Secure software engineering

Chapters 10: Process models

Fraunhofer

Institut
Experimentelles
Software Engineering

Fraunhofer-Platz 1
67663 Kaiserslautern
Germany

Dr. Holger Peine

Holger.Peine@iese.fraunhofer.de

Lecture at Kaiserslautern University of Applied Sciences, winter term of 2007/08

Secure Software Engineering: Process models

"Secure Software Engineering" - Overview of this course

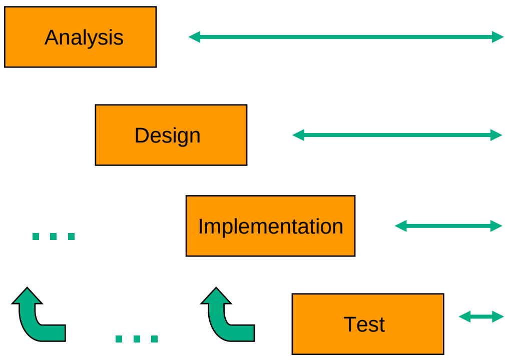
1. Introduction: IT security and software security
2. Fundamental notions and definitions of software security

- *(Crash course on web application security*
  - *Not really software engineering, but needed for many examples)*

3. Vulnerabilities and attacks (2 lessons)
4. Security in the software development process
5. Security requirements elicitation (½ lesson)
6. Threat analysis (1½ lessons)
7. Security in architecture and design (2 lessons)
8. Secure coding (2 lessons)
9. Quality assurance: Inspections, testing, static analysis (1½ lessons)
10. **Process models**
11. Usability, conclusions

Secure Software Engineering: Process models

Security should be integrated all over the development process



Analysis	static system description, security policy, threat analysis
Design	threat analysis, architectural risk analysis, patterns, design review
Implementation	secure coding guidelines, source code review
Test	risk-based security testing, final security review

Slide 3

Secure Software Engineering: Process models

10. Process models

10.1 The tension between security and other development goals

Slide 4

Security vs. other goals of software development

Security can be detrimental to other qualities ☹

- Performance
- Functionality
- Simplicity
- Usability
- Interoperability / reuse
- Low development effort

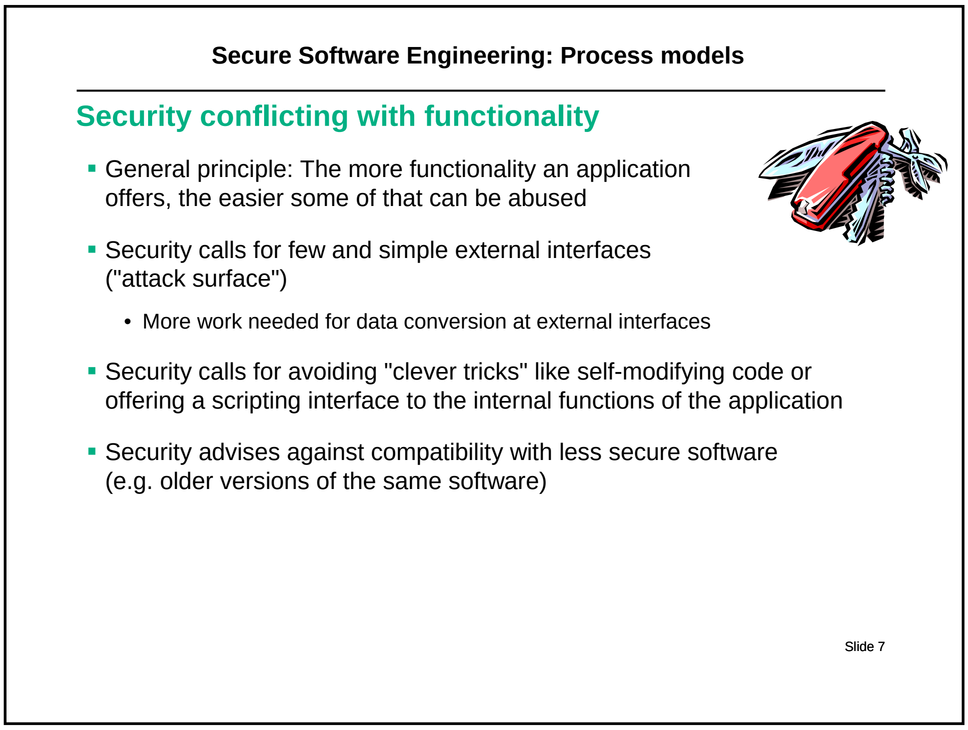
Security is often beneficial for correctness, safety and robustness ☺

