Security Assessment

Alessandro Bocci

name.surname@unipi.it

Advanced Software Engineering (Lab)

25/11/2025

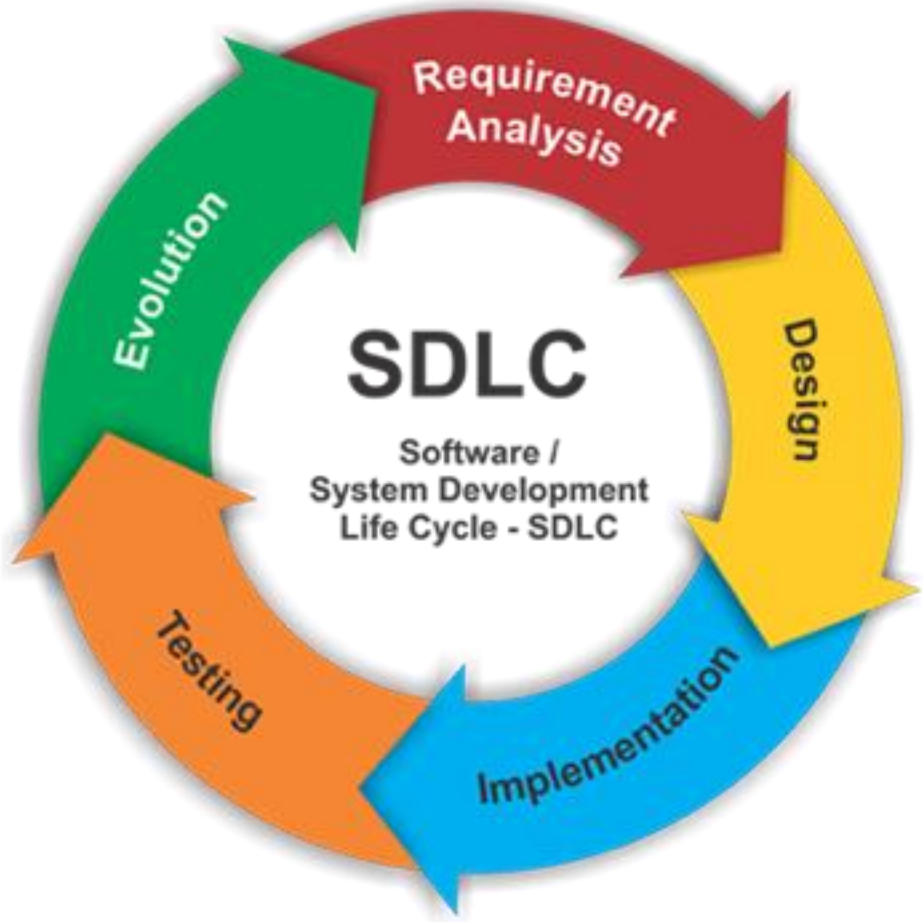
Security in the software life cycle

- Secure software development should be addressed in all the steps of the software life cycle.
- Security, as well, is an iterative process to improve the software and to react to updates of requirements and implementation.
- Ideally, you can start to use security techniques from the first iteration of your software. You can start in later iterations by doing an initial software assessment.

Software Security Assessment

Software lifecycle:

1. Requirement phase
2. Design phase
3. Implementation phase
4. Testing phase
5. Deployment and execution
6. (Review phase and go to 1)



Software Security Assessment

Software lifecycle:

1. **Requirement phase**
2. Design phase
3. Implementation phase
4. Testing phase
5. Deployment and execution
6. (Review phase and go to 1)

- Security requirements elicitation

  Security requirements need to be identified (e.g. authentication, authorization, data protection). Here, the assets to protect should be determined.

Software Security Assessment

Software lifecycle:

1. Requirement phase
2. **Design phase**
3. Implementation phase
4. Testing phase
5. Deployment and execution
6. (Review phase and go to 1)

- Security architecture review
  Identify the attack surface and evaluate which security principles to employ (e.g., defence in depth, least privilege, etc.)
- Threat modelling and risk analysis
  Determine the possible threats to the software and the derived risks.
- Attacker(s) modelling
  Model the attacker of the software by generating assumptions behind them and the actions that can be used to attack.

Software Security Assessment

Software lifecycle:

1. Requirement phase
2. Design phase
3. **Implementation phase**
4. Testing phase
5. Deployment and execution
6. (Review phase and go to 1)

- Secure practices and code reviews
  Applying best practices for security and structuring the code reviews from other people.
- Static analysis
  Scanning the source code (remember Bandit?).
- Dependency analysis
  Scanning the dependencies for Common Vulnerabilities and Exposures (remember pip-audit, Docker scout, Dependabot?)

