
    - Some common vectors include buffer overflows, HTMP email with JavaScript, networking protocol flaws, etc.
- Data - in transit / on disk / in memory
- Actors - benign / malicious, internal / external, employees, suppliers / customers / partners / etc.
- Assets - physical, logical, configuration, code, intellectual property, API contract (e.g. Swagger spec)
  - **Swagger**™ is a project used to describe and document RESTful APIs.
    - The Swagger specification defines a set of files required to describe such an API. These files can then be used by the Swagger-UI project to display the API and Swagger-Codegen to generate clients in various languages.

Model your system

- Data-Flow Diagrams
- Sequence Interaction Diagrams
- API contracts / Swagger definitions

- Keep It Simple - easy to understand
- Complexity is the enemy of Security

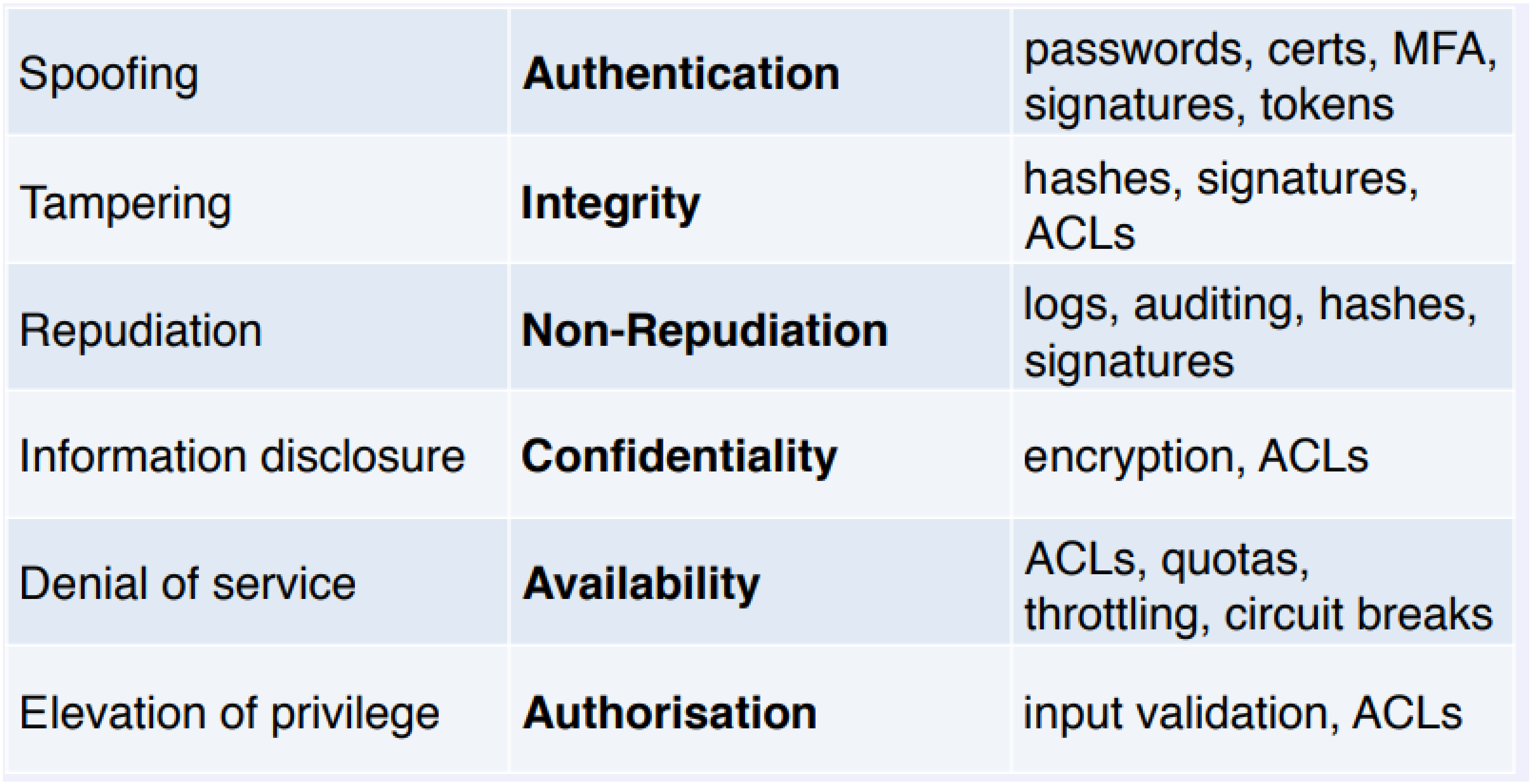
What can go wrong?