Security pays off in the longer run (complete product life time) ☺

- Well, most probably so... Empirical numbers are still missing ☹

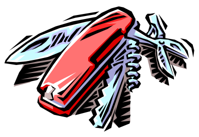
Security conflicting with performance

- Security checks take time
  - "Use several layers of defence" recommends more checks than absolutely necessary; and more checks take even more time
- Security calls for few and simple external interfaces ("attack surface")
  - More work needed for data conversion at external interfaces
- Security calls for dividing an application into mutually mistrusting, secured components
  - Inter-component invocations and data passing takes time



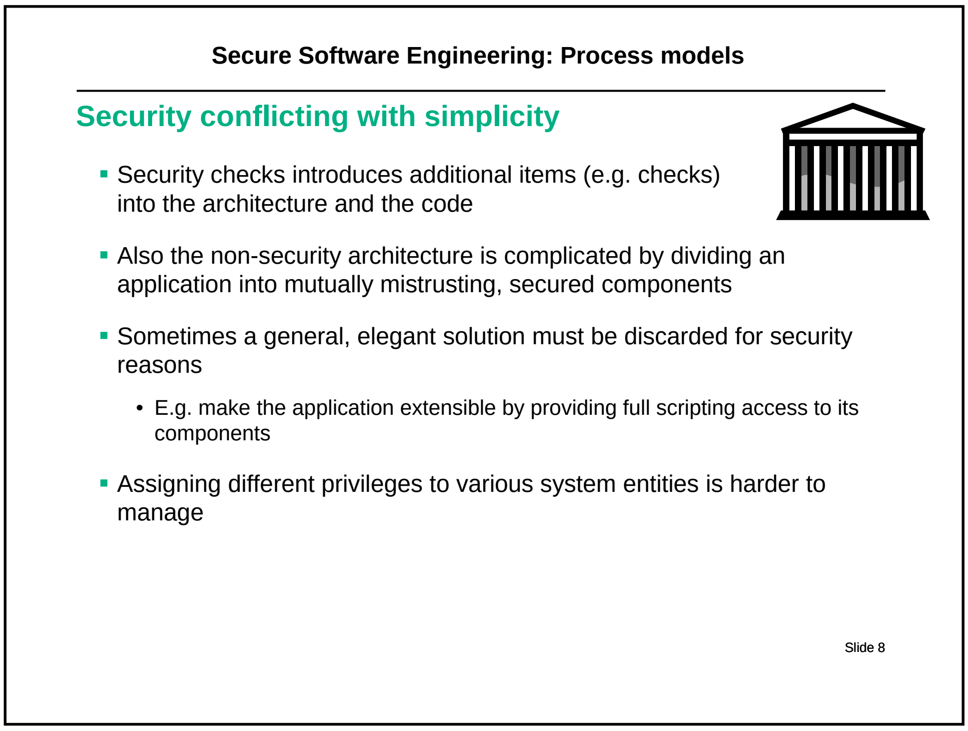
Secure Software Engineering: Process models

Security conflicting with functionality



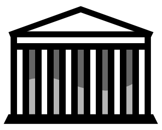
- General principle: The more functionality an application offers, the easier some of that can be abused
- Security calls for few and simple external interfaces ("attack surface")
  - More work needed for data conversion at external interfaces
- Security calls for avoiding "clever tricks" like self-modifying code or offering a scripting interface to the internal functions of the application
- Security advises against compatibility with less secure software (e.g. older versions of the same software)

