Contents

Aspects of secure system development

- Developing secure systems and software
- Software development life cycle
    - Requirements
    - Design
    - Development
    - Testing
- Microsoft’s SDL
- Processes
- Feedback
- Security software testing

Secure development of systems

Aspects of secure system development

- Secure Processes
- Secure Requirements Specification
- Secure Software Development
- Secure Testing
- Continuous improvement

Microsoft’s SDL

Security development life cycle approach:

(https://www.microsoft.com/en-us/securityengineering/sdl/)

- Education
  - Provide training
  - Coaching
  - Continuous professional improvement
- Continuous process improvement
  - Feedback, lessons learned
- Accountibility
  - Traceabilty starting from requirements
  - Incident response management

The software development life cycle in the SDL

Seven phases

- Training
  - core security training
- Requirements
  - establish security requirements, assess risks,
- Design
  - attack surface analysis (& reduction), threat modelling
- Implementation
  - approved tools, static analysis
- Verification
  - dynamic analysis, fuzz testing, attack surface review
- Release
  - incident response plan, final security review
- Response
  - execute incident response plan

Requirements specification

The information security manager’s role:

- ensure that security requirements are incorporated
- ensuring that requirements relating to security needs exist
- ensure that these requirements are agreed on by all stakeholders (including budget decision stakeholders)
- pay attention to legal aspects, industrial standards, governmental regulation etc.
- benefit: good requirements elicitation and documentation **reduces** development costs

Requirements elicitation

- Good idea:
    - also introduce “misuse case diagrams” (rather than only use case diagrams in UML)
    - functions that the system should NOT allow
    - can help you define both functional and non-functional requirements
    - foundational paper by Sindre and Opdahl identifies 5 steps
        - identify critical **assets**
        - define security **goals** for these *assets*
        - identify **threats** related to these *goals*
        - Identify and analyze **risks** for these *threats*
        - Define security **requirements** for these *risks*
    - You may want to maintain a pool of misuse cases for multiple projects

Requirements in security

- Understand and identify security needs within and outside the enterprise (customer)
    - interviews, workshops, meetings
- Document each identified requirement in an unambiguous and detailed way
    - traceability
    - verify
    - validate
- Review requirements (early testing)

General challenges in requirements engineering

- Requirements are
    - always changing
    - hard to elicit
    - hard to write
    - are sometimes wrong or missing
    - often neglected
- Some areas of focus in security related requirement engineering
    - privacy of users and user groups
    - standards: do we comply to all security policies
    - is a requirement testable
    - security vs. usability/efficiency

Design

- Based upon the specified requirements
- Analyze documented requirements
    - What is the most plausible approach to deliver the desired system in a secure way
- Document the design (link design elements to requirements etc.)
    - agile: whiteboard might be sufficient
    - waterfall: standardized documents

Design principles for security

- Recommendations by IEEE (https://ieeecs-media.computer.org/media/technical-activities/CYBSI/docs/Top-10-Flaws.pdf)
  - use secure authentication mechanisms
  - separate data and control instructions (do not make them into a single string)
  - all data should be explicitly validated
  - use cryptography correctly
    - do not design your own crypto
    - do not misuse libraries
    - key management
    - randomness
  - identify sensitive data
  - understand human user behavior
  - be aware of future changes
  - integrate external components with care

Secure development

- your role as a security management
    - keep in contact with the technical team implementing the requirements
    - make sure guidelines, standards, policies are understood and kept
    - when – due to deadlines, budget cuts etc – security trade-offs are required
        - defend the security requirements
        - be somewhat flexible
        - always explain the resulting risks and related threats
- development aspects
    - defensive coding (validate data)
    - static analysis testing (walkthrough, inspection)
    - analyse risks related to third party components
    - think about CIA (C: secure communication, A: back up data, etc.)

Secure development - coding activities

- Translate the design into workable code
  - satisfying the requirements
  - meeting functional specifications
  - implement technology to meet security requirements
    - firewalls, crypto protocols, authentication etc.
  - component testing
    - security, functionality, efficiency
  - component reviews
  - white-box techniques are important
    - memory buffer overflows
    - code insertion (by internal employee)
    - backdoor

Secure development - coding guidelines

- Some example coding practices
- Open web application security project (owasp.org) lists guidelines on
  - input validation
  - output encoding
  - password management and authentication
  - session management
  - access control
  - crypto practices
  - error handling and logging
  - data protection
  - communication / database security
  - memory and file management

Secure development - common errors and defects

- Some example coding practices
- SANS institute's list of 25 errors https://www.sans.org/top25-software-errors/
- Some categories
  - Insecure interaction between components, e.g.
    - SQL injection
    - XSS
  - Risky resource management, e.g.
    - buffer overflow
    - uncontrolled format string
  - Porous defenses, e.g.
    - hard coded credentials
    - incorrect authorization

