Secure Development Lifecycle: Microsoft, TouchPoints, SAFECode (Lecture 7)

Prof. Mathy Vanhoef

DistriNet – KU Leuven – Belgium

KU LEUVEN DistriNet

Agenda

- **Microsoft SDL (new): 12 best practices**
  - Released in December 2018: see their website
  - We will skip best practices that we already fully covered
- Touchpoints: 7 best practices
- SAFECode: 8 best practices

Note: the Next slides are partly based on "Cyber Security Body of Knowledge: Secure Development Lifecycle" by Laurie Williams.

1. Provide training

- Establish training criteria
  - Topics: secure design, development, test, and privacy. Cover everything that is important in the lifecycle.
  - Example: 80% of all technical personnel must be trained
- Establish minimum training frequency
  - Attacks constantly chance, knowledge must be refreshed
  - Example: employees must attend n classes per year

2. Security requirements

Identify security requirements of the product:

- Consider security requirements of functional requirements.
  - E.g., functional requirement to allow a nurse to edit a patient record
  - Security implications: nurse needs to be logged in, transaction must be logged, some parts of patient record need to be encrypted, etc.
- Use techniques to develop security requirements
  - SQUARE, abuse cases, i*, and KAOS
- Consider legal and industry compliance standards
  - Example: HIPAA, GDPR

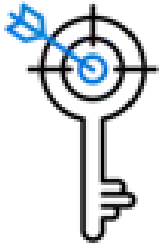
2. Security requirements: clarification

Note: the term "**security requirements**" can mean two things:

1. Security requirements of the **product's functionality**
   - This is the meaning of "security requirements" in the previous slide
   - Describes how the product/software should behave
   - E.g.: "The software must not divulge the data to unauthorized users"

2. Security requirements of the **development process**
   - This was the meaning of "security requirements" in the previous lectures
   - These don't depend on the functionality of the software
   - E.g.: banning the use of dangerous functions (such as `strcpy`)

3. Define metrics and compliance reporting



- Specify a **bug bar** for security (see previous lecture)
  - Requires a classification of what is a low, moderate, or critical security or privacy vulnerability. Can use CVSSv3 or other ratings.
- Ensure that **bug reporting tools** can track security issues and that a database can be queried for all security bugs
- Understand the regulatory standards you need to follow and what you need to report in order to be compliant
  - GDPR, Payment Card Industry (PCI), etc.

4. Perform threat modeling

- See previous lectures

5. Establish design requirements

- Architecture and design must be resistant to known threats
- Microsoft doesn’t list explicit **design principles**
- Example design principles are covered in a next lecture

6. Define and use cryptographic standards

Cryptography to secure data in transit and data at rest

- Provides confidentiality, authentication, and integrity
- **Consult experts** to define clear encryption standards
- Basic rule: use **industry-vetted** libraries and ensure they can be easily replaced if needed. E.g., use HTTPS and TLS.

See also: “Microsoft SDL Cryptographic Recommendations”

7. Manage security risk of 3rd party code

Nearly all software uses 3rd party components:

- Have an **inventory of 3rd party components** that you use
- Have a plan on **how to respond** when a vulnerability in a 3rd party component has been discovered
- Use tools to **scan for known vulnerabilities** in your 3rd party components
- **Disable unused features** of 3rd part components
- Audit new versions of 3rd part components before using them

8. Use approved tools

Create a list of approved (development) tools

- Include their associated security checks, such as compiler/linker options and warnings

Use the latest version of approved tools

- For instance, the latest compiler versions
- Take advantage of new security functionality and protections

9. Perform static analysis testing

Use tools to analyze the source code prior to compilation

- Ensure that coding policies are being followed
- These can also look for security bug patterns
- Can be integrated into the commit pipeline or into the developer environment
- Examples: Lint, Coverty,…

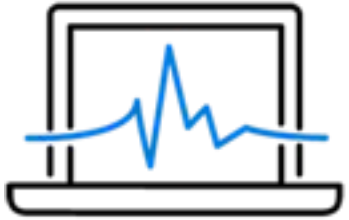
Note: must consider that they can result in false positives

10. Perform dynamic analysis testing

Run-time verification of your fully compiled software

- Precise tool depends on the product
- Example: web app scanning tools (XSS, SQL injection, etc.)
- Example: fuzzing is also a very common technique
- Can be run by the developer **or integrated into the build and deployment pipeline**

