Threat Modeling

Threat Modeling

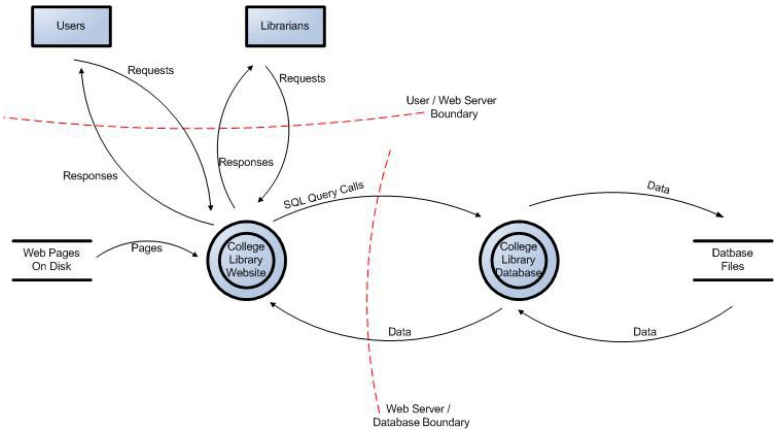
- Threat modeling analyzes system representations to highlight concerns about security and privacy characteristics.
- It examines the system from an attacker's perspective.
- Shostack's Four Question Framework:
    - **What are we working on?**
    - **What can go wrong?**
    - **What are we going to do about it?**
    - **Did we do a good enough job?**
- Principles (Threat Modeling Manifesto):
    - The best use of threat modeling is to improve the security and privacy of a system through early and frequent analysis.
    - Threat modeling must align with an organization's development practices and follow design changes in iterations that are each scoped to manageable portions of the system.
    - The outcomes of threat modeling are meaningful when they are of value to stakeholders.
    - Dialog is key to establishing the common understandings that lead to value, while documents record those understandings and enable measurement.

Threat Modeling

- Benefits:
  - Identify potential flaws at the start of the SLDC
    - …with benefits to the whole SDLC;
  - Create secure-by-design software
- It can be deployed at different levels (e.g., application vs. operational threat modeling)
- Steps:
  1. **Scope of the work** — gaining an understanding of what you're working on;
  2. **Identify Assets & Data Flows** — document key assets, such as data stores, user credentials, etc.;
  3. **Threat Finding** — identifying potential security risks that could exploit vulnerabilities in a system;
  4. **Analyse Vulnerabilities & Risk Assessment** — prioritize risks based on likelihood and impact
  5. **Countermeasures and Mitigation** — evaluate the impact and likelihood of threats; accept/eliminate/mitigate/transfer risks;
  6. **Assessment** and documentation

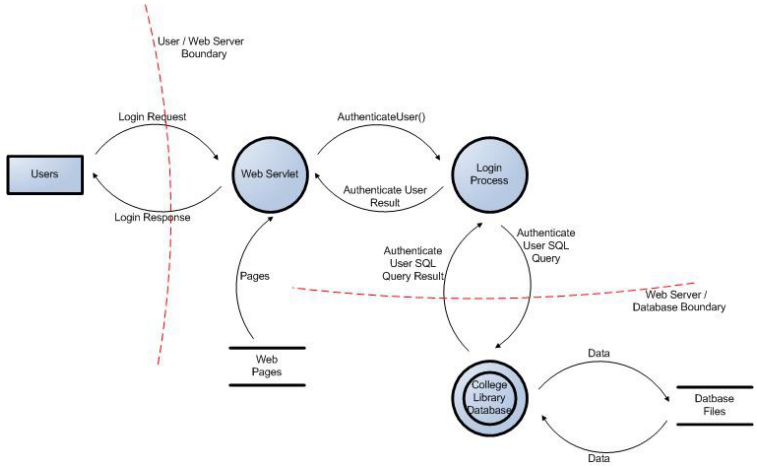
Data Flow Diagrams

- Data flow diagrams (DFDs) are frequently used to present a high-level view of the system.
- Provide a visual representation of how the application processes data.
- Show data flow pathways, highlighting the privilege or trust boundaries.