Slide 7



Secure Software Engineering: Process models

Security conflicting with simplicity



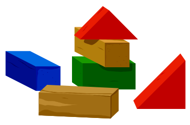
- Security checks introduces additional items (e.g. checks) into the architecture and the code
- Also the non-security architecture is complicated by dividing an application into mutually mistrusting, secured components
- Sometimes a general, elegant solution must be discarded for security reasons
  - E.g. make the application extensible by providing full scripting access to its components
- Assigning different privileges to various system entities is harder to manage

Slide 8

Security conflicting with usability

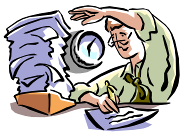
- Using many different privileges is hard to understand
- Having to authenticate (especially repeatedly) is a nuisance
- Disallowing by default makes the user run into many security failures until configuring the system according to their needs
- Avoiding compatibility with insecure software is a nuisance
- Systems trying to recognize attacks and to defend themselves will invariably take some legal user actions for attacks
- ... and of course, anything which hurts performance may also hurt usability ☹

Security conflicting with interoperability / reuse



- Reusing components whose security properties are not fully understood is discouraged
- Finding out and setting the least possible privilege for reused components is difficult (or even impossible with black-box components)
- Avoiding unnecessary functionality and limiting the external interfaces of componentes makes them less general and thus harder to reuse

Security conflicting with low development effort



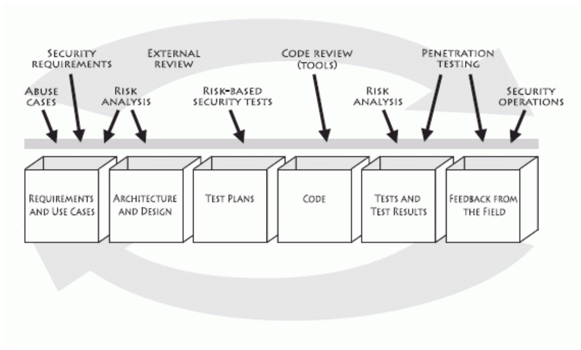
- A secure architecture takes longer to design
- Security checks and settings take time to implement
- Determining the least possible privilege takes time in each case
- Designing a custom solution takes longer then reusing a more general (but potentially insecure) solution
- Determining the security properties of reused components takes time
  - ... unless they have been properly documented ☺

10. Process models

10.2 Seven touchpoints

Secure Software Engineering: Process models

"Seven Touchpoints of Software Security"



In order of importance:

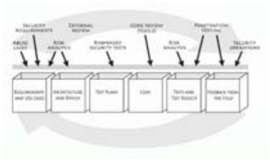
1. Code review
2. Architectural risk analysis
3. Penetration testing
4. Risk-based security tests
5. Abuse cases
6. Security requirements
7. Security operations

Gary McGraw: *Software Security (Addison-Wesley 2006)*

Slide 13

Secure Software Engineering: Process models

Seven touchpoints mapped to this lecture



- Code review
  - See inspections and static analysis in chapter on quality assurance
- Architectural risk analysis
  - See chapter on threat analysis
- Penetration testing, risk-based security tests
  - See testing in chapter on quality assurance
- Abuse cases, security requirements
  - See chapter on requirements elicititation
- Security operations
  - This does not belong in a software development lecture

Slide 14

Secure Software Engineering: Process models

10. Process models

10.3 Microsoft Secure Development Life Cycle (SDLC)

Slide 15

Secure Software Engineering: Process models

Microsoft Security Development Life Cycle

- Introduced at Microsoft in 2002, continuously evolving since then
- Main activities, per phase:
  - **Requirements**
    - Consider security “up front”, assign Security Advisor (stays through project)
  - **Design**
    - Architecture,TCB, least privilege, attack surface, threat model
  - **Development**
    - Code reviews, safer libraries, fuzz testing, static analysis tools
  - **Verification**
    - Penetration testing, other security testing, review legacy code
  - **Final security review of complete system**
    - Any errors found this late serve to improve the whole process

