Software Security

Lecture 3

Secure Software Life Cycle

Secure software development is an ongoing process that starts with the initial design and implementation of the software. The secure software life cycle only finishes when software is retired and no longer used anywhere.

Why do we need secure software development life cycle?

It not only focuses on the functional requirements but additionally **defines security requirements** (e.g., access policies, privileges, or security guidelines). It also states **how to react if new flaws are discovered**.

Steps of SSLC,

1. Software Design
2. Software Implementation
3. Software Testing
4. Continuous Updates and Patches

Software Design

- The design phase of a software project is split into two sub phases:
  - Requirement specification : defines tangible functionality for the project, individual features, data formats, as well as interactions with the environment.
  - Design that follows specification : UML, Use Case, Context, Data Flow etc.
- From a security perspective, the software engineering requirement specification is extended with :
  - Security Specification
  - Asset Identification
  - Environmental Assessment

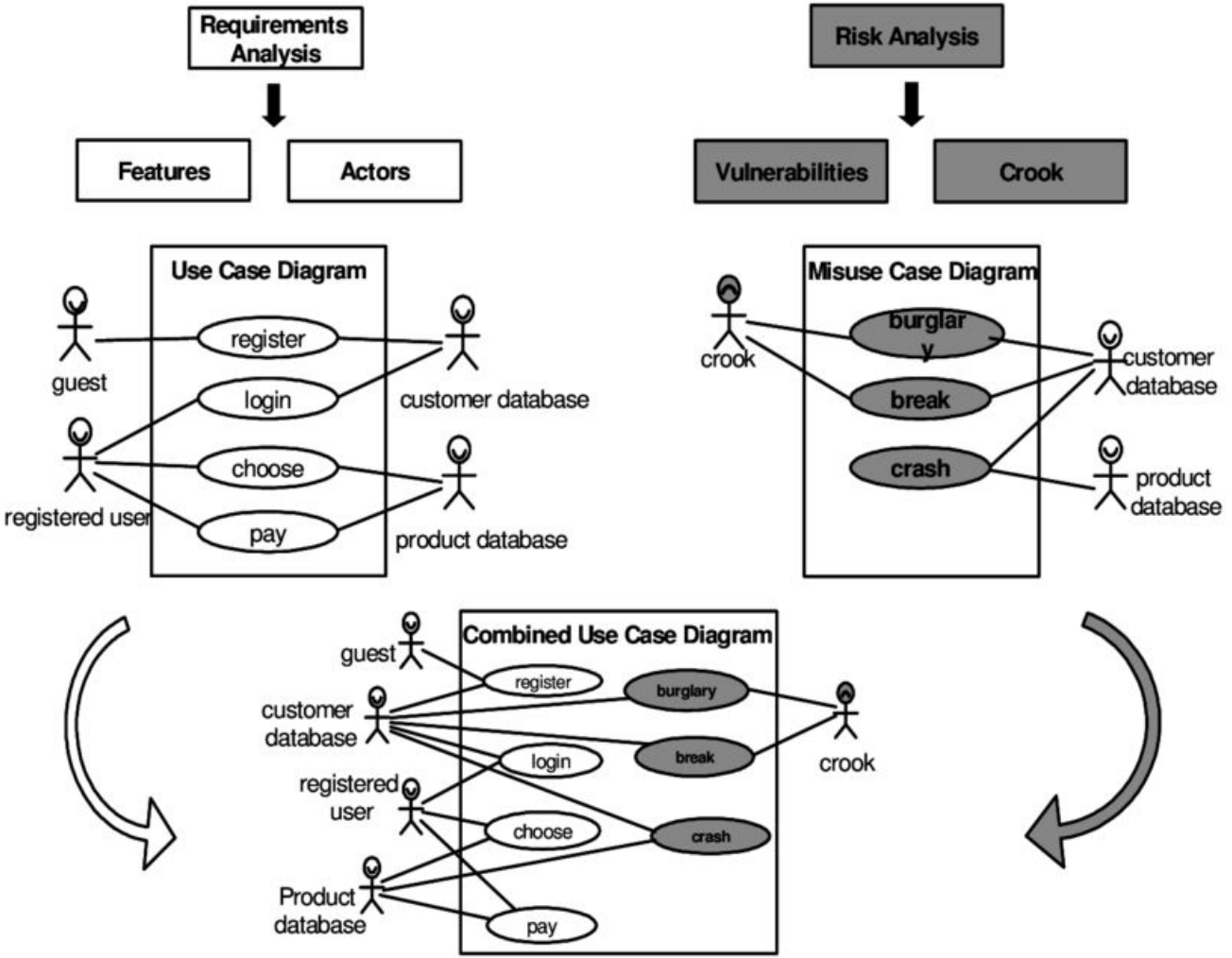
Continued

- **Security Specification** involves a threat model and risk assessment.
- The **asset specification** defines what kind of data the software system operates on and who the entities with access to the data are (e.g., including privilege levels, administrator access, and backup procedures).
- **Environmental Assessment** includes aspects of the environment which can influence the threats, e.g., a public terminal is at higher physical risk than a terminal operating in a secured facility.

An example SRS using all the above security measures.

Transitioning from the requirement specification phase to the software design phase, security aspects must be included as integral parts of the design. The final design document contains full specifications of requirements with all the security constraints. The most common diagram is Use/Misuse case Diagram.

Use\Misuse Case Diagram



Software Implementation

- Source Code should be checked for bugs and then handled through version control such as git.
    - Scenario : A developer finds a bug, and immediately create a branch where the latest code resides with that bug and shift the main branch to some previous version where the bug has not yet created. Now until developer has found a solution for that bug he will not merge these branches, keeping the main branch flawless.
- Each project should follow a strict coding standard that defines how code is indented and what features are available. The Google C++ style guide or Java style guide are great examples of such specification documents.
    - For C++, it is worthwhile to define how exceptions will be used, what modern features are available, or how memory management should be handled.