Data Flow Diagrams

- DFDs can be used to decompose the application into subsystems and lower-level subsystems.
  - The high-level DFD clarifies the scope of the application being modeled.
  - The lower-level iterations allow focusing on the specific processes involved when processing specific data



Threat Modeling frameworks

- Each methodology is unique and provides varied benefits; it is not uncommon to mix (ingredients from) distinct methodologies.
- They usually put emphasis on different aspects (e.g., organizational risk, concrete programming vulnerabilities, privacy, etc.).
- Pretty much all depend/benefit from a comprehensive knowledge base of common vulnerabilities.
- Some examples:
  - STRIDE
    - Focused on the identification of threats and vulnerabilities based on simple categorization;
    - Simple to setup and running, making it ideal for less experienced developers
    - Works well with other security frameworks like DREAD, PASTA, and NIST.
  - PASTA (Process for Attack Simulation and Threat Analysis)
    - A risk-focused approach integrating business impact.
  - LINDDUN
    - Focused on privacy threats, such as data-leaks, profiling, surveillance, or compliance with legal privacy requirements (e.g. RGPD).
  - Attack-Trees
    - Visualize attack paths to find weaknesses.

STRIDE

Type	Description	Security Control
Spoofing	Threat action aimed at accessing and use of another user's credentials, such as username and password.	**Authentication**
Tampering	Threat action intending to maliciously change or modify persistent data, such as records in a database, and the alteration of data in transit between two computers over an open network, such as the Internet.	**Integrity**
Repudiation	Threat action aimed at performing prohibited operations in a system that lacks the ability to trace the operations.	**Non-Repudiation**
Information Disclosure	Threat action intending to read a file that one was not granted access to, or to read data in transit.	**Confidentiality**
Denial of Service	Threat action attempting to deny access to valid users, such as by making a web server temporarily unavailable or unusable.	**Availability**
Elevation of Privilege	Threat action intending to gain privileged access to resources in order to gain unauthorized access to information or to compromise a system.	**Authorization**

Analyse Vulnerabilities & Risk Assessment

- Map vulnerabilities to known attack patterns (CAPEC, OWASP top-10, etc.)
- Threats should be prioritized according to their risk:
  - Focus on high-impact, high-likelihood threats first.
  - Or rely on some risk metric (e.g., CVSS)

Risk-factor	High	Medium	Low
Likelihod	Easy to exploit, widely known	Requires effort, some knowledge needed	Hard to exploit, rare occurrence
Impact	Severe system/data loss	Limited impact, minor disruptions	Minimal damage, easily recoverable

Mitigation & Countermeasures

- For each high-priority threat, implement appropriate security controls.
  - **Preventive** — such as the use of cryptography authentication, and authorization mechanisms, etc;
  - **Detective** — such as logging, monitoring, etc.
  - **Corrective** — incidence response, backups, etc.
- Example (SQL injection):
  - input validation;
  - use parametric procedures;
  - restrict database permissions.

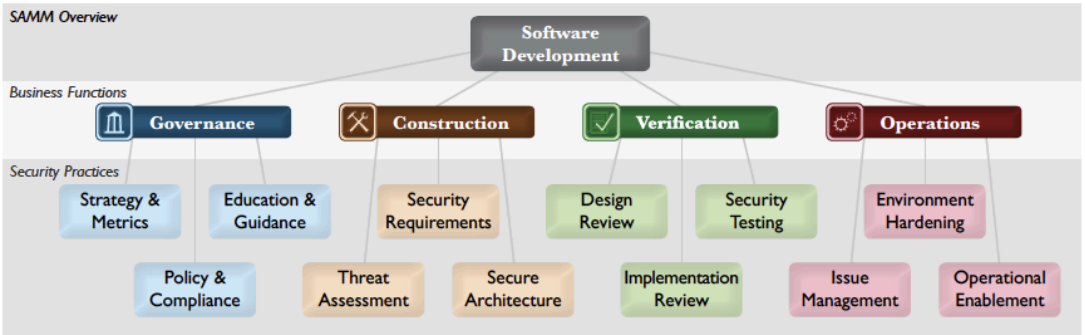
Maturity Models

Maturity Models

- We can use a **Maturity Model** to evaluate and improve secure software development practices in an organization.
  - The behavior of organizations changes slowly, so changes must be iterative, according to long-term objectives (secure software development);
  - There is no recipe that works for all organizations, so the organization must be able to choose/prioritize risks;
  - It becomes necessary to evaluate the existing software security practices in the organization and create a software security program that allows you to achieve your goals in well-defined iterations;
  - The guide to security activities to be considered in software development should provide sufficient details for people outside the security area;
  - In general, the model to be used for the development of secure software should be simple, well-defined, and measurable, as well as integrated into the organization's SDLC.
- Some Examples:
  - Microsoft SDL Optimization Model
  - Building Security In Maturity Model (BSIMM)
  - OWASP Software Assurance Maturity Model (SAMM) - https://owasp.org/www-project-samm/ -