Slide 16

Secure Software Engineering: Process models

MS-SDLC: Requirements phase

- Central (i.e. one per company) security team assigns Security Advisor (SA)
  - SA usually a member of the central security team
  - One SA can serve several projects at a time
  - SA will stay with a project through its Final Security Review
- Development team identifies security requirements
- SA reviews product plan, makes recommendations, ensures resources allocated by management
- SA assesses security milestones and exit criteria

based on a slide by Steve Lipner, Microsoft

Slide 17

Secure Software Engineering: Process models

MS-SDLC: Design phase

- Define and document security architecture

"TCB"

- Identify security critical components (“trusted base”)
- Identify design techniques (e.g., layering, managed code, least privilege, attack surface minimization)
- Document attack surface and limit through default settings
- Create threat models (e.g., identify assets, interfaces, threats, risk), mitigate threats through countermeasures
- Identify specialized test tools
- Define supplemental ship criteria due to unique product issues (e.g., cross-site scripting tests)
- Confer with SA on questions

based on a slide by Steve Lipner, Microsoft

Slide 18

MS-SDLC: Development phase

- Apply coding and testing standards (e.g., safe string handling)
- Apply fuzz testing tools (structured invalid inputs to network protocol and file parsers)
- Apply static code analysis tools (to find, e.g., buffer overruns, integer overruns, uninitialized variables)
- Conduct code reviews

based on a slide by Steve Lipner, Microsoft

MS-SDLC: Verification phase

- Software functionally complete and enters Beta
- Because code complete, testing both new and legacy code
- Security Push
  - Code reviews – focus on legacy code
  - Penetration and other security testing
  - Review design, architecture, threat models in light of new threats

based on a slide by Steve Lipner, Microsoft

MS-SDLC: Final Security Review

- "From a security viewpoint, is this software ready to deliver to customers?"
- 2 – 6 months prior to release; no more security-relevant change expected
- Completion of a questionnaire by the product team
- Interview by a security team member assigned to the FSR
- Review of bugs initially identified as security bugs, but on further analysis were determined not to have impact on security
- Analysis of any newly reported vulnerabilities affecting similar software
- Additional penetration testing, possibly by outside contractors to supplement the security team

based on a slide by Steve Lipner, Microsoft

Microsoft SDLC experiences

- SDLC substantially reduced vulnerabilities in shipped software
  - # security bulletins in Windows 2000 server, 450 days after release: **55**
  - # security bulletins in Windows 2003 server, 450 days after release: **17**
- Introducing the whole SDLC at Microsoft caused moderate additional effort
  - **Michael Howard**, program manager on Microsoft's security team:
    "We estimate it on the order of [a] 12% increase in development cost"
  - **Steve Lipner** (SDLC book author): 20% on 1st iteration, 10% on later iterations
- MS SDLC designed for larger projects with many person-years of effort
  - Many mandatory roles, activities and tools (e.g. MS PreFast): Only for MS?
  - Tailorable? Microsoft says "it's NOT all-or-nothing"