11. Perform penetration testing



Manual security testing by experts

- Testers can be internal people or can be consultants
- Objective is to uncover any vulnerability in the system
- This **simulates what an attacker might do**/try in practice
- **Pentesters** are usually called white hat (or ethical) hackers

12. Create an incident response process

Standard Incident Response Plan

- Created in coordination with your organization's Product Security Incident Response Team (PSIRT)
- Have a response policy (for public & private vulnerabilities)
- The plan should include **who to contact** in case of a security emergency in your product. E.g., security@company.com
- Define who is responsible to handle vulnerabilities
- Monitor vulnerabilities in 3rd party components

12. Create an incident response process

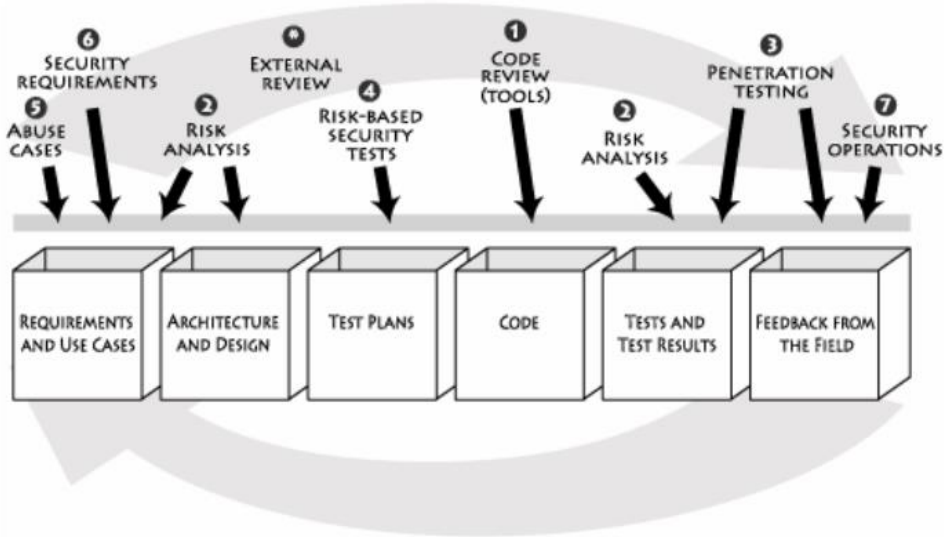
Standard Incident Response Plan

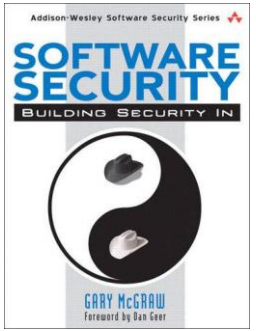
- Fix the vulnerability and **disclosure it** publicly
- Perform a **root cause analysis** to prevent similar bugs
- Crucial to rapidly deploy possible patches
- Incident response **plan should be tested** before it’s needed!

Agenda

- Microsoft SDL (new): 12 best practices
- **Touchpoints: 7 best practices**
- SAFECode: 8 best practices

Seven Security Touchpoints



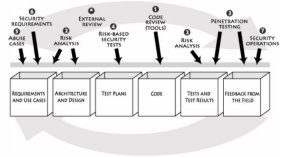


1. Code review tool
2. Risk analysis
3. Penetration testing
4. Risk-based testing
5. Abuse cases
6. Security requirements
7. Security operations

- Number associated to each touchpoint indicates their priority
- Agile: each “touchpoint” can be repeated in every iteration

Reference: “Software Security: Building Security in” by G. McGraw

Seven Security Touchpoints



1. Code review tools (already seen - static/manual analysis)
2. Risk analysis (partly already seen)
3. Penetration testing (see Microsoft best practice)
4. **Risk-based testing**
5. **Abuse cases**
6. Security requirements (see Microsoft best practice)
7. **Security operations**

2. Risk analysis = threat modeling

Detect & prevent design flaws by identifying possible attacks:

1. **Attack resistance analysis**: use checklist to identify known threats. E.g., Microsoft STRIDE-based attack analysis.
2. **Ambiguity analysis**: two analysts study threats, might discover different things. Might expose invalid assumptions.
3. **Weakness analysis**: map the assumptions being made about 3rd party components. What if those assumptions fail?