Software Security Assessment

Software lifecycle:

1. Requirement phase
2. Design phase
3. Implementation phase
4. **Testing phase**
5. Deployment and execution
6. (Review phase and go to 1)

- Dynamic analysis
  Looking for vulns when executing the code.
- Penetration testing
  Simulate real work attacks.
- Fuzz testing
  Random input testing for unexpected behaviour.
- Configuration review
  Review of deployment scripts, container files (e.g. Dockerfiles, docker compose files), and other configuration files.

Software Security Assessment

Software lifecycle:

1. Requirement phase
2. Design phase
3. Implementation phase
4. Testing phase
5. **Deployment and execution**
6. (Review phase and go to 1)

- Monitoring

  Continuously check the behaviour of the running software (e.g., Intrusion detection systems).
- Incident response

  Analyse and patch urgent problems.

Software Security Assessment

Software lifecycle:

1. Requirement phase
2. Design phase
3. Implementation phase
4. Testing phase
5. Deployment and execution
6. **(Review phase and go to 1)**

- Collect information

  Organise and collect data about not urgent problems, feedback from users and outputs of the monitoring.
- Next cycle pianification

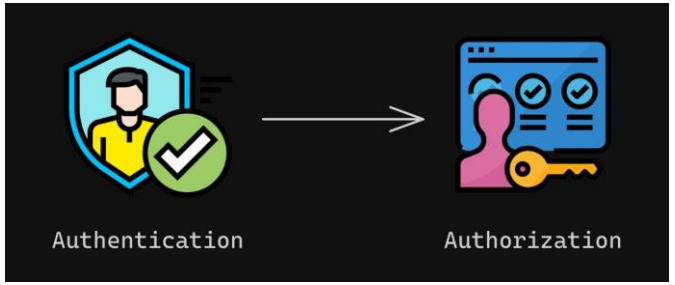
  The collected information should be prepared for the next iteration, prepare them in terms of requirements, threats, etc.

Previously...

In our three labs on security:

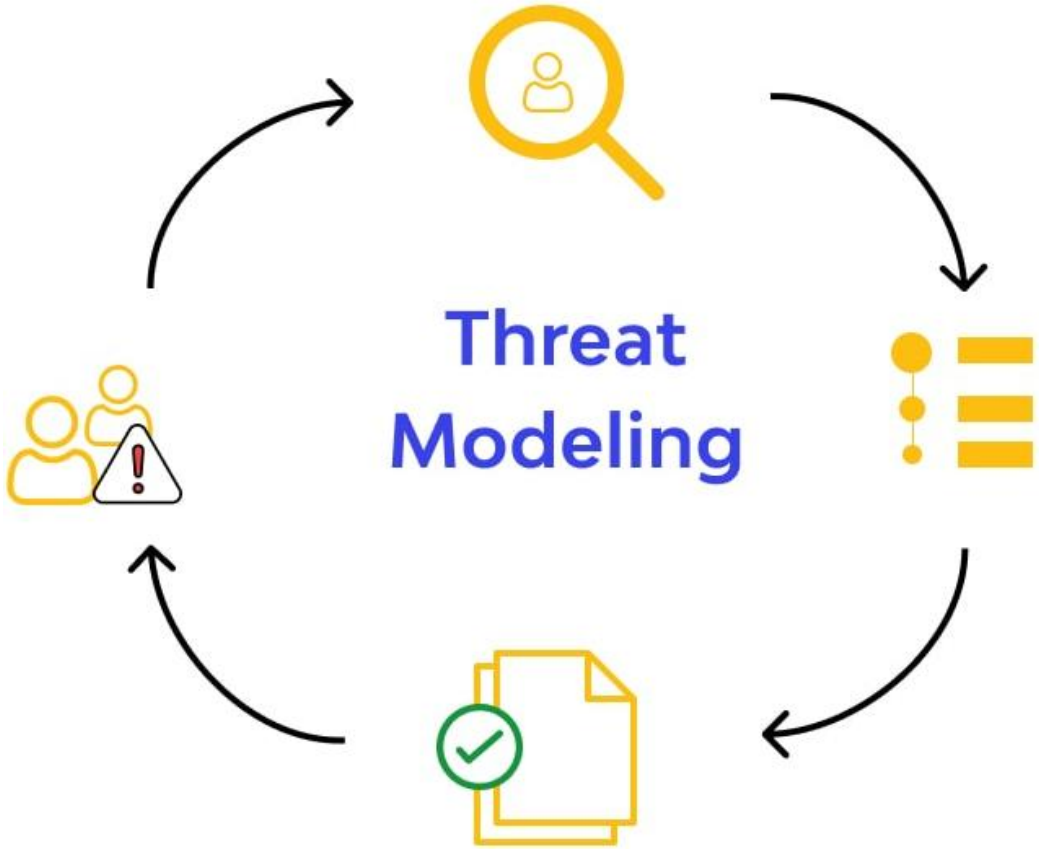
- We saw how to protect one of the most important assets: data.
- We saw static analysis of code, dependencies and containers.
- We saw how to employ authentication and authorization.