- Map attack surface
- Actors -vs- Motives
- STRIDE threat analysis
- Security Risk analysis (= Risk assessment)
  - It is essential in ensuring that controls and expenditure are fully commensurate with the risks to which the organization is exposed.
  - Security in any system should be commensurate with its risks. However, the process to determine which security controls are appropriate and cost effective, is quite often a complex and sometimes a subjective matter. One of the prime functions of security risk analysis is to put this process onto a more objective basis.
    - Commensurate: ~ (with sth) (격식) (크기·중요도·자질 등에) 어울리는[상응하는]
- Controls testing
  - Tests of controls are carried out to evaluate the operating effectiveness of the internal control policies and procedures. The auditor must decide on the nature, timing and extent of tests of control
  - A test of controls is an audit procedure to test the effectiveness of a control used by a client entity to prevent or detect material misstatements.

STRIDE threat analysis

- **Spoofing** - pretending to be someone / something else
- **Tampering** - modifying something that should not be modified
- **Repudiation** - denial of something that was done (true or not)
- **Information disclosure** - divulge information that should not be divulged, a breach of confidentiality
- **Denial of service** - prevent a system or service from being available or fulfilling its purpose
- **Elevation of privilege** - executing something without being allowed to do so

What should be done?



Spoofing	**Authentication**	passwords, certs, MFA, signatures, tokens
Tampering	**Integrity**	hashes, signatures, ACLs
Repudiation	**Non-Repudiation**	logs, auditing, hashes, signatures
Information disclosure	**Confidentiality**	encryption, ACLs
Denial of service	**Availability**	ACLs, quotas, throttling, circuit breaks
Elevation of privilege	**Authorisation**	input validation, ACLs

Risk Analysis

Risk

- Unauthorized or accidental disclosure
- Unauthorized or accidental modification
- Unavailability of facilities / services
- Destruction of assets

Risk Impact

- Monetary losses
- Loss of personal privacy
- Loss of commercial confidentiality
- Legal actions
- Public embarrassment
- Danger to personal safety

Risk Control Strategy

- Risk prevention
- Reduction of impact
- Reduction of likelihood
- Early detection
- Recovery
- Risk transfer

Qualitative Risk Analysis

- This is by far the most widely used approach to risk analysis. Probability data is not required and only estimated potential loss is used.

- Most qualitative risk analysis methodologies make use of a number of interrelated elements:
  - THREATS
  - VULNERABILITIES
  - CONTROLS

Qualitative Risk Analysis

- THREATS
  - These are things that can go wrong or that can 'attack' the system.
  - Examples might include fire or fraud. Threats are ever present for every system.
- VULNERABILITIES
  - These make a system more prone to attack by a threat or make an attack more likely to have some success or impact.
  - For example, for fire a vulnerability would be the presence of inflammable materials (e.g. paper).
- CONTROLS
  - These are the countermeasures for vulnerabilities. There are four types:
    - Deterrent controls reduce the likelihood of a deliberate attack
    - Preventative controls protect vulnerabilities and make an attack unsuccessful or reduce its impact
    - Corrective controls reduce the effect of an attack
    - Detective controls discover attacks and trigger preventative or corrective controls.

Controls testing

- Scoping assessments, targeted testing
  - Understand the system - testers get involved earlier on in the design.
  - Later tests are more targeted in approach, validation of controls rather than find new issues
  - Security built in right from the outset rather than being bolted on at the end - saves time & money !

Summary

- Threat model
- Further reading
    - ‘Threat Modeling: Designing for Security - Adam Shostack, (Wiley, 2014)