Software Assurance Maturity Model (SAMM)

- The goal is to help organizations (whether developing, outsourcing, or acquiring software) evaluate, formulate, and implement a strategy for software security, which can be integrated into the SDLC used.
- SAMM Description:
  - Based on 12 security practices, which are grouped into 4 business functions;
  - Each security practice contains a set of activities structured in three maturity levels;
  - Activities at a lower maturity level are more easily executable and require less formalization than activities at a higher level of maturity.

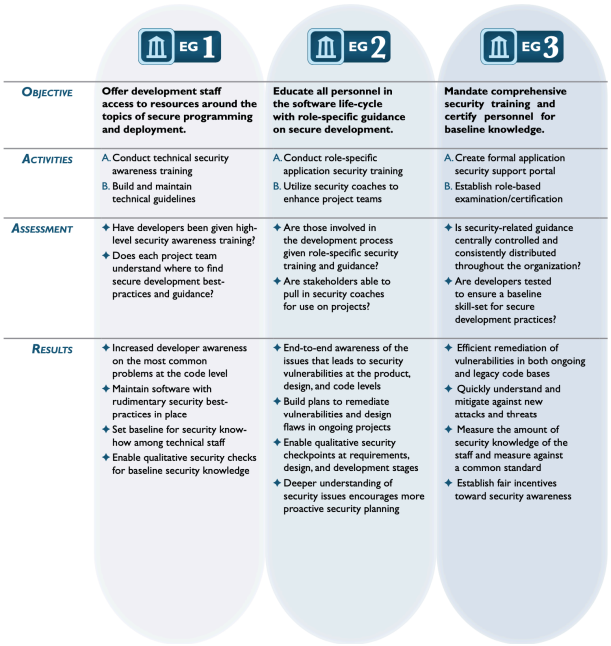


Software Assurance Maturity Model (SAMM)

- The value of SAMM is to provide a mechanism to assess where the organization is on its way toward software security and to allow an understanding of what is recommended to move to a higher level of maturity.
- SAMM does not insist that all organizations reach the highest maturity level (3) in all security practices. Each organization determines the level of maturity of each "Security Practice" that best suits its needs.
- The configuration and cycle of the SAMM maturity model are made to allow:
  - The evaluation of current software security practices,
  - The definition of the objectives that best suit the organization,
  - The formulation of a path to achieve the targeted objectives (iteratively), and
  - The execution of the specific activities of security practices in the appropriate maturity for the company, following descriptive measures.

SAMM - Maturity Level (example)

- The three maturity levels of the "Education and Guidance" security practice (included in the "Governance" business function)



	EG 1	EG 2	EG 3
Objective	**Offer development staff access to resources around the topics of secure programming and deployment.**	**Educate all personnel in the software life-cycle with role-specific guidance on secure development.**	**Mandate comprehensive security training and certify personnel for baseline knowledge.**
Activities	A. Conduct technical security awareness training B. Build and maintain technical guidelines	A. Conduct role-specific application security training B. Utilize security coaches to enhance project teams	A. Create formal application security support portal B. Establish role-based examination/certification
Assessment	✦ Have developers been given high-level security awareness training? ✦ Does each project team understand where to find secure development best-practices and guidance?	✦ Are those involved in the development process given role-specific security training and guidance? ✦ Are stakeholders able to pull in security coaches for use on projects?	✦ Is security-related guidance centrally controlled and consistently distributed throughout the organization? ✦ Are developers tested to ensure a baseline skill-set for secure development practices?
Results	✦ Increased developer awareness on the most common problems at the code level ✦ Maintain software with rudimentary security best-practices in place ✦ Set baseline for security know-how among technical staff ✦ Enable qualitative security checks for baseline security knowledge	✦ End-to-end awareness of the issues that leads to security vulnerabilities at the product, design, and code levels ✦ Build plans to remediate vulnerabilities and design flaws in ongoing projects ✦ Enable qualitative security checkpoints at requirements, design, and development stages ✦ Deeper understanding of security issues encourages more proactive security planning	✦ Efficient remediation of vulnerabilities in both ongoing and legacy code bases ✦ Quickly understand and mitigate against new attacks and threats ✦ Measure the amount of security knowledge of the staff and measure against a common standard ✦ Establish fair incentives toward security awareness