Prioritize issues/threats that result in a high risk exposure:

Risk exposure = probability of occurrence * cost of attack

4. Risk-based testing

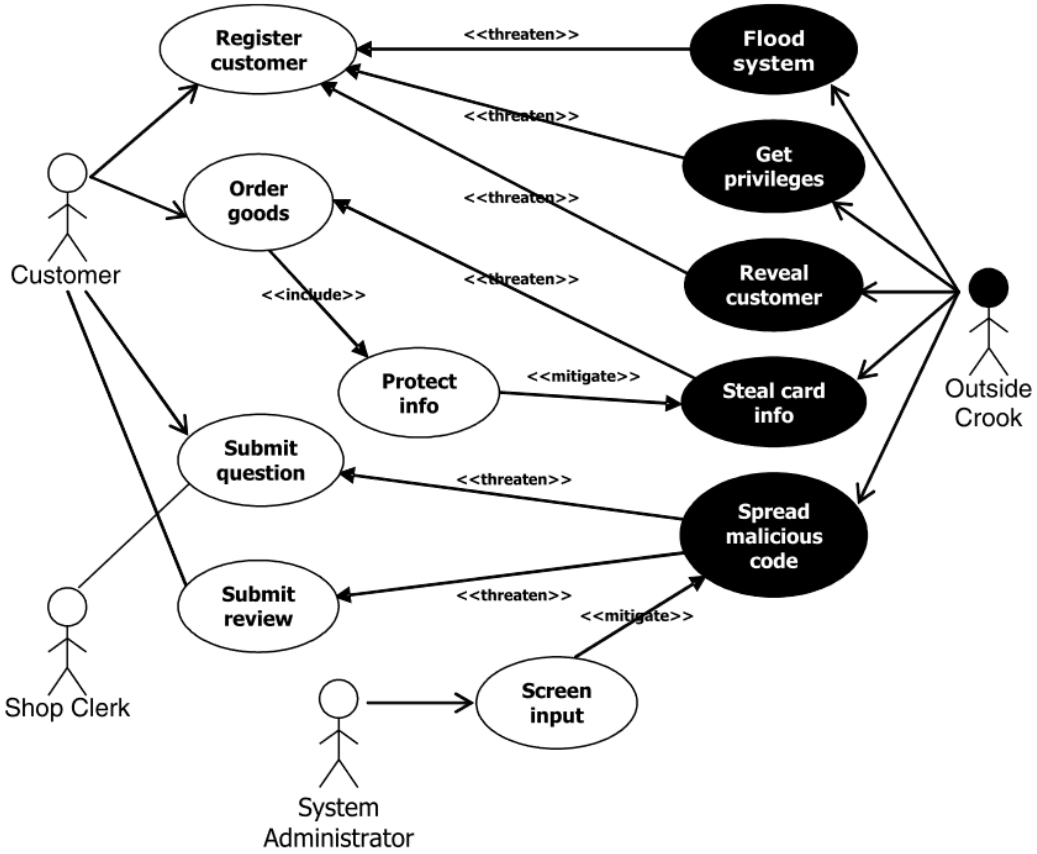
Risk-based testing resembles penetration testing, but:

- **Testing is done at the feature or component/unit level**
- Testing is done prior to system integration, that is, before the product has been completed.
- If each component has been tested successfully, then the system as a whole should (hopefully) be in reasonable shape
  - Not always a valid assumption! See KRACK attack against WPA2.

5. Abuse cases

- Abuses cases are a combination of:
  - **Threat agent**: identify who might attack the system
  - **Anti-requirements**: things that the software shouldn't do
  - **Attack model**: attack patterns that the system might be vulnerable to
- The abuse case then is combined with a mitigation method
- Abuse cases are a specific form of threat modeling
  - Instead of determining general risks, abuse cases **force you to think like an attacker**, i.e., think about specific attacks and attack patterns

5. Abuse cases: example



- Start from use cases
- Include all the bad actors
- What do they want to do?
- Tie together the use and abuse cases
- How do to stop them?

Source: Eliciting security requirements with misuse cases by Sindre and Lothe Opdahl

7. Security operations

Bridging the gap between **developers and security experts**

- How can security experts help software development?
  - Abuse cases, risk analysis, designing risk-based test scenarios
  - Code review, penetration testing during Q&A testing, etc.
- Knowledge gained from (network) **attacks and exploits is then cycled back into software development**, i.e., into the six other touchpoints
- These days: security operations = incidence response

Agenda

- Microsoft SDL (new): 12 best practices
- Touchpoints: 7 best practices
- **SAFECode: 8 best practices**

SAFECode: background

Software Assurance Forum for Excellence in Code (SAFECode)

- An organization of companies. Charter members are:

- Their goal is to produce documentation of practices to develop secure software. They have several guides.
- One of their main guides is "Fundamental Practices for Secure Software Development". It describes 8 practices.