Those 3 are small parts of the security development/assessment of the software.



Today

We focus on an important part of security assessment: **threat modelling**.



Threat Modelling: What could possibly go wrong?

- Threat modeling, like risk assessment, identifies and classifies assets, their potential vulnerabilities and threats, and prioritizes each threat.
- But while risk assessments only determine whether countermeasures are needed, threat modeling goes a step further and defines those countermeasures.
- Threat modeling "thinks like an attacker," and, as a result, focuses on the attacks that are the most likely to occur.

Threat modelling

- Definition and importance:

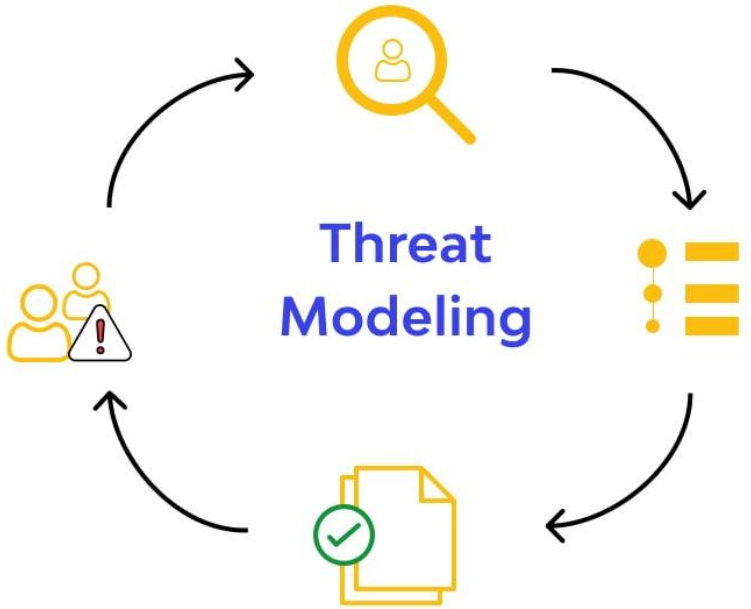
Systematic identification, analysis, and mitigation of potential threats before they are exploited.

- When to perform threat modeling:

During the design phase and whenever significant changes are made. It is an iterative process.

Threat modelling – Key phases

1. Work from a model.
2. Identify assets.
3. Identify attack surfaces.
4. Identify trust boundaries.
5. Identify threats.
6. Mitigate threats.



Threat modelling – Work from a model

Create a visual or conceptual representation of the system to understand its components, data flow, and interactions.

- Approaches:
  - Data Flow Diagrams (DFDs): Map out how data moves through the system, including processes, storage, and external entities.
  - Architectural Diagrams: Highlight the system's architecture and major components.
- Purpose: Helps uncover areas where security issues might arise, such as unprotected data flows or exposed endpoints.

Example: For a shopping website, draw a DFD showing how user credentials are sent, stored, and verified.

Threat modelling – Identify assets

Identify key elements within the system that need protection. Assets are anything of value, including data, infrastructure, and users.

- Steps:
  1. Catalog assets such as sensitive data (e.g., personal information, payment details), APIs, and system components.
  2. Rank assets based on their criticality to business operations and potential impact of a compromise.

Example:

- Critical assets: User credentials and payment information.
- Supporting assets: Logs, third-party integrations.

Threat modelling – Identify attack surfaces

Determine points where an attacker can interact with the system, such as user interfaces, APIs, and network endpoints.

- Categories:
  - External Attack Surfaces: Publicly exposed areas like web portals or APIs.
  - Internal Attack Surfaces: Privileged areas accessible by employees or trusted systems.
- Steps:
  1. Review all system entry points (e.g., login forms and file upload features).
  2. Examine backend services (e.g., databases, microservices).