10. Process models

10.4 OWASP CLASP

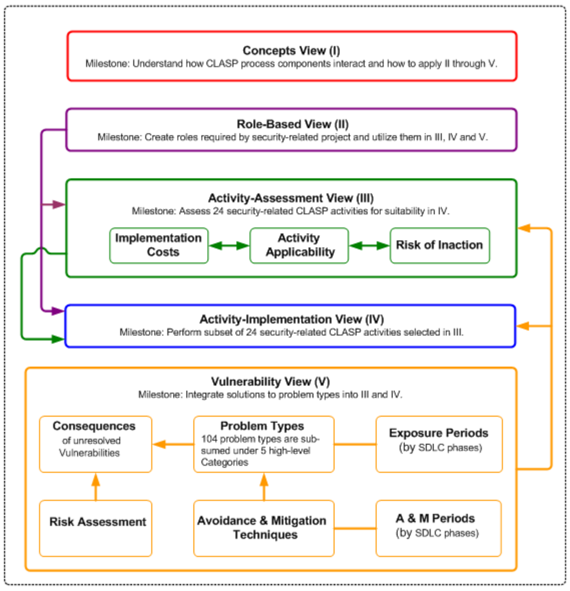
CLASP process

Secure Software

- "Comprehensive Light-weight Application Security Process"
  - Orginally commercial consulting by SecureSoftware Inc., now free: OWASP.org
  - Either stand-alone process or plug-in to Rational Unified Process (RUP)
  - Rather a toolbox of process pieces than one unified process: Tailorable!
- Seven top-level activities, each with detailed instructions and templates
  1. Institute awareness programs
  2. Perform application assessments
  3. Capture security requirements
  4. Implement secure development practices
  5. Build vulnerability remediation procedures
  6. Define and monitor metrics
  7. Publish operational security guidelines

Only these two covered in this lecture: CLASP's scope is much wider!

Secure Software Engineering: Process models



Concepts View (I)
Milestone: Understand how CLASP process components interact and how to apply II through V.
Role-Based View (II)
Milestone: Create roles required by security-related project and utilize them in III, IV and V.
Activity-Assessment View (III)
Milestone: Assess 24 security-related CLASP activities for suitability in IV.
Implementation Costs
Activity Applicability
Risk of Inaction
Activity-Implementation View (IV)
Milestone: Perform subset of 24 security-related CLASP activities selected in III.
Vulnerability View (V)
Milestone: Integrate solutions to problem types into III and IV.
Consequences of unresolved Vulnerabilities
Problem Types 104 problem types are sub-sumed under 5 high-level Categories
Exposure Periods (by SDLC phases)
Risk Assessment
Avoidance & Mitigation Techniques
A & M Periods (by SDLC phases)

- Five views:

1. Concepts
2. Roles

- Activities
  3. Select subset
  4. Perform those

5. Vulnerabities

Slide 25

Secure Software Engineering: Process models

CLASP: Activities and roles (1)

- Institute security awareness program (Project Manager)
- Monitor security metrics (Project Manager, Integrator)
- Manage certification process (Project Manager)
- Specify operational environment (Requirements Specifier)
- Identify global security policy (Requirements Specifier)
- Identify user roles and requirements (Requirements Specifier)
- Detail misuse cases (Requirements Specifier)
- Perform security analysis of requirements (Security Auditor)
- Document security design assumptions (Software Architect)
- Specify resource-based security properties (Software Architect)

Various subsets possible: Can start small

Slide 26

Secure Software Engineering: Process models

CLASP: Activities and roles (2)

- Apply security principles to design (Designer)
- Research and assess security solutions (Designer)
- Build information labeling scheme (Designer, UI Designer)
- Design UI for security functionality (UI Designer, Designer)
- Annotate class designs with security properties (Designer)
- Perform security functionality usability testing (UI Designer)
- Manage System Security Authorization Agreement (Security Auditor)
- Specify database security configuration (Database Designer)
- Perform security analysis of system design (Security Auditor, Designer)
- Integrate security analysis into build process (Integrator)

role "auditor" performs inspections of various documents

Slide 27

Secure Software Engineering: Process models

CLASP: Activities and roles (3)

- Implement and elaborate resource policies (Implementer, Designer)
- Implement interface contracts (Implementer)
- Perform software security fault injection testing (Implementer)
- Address reported security issues (Implementer, Designer)
- Perform source-level security review (Security Auditor, Implementer)
- Identify and implement security tests (Test Analyst, Security Auditor)
- Verify security attributes of resources (Tester)
- Perform code signing (Integrator)
- Build operational security guide (Implementer)
- Manage security issue disclosure process (Project Manager, Implementer, Designer)