- We should maintain **formal code review process** : Before committing code to a repository, it must be checked by another person on the project to test for code guidelines, security, and performance violations. Like, **if it is correctly indented, if its has security issues and is it running within the time limit**.

Software Testing

Testing in software engineering focuses primarily on functionality and performance and it somewhat neglects security aspects. Moreover, Security testing is different from functional testing. Functional testing measures if software meets certain performance or functional criteria. **Security as an abstract property is not inherently testable**. Crashing test cases indicate some bugs but there is no guarantee that a bug will cause a security issue. **Automatic security testings are based on** some basic testing procedure. One of which is **Fuzz testing**.

Fuzz testing (or **fuzzing**) is a software testing technique that involves **feeding a program** with random, unexpected, or invalid data **to expose** potential vulnerabilities, crashes, or errors.

Additionally, a red team evaluates the system from an adversary's perspective and tries to find exploitable flaws in the design or implementation.

Continuous Updates and Patches

Software evolves and, in response to changes in the environment, will continuously expand with new features, potentially resulting in security issues.

Google Chrome leverages a partial hot update system that quickly pushes binary updates to all Google Chrome instances to protect them against attacks.

Linux distributions such as ubuntu also leverage a market-style system with an automatic software update mechanism that continuously polls the server for new updates and informs the user of new updates (e.g., through a pop up) or, if enabled, even automatically installs the security updates.

Modern Software Engineering

Agile software development is one of the modern software engineering process, where the **main focus is on functionality** which **creates a lack of written** specification or documentation. These lack of written specification engender security challenges. Following are the techniques we can utilize to integrate security measurements in our agile software development process,

- **Incorporate security constraints into user stories**. Like, a user story can be " As a user, I want my data to be encrypted with AES-256 during transmission to ensure confidentiality".
- **Incorporate Security Testing in CI/CD Pipelines**. Conduct SAST(static application security testing) or Fuzz testing before any integration or releases. SAST can catch SQL injection, cross-site scripting.
- **Security sprints** are dedicated occasional sprints entirely to security tasks, such as fixing vulnerabilities, reviewing security policies, and refining security automation.
- **Secure dependencies** means to use well-maintained and secure third-party libraries and frameworks. Regularly scan for vulnerable dependencies using tools like OWASP Dependency-Check, Snyk, or npm audit.