Example: A mobile app's attack surfaces include the login screen, API endpoints, and the app-store deployment.

Threat modelling – Identify trust boundaries

Map the boundaries where data flows between trusted and untrusted components or environments.

- Purpose: Trust boundaries often indicate areas requiring extra security measures, like authentication, validation, or encryption.
- Steps:
  1. Use the system model to highlight where trust boundaries exist.
  2. Identify protections at each boundary (e.g., firewalls, secure APIs).

Example: Trust boundaries for an e-commerce platform include user-to-application interaction and application-to-payment gateway communication.

Threat modelling – identify threats

Enumerate potential threats using a systematic approach.

- Approach:
  - Brainstorm possible attack scenarios targeting the identified assets and attack surfaces.
  - Use frameworks (e.g., STRIDE model) to categorise threats.
  - Map threats to specific components (e.g., SQL injection targeting a database).

Example: Threats to an online shopping app include:

- Spoofing: Fake login attempts to impersonate users.
- Tampering: Modifying order data in transit.
- Denial of Service: Overloading servers with fake traffic.

Threat modelling – mitigate threats

Develop countermeasures to reduce the likelihood or impact of identified threats.

- Approaches:
  - Preventative Controls: Stop threats before they occur (e.g., input validation, access control).
  - Detective Controls: Identify when threats are occurring (e.g., logging, monitoring).
  - Corrective Controls: Minimize the impact of a successful attack (e.g., backups, incident response).
  - Prioritise threats based on risk level (using the risk assessment).

Examples:

- For SQL injection threats: Use parameterized queries and input validation.
- For denial of service: Use rate-limiting and Web Application Firewalls (WAFs).

Outcomes of Threat Modeling

- Proactive Defence: Mitigations are implemented before threats can materialise.
- Enhanced Understanding: Teams better understand system architecture and vulnerabilities.
- Prioritisation: Focuses resources on addressing the most critical threats.

STRIDE - A Threat Classification Model

Repudiation

The ability to deny actions taken, complicating accountability.

Tampering with Data

Unauthorized alteration of data to disrupt or deceive.

Information Disclosure

Unauthorized access to confidential information.

Spoofing Identity

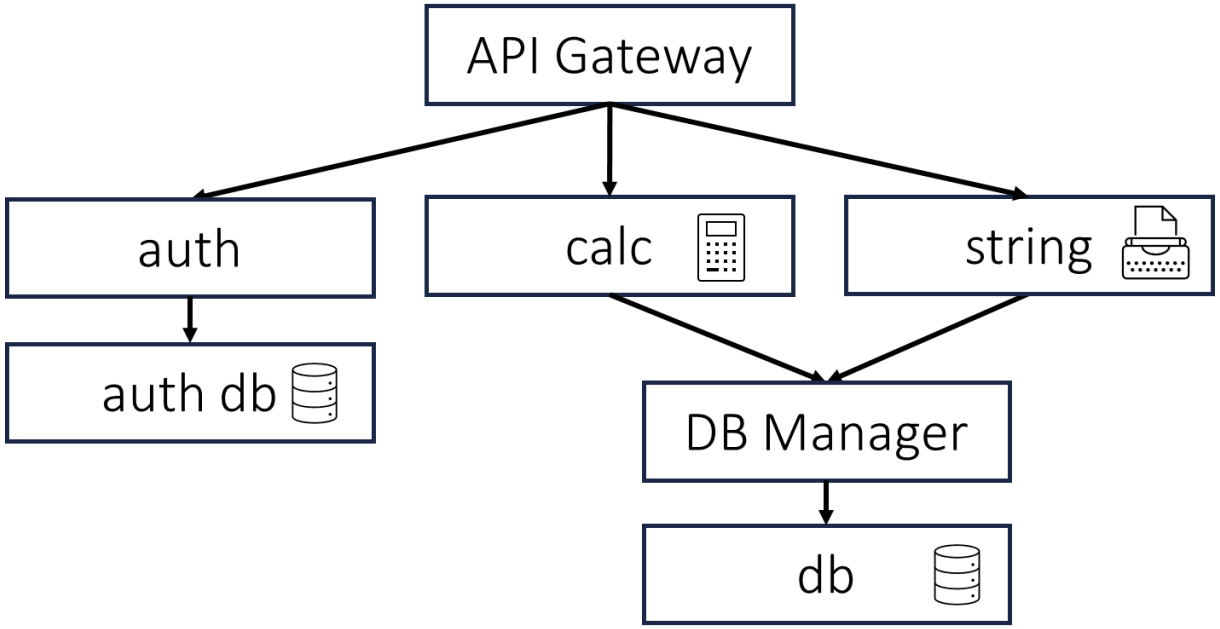
The act of impersonating another user to gain unauthorized access.