SAMM cycle

- SAMM cycle, suitable for continuous improvement (run continuously in periods of 3 to 12 months)

- **Goal**:
  - Ensure the involvement of the people necessary to determine the current level of maturity of the organization and identify the level of maturity in which it intends to be;

- **Activities**:
  - Define the scope;
  - Identify stakeholders and ensure their support for the project;
  - Inform people about the initiative and provide the information that allows them to understand what will be done.

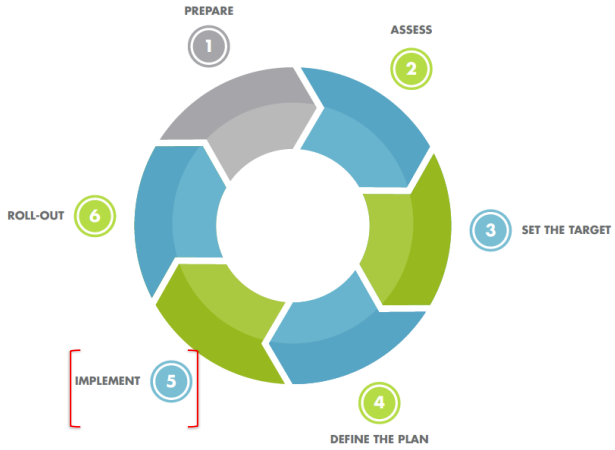
- **Goal**:
  - Identify and understand the maturity of the organization in each of the 12 security practices;

- **Activities**:
  - Evaluate current practices (using the SAMM toolbox);
  - Determine the maturity level (using the SAMM toolbox).

- **Goal**:
  - Identify a target score for each of the 12 security practices, which will serve as a guide to act on the most important activities;
- **Activities**:
  - Set the goal (using the SAMM toolbox);
  - Estimate the impact of the objective on the organization (in financial terms, if possible).

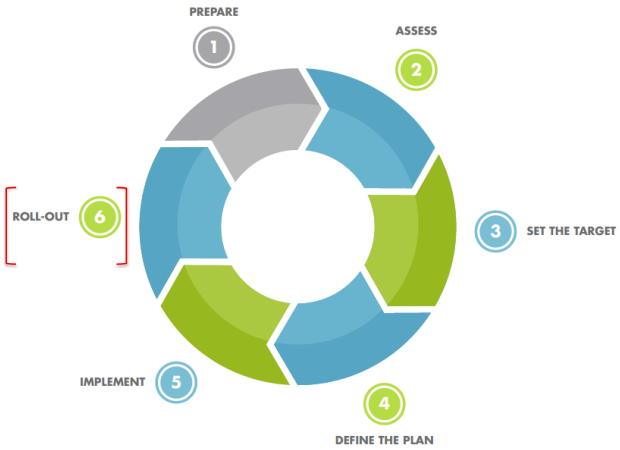
- **Goal**:
  - Develop the plan to achieve the desired maturity level (identified in phase 3);
- **Activities**:
  - Define a calendar in terms of the number and duration of phases (usually 4 to 6 phases, for 3 to 12 months);
  - Develop a plan considering the necessary effort and the possible dependence between activities.

- **Goal**:
  - Implementation of the plan to achieve the desired maturity level (defined in phase 4);



- **Activities**:
  - Plan implementation activities, considering their impact on processes, people, knowledge, and tools.

- **Goal**:
  - Ensure that improvements in the software security maturity model are effective and are being followed/used in the organization;



- **Activities**:
  - Give visibility through training sessions and communication with stakeholders.
  - Measure the adoption and effectiveness of the implemented improvements by analyzing their use and impact.