Slide 28

Numerous resources included in CLASP

- More than 100 "problem types"
  - Vulnerability categories similar to CWE (actually: mostly part of CWE now)
  - Used as background information referred to in many guidelines
    - *Ex.* Guideline "Use prepared statements for DB access" refers to problem type "SQL Injection" which explains this issue in depth
- List of common security requirements (for copying from)
- Threat analysis guide
  - Including checklists and a list of common threats (referring to problem types)
- Checklists for testing, for assessing COTS (commercial standard sw.)
- Visual modeling notations
  - Simple ad hoc notation, plus a UML extension

10. Process models

10.5 Agile processes and security

Agile processes

- "Agile manifesto" (2001): Declaration of 12 principles for software development processes, operationalizing the following 4 weighting statements:
  - **Individuals and interactions** over processes and tools
  - **Working software** over comprehensive documentation
  - **Customer collaboration** over contract negotiation
  - **Responding to change** over following a plan ("agile"!)
- Several process models are subsumed under "agile":
  - **Extreme Programming (XP)**, SCRUM, Pragmatic Programming, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development, ..
- Gained lots of publicity in early 2000s
  - Including clever PR à la "finally someone listens to the developers"

12 Agile principles (1)

1. Satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Trust them to get the job done.
6. Face-to-face conversation is the best means of communication.

Secure Software Engineering: Process models

12 Agile principles (2)

7. Working software is the primary measure of progress.
8. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective.

Slide 33

Secure Software Engineering: Process models

Agile security?

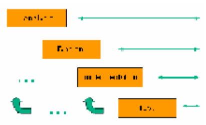
- Agile processes rely strongly on each developer's compentence.
  Not all developers, however, are competent in security (today).
- Agile processes often lack explicit design activities ("emergent design").
  This local focus may miss many design-level vulnerabilities.
- Agile processes are highly iterative. Incorporating frequent security review activities could be an organizational problem when the reviewers are not part of the development team (as they should be).
  - Possibly do reviews only after first and last iteration?
- XP is feature-driven (user stories, coding to the tests).
  Security, however, is not a feature, and might be missed that way.
- XP's pair programming might work well for seamless source code security reviews – provided that developers are security sensitive.

Slide 34

10. Process models

10.6 Integration of security into the development process

Security and various development processes



- Adding security to a process does not require a stricly sequential process ("waterfall model")
  - This lecture follows a classic waterfall order – but this is only because a lecture is a sequential activity ☺
  - Any software development process will contain phases of analysis, of design, of implementation and of quality assurance
    - Only the order and granularity will vary
    - The order and granularity of the corresponding security activities can vary along
- Caveat: Processes without explicit design ("emergent design", code-centric processes) will have trouble to avoid vulnerabilities which arise at design time and cannot be pinned down to specific pieces of code
  - Applies to most "agile processes"

Secure Software Engineering: Process models

Incremental introduction of security

- Incremental introduction is preferable even if the goal is a "security-complete process"
  - Progressing step by step will help with the necessary fine-tuning of the security activities that is usually necessary to fit well with the existing process
- Source code inspections and threat analysis are usually recommended as the first steps
- Individual measures help
- Combined measures may help even more, e.g. ...
  - Testing more effective and efficient if based on threat analysis: Won't omit tests for identified attacks, won't test any irrelevant attack
  - Programming guidelines can be fine-tuned to a specific developer force's needs based on errors found during source code inspections or tests

Slide 37

Secure Software Engineering: Process models

Ch. summary: Security in the software development process

- ... should (and can) be applied in all activities of the process
  - Analysis, design, implementation, quality assurance
- ... is possible with most processes without major disruptions
  - Some processes offer more hooks for security than others, but *all* can profit
  - Amount of "paperwork" can be kept proportional to the "rest" of the process
- ... can (and should) be introduced incrementally
- ... takes initially more effort, but that may expected to pay off over complete product life time
  - Hard numbers from empirical studies still missing
- A few detailed process models for secure software engineering exist
  - OWASP CLASP, Microsoft SDLC; some less comprehensive others

Slide 38