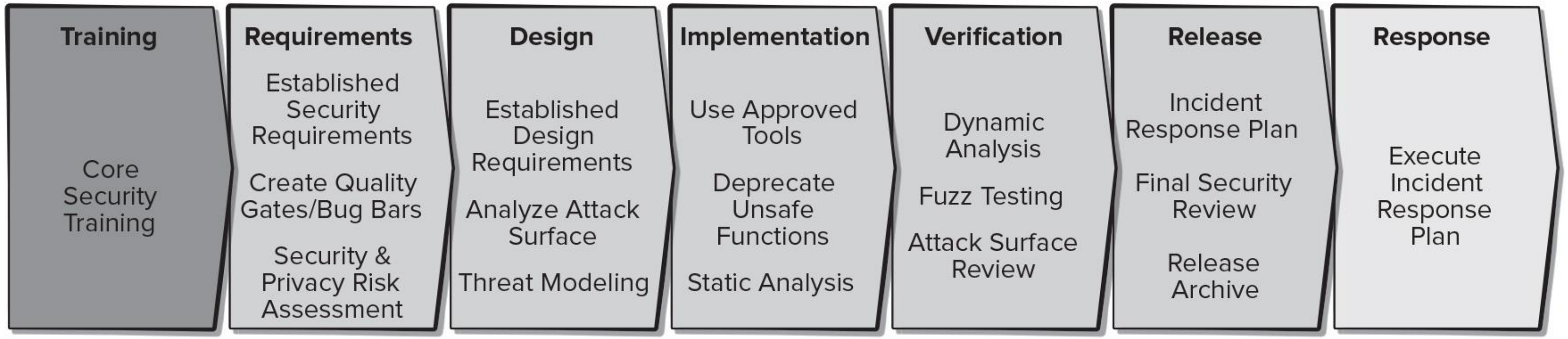
Software Engineering

Vinaya Sathyanarayana

Secure Software Development Process Model



Secure Software Development Process Model at Microsoft

Adapted from Shunn, A., et al. Strengths in Security Solutions, Software Engineering Institute, Carnegie Mellon University, 2013. Available at http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=77878.

- [Access the text alternative for slide images.]()

Microsoft Secure by Design

- **Secure architecture, design, and structure.** Developers consider security issues part of the software architectural design process.
- **Threat modeling and mitigation.** Threat models created and mitigations are present in all design and functional specifications.
- **Elimination of vulnerabilities.** This review includes the use of analysis and testing tools to eliminate classes of vulnerabilities presents in the code.
- **Improvements in security.** Less secure legacy protocols and code are deprecated, users are provided with secure alternatives consistent.

Microsoft Secure by Default

- **Least privilege.** All components run with the fewest possible permissions.
- **Defense in depth.** Components do not rely on a single threat mitigation solution that exposes users if it fails.
- **Conservative default settings.** Development team minimizes attack surface in default configuration.
- **Avoidance of risky default changes.** Applications do not make any changes that reduce computer security.
- **Less commonly used services off by default.** If fewer than 80 percent of a program's users use a feature, that feature should not be activated by default.