Testing components

- Input validation
- Compiler warnings
- Default deny
- Sanitize data sent across the system
- Quality assurance techniques
- Secure coding standard
- Best practice checklists

System or acceptance testing

- End-to-end test of the full system
  - waterfall: only at the end of the SDLC
  - agile: incrementally
- Test plan and design should have included security requirements
  - test security requirements from a system perspective

System or acceptance testing

- System testing
  - test in an environment identical (or very close to) the final target
- Acceptance testing
  - install in operational environment
  - check security acceptance criteria
- Should rather be black-box

Secure testing - summary

- Vulnerability testing
  - Tools for typical vulnerabilities exist: buffer overrun, SQL injection etc.
- Penetration testing
  - black box or white box
  - independent testing may help
- Static methods
  - white box methods can help you find code defects, deviation from standards
  - tools (integrated with the IDE sometimes) exist
  - ELTE has related projects :)
- Reporting
  - Use templates, enclose all information, traceability matters
- Verification (continuously)
  - Are we developing according to the requirements?

Maintenance and support

- Types of maintenance
  - Corrective: remove bugs
  - Adaptive: new environment
  - Perfective: new features required
- Handle all maintenance-related development and testing activities with the same caution as in the primary life cycle
- End-to-end regression scenarios

Some security mechanisms and testing

- The international software testing qualifications board (ISTQB) lists the following security mechanisms in their security testing syllabus
  - System hardening
  - Authorization and authentication
  - Encryption
  - Firewalls and network zones
  - Intrusion detection
  - Malware scanning
  - Data obfuscation
  - Training

System hardening

- Goal: reduce attack surface which is large by complexity
  - remove unnecessary components, libraries, accounts, applications, features, peripherals
  - use security mechanisms
  - use updates, follow coding rules, guidelines etc.
  - Testing these mechanisms:
    - reviews and audits
    - vulnerability scanner

Authorization and authentication

- Authenticate
  - who is this user? Is it really?
  - requirements vary
    - one-time pw., hardware tokens, single sign-on etc.
- Authorize
  - What is the user allowed to do?
- Tests should include
  - weak password, input filtering, unauthorized URL, etc.

Encryption

- Storage and communication
- Symmetric and asymmetric
- Testing design
  - right modes used
  - key sizes
  - avoid man-in-the-middle by validating certificates
  - configuration
  - outdated protocols
- Testing implementation
  - code reviews
  - fuzz testing
  - timing attacks, side-channel attacks

Firewalls and network zones

- Know your firewall
- Testing
  - Fuzz
  - Port scanning
  - Malformed network packets

Intrusion detection

- Intrusion detection
  - no 100% secure system
  - monitor activities at different levels
  - detect anomalies
  - negative security model (blacklist)
    - based on a list of suspicious source or behavior
  - positive security model (whitelist)
    - deviation from specified behavior / input detected
- testing of IDS
  - Input modification
  - obfuscation
  - URL encoding
  - IP fargmentation

Other mechanisms in security testing

- Malware scanning
  - if external tool is used
  - validate the tool and vendor from various perspectives
- Data obfuscation
  - goal: code obfuscation makes reverse engineering harder, copyright
  - testing
    - brute force / dictionary attacks
    - reverse engineering of byte code
  - reverse engineering always theoretically possible (debugging)

Testing with human factors in mind

- Testing for vulnerabilities of human origin
  - social engineering (ask for password over phone)
  - looking around for post-its on desks and monitors
  - password audits, brute-force attacks
  - security tester feedback

Testing for human-related problems

- Understand attackers, analyze risks
    - Question of “when”, not “if”
- Hackers will use
    - hacker databases
        - google hacker database etc.
    - tools
    - methods to destroy evidences

Reaction to hacking

- Computer forensics
  - analyze who, how and why attacked the system
  - make snapshot
  - use forensic tool
  - fix and recover
- Symptoms include
  - suspicious log entries
  - user accounts
  - unusual system behavior
  - unsuccessful logins
  - unusual network traffic

Tools in security testing

- Test basis for selecting tools
  - Security policy of the enterprise
  - Testing policy
  - Risk analysis of the developed product
  - Requirements and design decisions
- What is important
  - Interfaces to be tested
  - Protocols, standards
  - hardening

Tools in security testing - cont’d

- OWASP lists tools e.g. in www.owasp.org/index.php/Appendix_A:_Testing_Tools
- Most tools are open source
  - know the license
  - learning curve
  - update
  - support
  - future life of the tool (disappear / license change)

Evaluation of security test tools

- Types of licenses offered
- Support
- Forum / wiki
- update frequency
- manuals
- contracts

Introducing tools

- Start with pilot projects
- Assess best practice to use the tool
- Document lessons learned
- Introduce company-wide usage
- Continuously update documentation
- Gather feedback
- Incorporate tool into processes

Release and follow-up

When releasing the product:

- Perform acceptance test based on the requirements
- Have an incident response plan ready

You may be working on a continuously rolled-out system anyway...