SAFECode: 8 practices

1. Security controls (see Microsoft security requirements)
2. **Design (= design principles)**
3. Secure/defense coding practices (see next lecture)
4. Manage risk of 3rd party components (see Microsoft SDL)
5. **Testing and validation**
6. **Manage security findings**
7. Vulnerability response (see Microsoft incidence response)
8. **Planning the deployment of an SDL**

2. Design

SAFECode splits this into five points:

2.1. Secure design principles (covered in next lecture)

2.2. Threat modelling (already seen)

2.3. Develop an encryption strategy

2.4. Identity and access management

2.5. Log requirements and audit practices

2.3. Encryption strategy

Most organizations benefit from having a centralized encryption strategy that is created by experts. This strategy encompasses:

- Define **what to protect**: all internet traffic and ideally also traffic in private networks. For storing data, criteria must be defined for what type of data should be encrypted.
- Define **encryption mechanisms**:
  - For traffic in transit, e.g., recent TLS versions
  - For traffic at rest. This depends on whether the primary risk is device theft or whether application compromise must also be considered.

2.3. Encryption strategy

Most organizations benefit from having a centralized encryption strategy that is created by experts. This strategy encompasses:

- Define the **key management** solution
  - Who (person or service) can access the encryption keys?
  - What is the lifetime of keys?
  - What is the process to revoke compromised keys?
- Keep **crypto agility** in mind: crypto algorithms or libraries can have vulnerabilities. Assure that updates can be written (when needed) to switch to new crypto mechanisms or libraries.

2.4. Identity and access management

Organizations should standardize authentication and authorization mechanisms

- **User authentication**: how to sign up, type of credentials to use (including multi-factor authentication), how to restore access when credentials are lost/forgotten, etc.
- **Service authentication**: mechanism to use, where keys of servers are stored, how keys are updated, etc.
- **Authorization strategy**: see complete mediation principle, least privilege, and default deny/fail principles.

2.5. Log requirements and audit practices

- Carefully identify what needs to be logged
  - This should be determined by those who will analyze the logs
  - Only log critical information. Logging affects system resources.
- Use logging features of the operating system
  - Likely to be more secure and provide tamper protection
- Make the amount of info being logged configurable
- Don’t delete (local) logs too quickly

5. Testing and validation: automated testing

- Static analysis, dynamic analysis, fuzzing: already covered
- **Software composition analysis**: scan for open-source code in a project. This is done to evaluate security, license compliance, and code quality.
- **(Network) vulnerability scanning** : scan for known security issues (CVEs), including 3rd party software being used
- Tools to **validate configurations**: check that mitigations are enabled (ASLR, DEP, CFG) and that software is configured securely (HTTP security options, SSL configurations, etc.)

5. Testing and validation: manual testing

- Manual **quality assurance** should include verification of security features
- **Penetration testing**: already covered.

6. Manage security findings

- Assure a system exists to track vulnerabilities
- Have clear definitions of severity (low, medium, high).
- If an issue cannot be addressed, have a process in place to “accept or reject the risk”.
  - E.g., VP of Business Unit can accept/reject critical risks, Engineering Manager can accept/reject medium risks, etc.

8. Deploying the SDL

Many factors may aid or impede the adoption of the practices:

- **Culture**: does the organization respond better to corporate mandates or to a groundswell from engineers?
- **Expertise**: give sufficient training. People are less likely to adopt practices if they don't understand their importance.
- **Current lifecycle**: consider how often and when best practices must be adhered to. E.g., in agile development, not all best practices can be followed in each sprint.

8. Deploying the SDL

Many factors may aid or impede the adoption of the practices:

- **Prioritization (scope)**: prioritize the best practices and start with only a subset of them. Assure there is enough time to plan for the adoption of new practices.
- **Stakeholders**: assure key people support the new practices.
- **Compliance**: specify what is mandatory or optional.
- **Feedback**: assure there are feedback mechanisms to identify what is working and what is not.

Required reading

- SAFECode: Fundamental Practices for Secure Software Development: Essential Elements of a Secure Development Lifecycle Program

- You ***don't*** need to memorize the detailed differences between Microsoft SDL, Touchpoints, and SAFECode.
- You ***do*** need to understand *all* the best practices.

Optional extra information

- On the Secure Software Development Process: CLASP, SDL and Touchpoints Compared by De Win, Scandariato, Buyens, Grégoire, and Joosen, 2007.
  - Discusses overlap and specificity of each SDL model