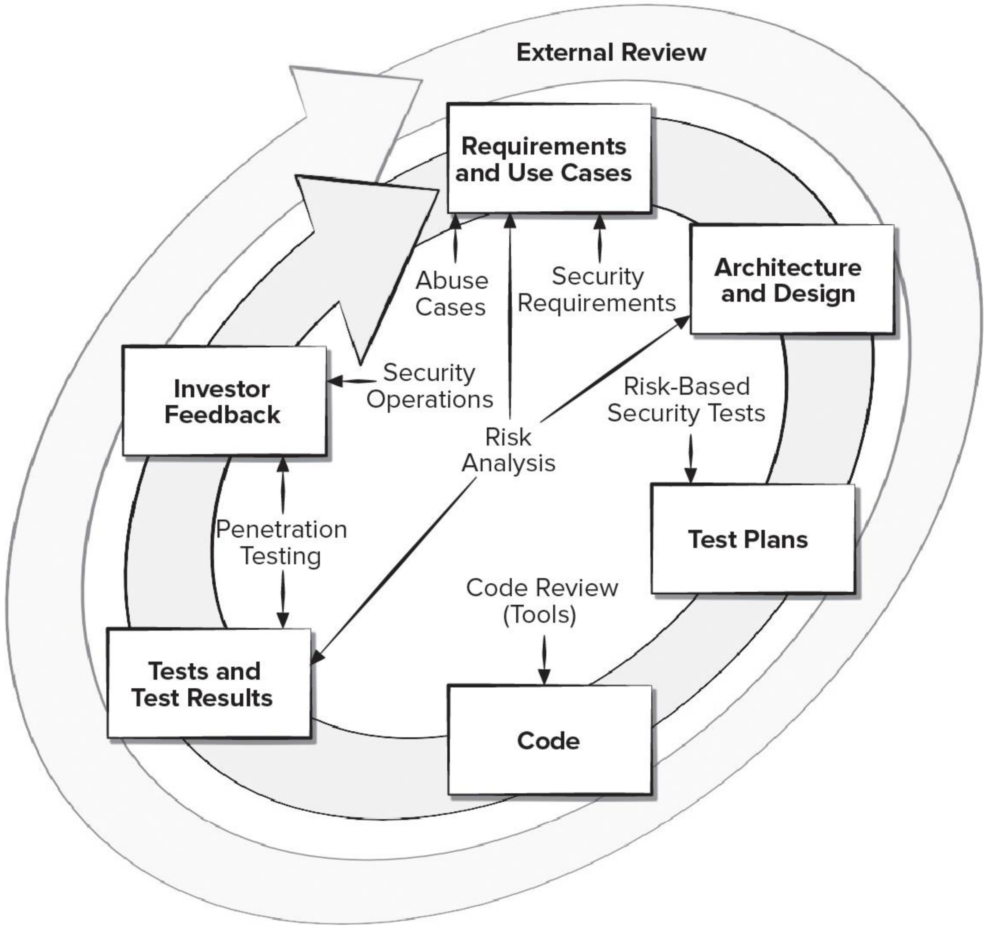
Microsoft Secure in Deployment

- **Deployment guides.** Prescriptive deployment guides outline how to deploy each feature of a program securely, including providing users with information that enables them to assess the security risk of activating non-default options.
- **Analysis and management tools.** Security analysis and management tools enable administrators to configure the optimal security level for a release.
- **Patch deployment tools.** Deployment tools aid in patch deployment.

Communications

- **Security response.** Development teams respond promptly to reports of security vulnerabilities and communicate information about security updates.
- **Community engagement.** Development teams proactively engage with users to answer questions about security vulnerabilities, security updates, or changes in the security landscape.

Software Security Touchpoints (Activities)



- Access the text alternative for slide images.

SQUARE Security Requirements Engineering [1]

- **Step 1. Agree on definitions.** Needed as a prerequisite to security requirements engineering so there is no semantic confusion.
- **Step 2. Identify assets and security goals.** Step occurs at project organizational level and needed to support software development.
- **Step 3. Develop artifacts.** Often, organizations do not have key documents needed to support requirements definition, or they may not be up to date.
- **Step 4. Perform risk assessment.** Requires an expert in risk assessment methods, support of stakeholders, and support of a security requirements engineer.

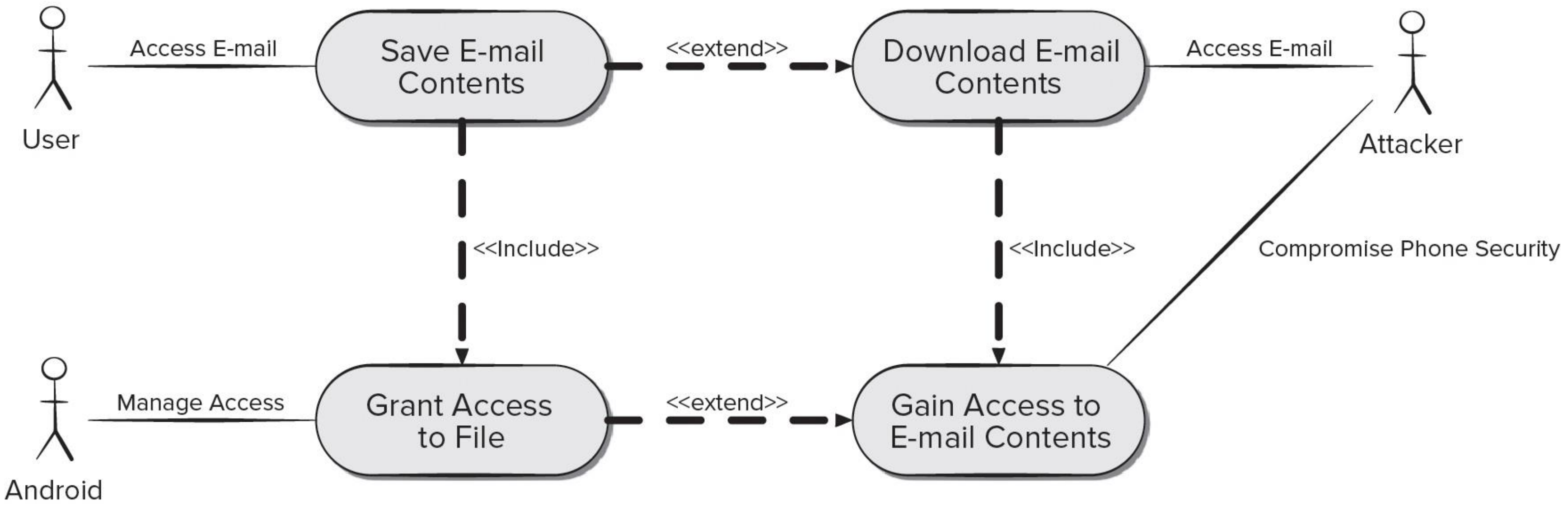
SQUARE Security Requirements Engineering 2