Denial of Service

Disrupting access to services or resources.

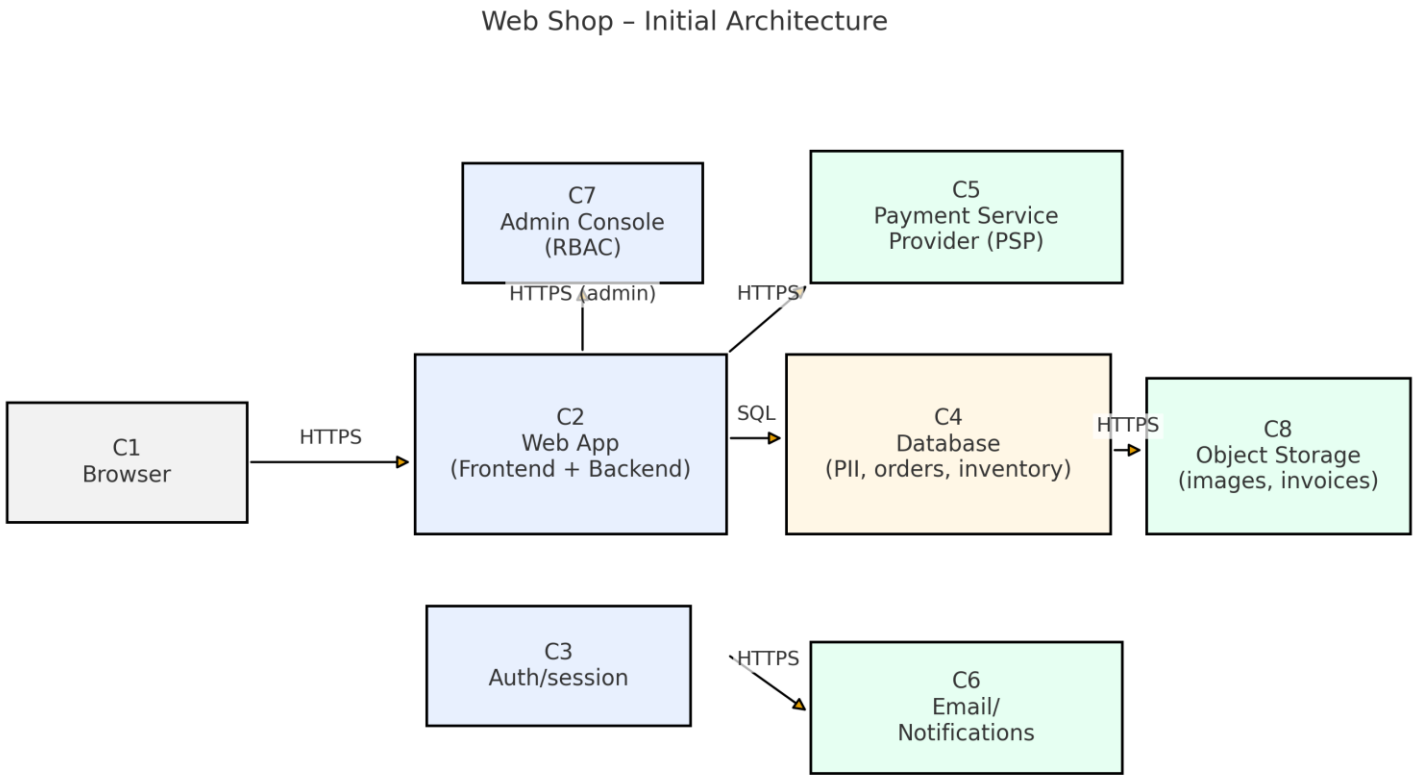
Today's Lab

- Do the Threat Model of `microase` (assuming what was added in last labs), or use your project.
- Apply the key phases starting from the architecture, the API and all the knowledge you have on the software.
- Use STRIDE to classify threats.



Threat model example

On the website, I put a PDF with an example of the threat model on a web shop with 8 components (C1—C8). Use it as reference to create yours.



Lab take away

- ❑ Understanding security assessment in the software lifecycle.
- ❑ Learn about threat modelling.
- ❑ Apply STRIDE to a known software architecture.

Project take away

- In the project you will have to write the threat model of your microservice application.