- **Step 5. Select elicitation technique.** This step becomes important when there are diverse stakeholders.
- **Step 6. Elicit security requirements.** This builds on the artifacts that were developed in earlier steps.
- **Step 7. Categorize requirements.** Allows security requirements engineer to identify essential requirements.
- **Step 8. Prioritize requirements.** Performs a cost-benefit analysis to determine security requirements with a high payoff relative to their cost.
- **Step 9. Requirements inspection.**

Misuse (Abuse) Cases

- A ***misuse case*** can be thought of as a use case that the attacker initiates.
- Misuse cases need to be prioritized as generated.
- Trying to answer such questions like these help developers to analyze their assumptions and allows them to fix problems up front:
- How can the system distinguish between valid and invalid input data?
- Can it tell whether a request is coming from a legitimate application or a rogue application?
- Can an insider cause a system to malfunction?

Misuse Case Example



- [Access the text alternative for slide images.]

Attack Patterns

- Attack patterns can provide some help by providing a blueprint for creating an attack.
- For example, buffer overflow is one type of security exploitation.
- Attackers trying to capitalize on a buffer overflow make use of similar steps.
- Attack patterns can document these steps (for example, timing, resources, techniques) as well as practices software developers can use to prevent or mitigate their success.
- When you’re trying to develop misuse cases, attack patterns can help.

Risk Management Framework (RMF) Steps

- **Categorize** the information system and the information processed, stored, and transmitted by that system based on an impact analysis.
- **Select** an initial set of baseline security controls for the information system based on the security categorization.
- **Implement** the security controls and describe how the controls are employed within the information system and its environment.
- **Assess** security controls to determine the extent to which they are operating to meeting system security requirements.

- **Authorize** information system operation based on a

STRIDE Threat Categories

- **Threat**
- Spoofing
- Tampering
- Repudiation
- Information disclosure
- Denial of service
- Elevation of privilege

- **Security Property**
- Authentication
- Integrity
- Nonrepudiation
- Confidentiality
- Availability
- Authorization

STRIDE Threat Modeling Steps

- Typical STRIDE implementation includes modeling a system with data flow diagrams (DFDs):
- Mapping the DFD elements to the six threat categories,
- Determining the specific threats via checklists or threat trees.
- Documenting the threats and steps for their prevention.

- In the next stage, the STRIDE user works through a checklist of specific threats that are associated with each match between a D FD element and threat category.
- Once the threats have been identified, mitigation strategies can be developed and prioritized.
- Typically, prioritization is based on cost and value considerations of implementing or not implementing a mitigation strategy,

Attack Surface

- The ***attack surface*** of an application is:

1. The sum of all paths for data/commands into and out of the application.
2. The code that protects these paths.
3. All valuable data used in the application.
4. The code that protects these data.

- ***Attack Surface Analysis*** involves mapping the parts of a system need to be reviewed and tested for security vulnerabilities with the intention of minimizing risks to the attack surface.

Secure Coding Practices [1]

1. **Validate input.** Validate input from all untrusted data sources.
2. **Heed compiler warnings.** Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code.
3. **Architect and design for security policies.** Create a software architecture and design your software to implement and enforce security policies.
4. **Keep it simple.** Keep the design as simple and as small as possible.
5. **Default deny.** Base access decisions on permission rather than exclusion.

Secure Coding Practices 2

6. **Adhere to the principle of least privilege.** Every process should execute with the least set of privileges necessary to complete the job.
7. **Sanitize data sent to other systems.** Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf (COTS) components.
8. **Practice defense in depth.** Manage risk with multiple defensive strategies.
9. **Use effective quality assurance techniques.**
10. **Adopt a secure coding standard.**

Measurement

- Measures of software quality can go a long way toward measuring software security.
- Defect and vulnerability count are useful measures.
- Not all software defects are security problems, vulnerabilities in software generally result from a defect of some kind in the requirements, architecture, or code.
- To assess software vulnerabilities and associated security issues, data must be collected data so that patterns can be analyzed over time.
- Without collecting data about software security issues, it is impossible to measure its improvement.

Security Measure Examples

- Percentage of security requirements covered by attack patterns, misuse and abuse cases, and other specified means of threat modeling and analysis (Requirements Engineering).
- Percentage of architectural and design components subject to attack surface analysis and measurement (Architecture and Design).
- Financial and/or human safety estimate of impact for each threat category (Risk).
- Number of (vetted) trusted suppliers in the supply chain by level (Trusted Dependencies).

Software Assurance Maturity Model (SAMM)

- SAMM is an open framework with the following objectives:
- Evaluate an organization’s existing software security practices.
- Build a balanced software security assurance program in well-defined iterations.
- Demonstrate concrete improvements to a security assurance program.
- Define and measure security-related activities throughout an organization.