

CS574

Computer Security

San Diego State University

Spring 2008

Lecture #15

Today's Structure

- Administrivia
- Questions
- Lecture

Administrivia

- Question rewrites reviewed.
- Assignment #1
 - Turnin script disabled.
 - Grading not started.
- Assignment #2
 - On webpage.
- Midterm #2

Questions?

Lecture

Assurance & Verification

- The problem:
We can't write all the software that we run on our computer(s).
- We desire:
Assurances that we're not running a totally insecure system.
- We need:
Verification that our computer systems are “good enough” for our purposes.

Security vs Assurance

- Security is provided by security features.
- Assurance is provided by someone who tries to convince us that the security features work as intended.
- As trust requirements go up, so must assurance.
- Authentication::Assurance

Types of Assurance

- Assurance by domain.
- Assurance by feature.
- Assurance by rigor.
- Assurance by approach.

Types of Assurance

- By domain:
 - policy assurance
 - design assurance
 - implementation assurance
 - operational assurance
- By feature:
 - access mediation
 - tamper-resistance
 - security policy enforcement

Types of Assurance

- By rigor:
 - Informal
 - Semiformal
 - Formal
- By approach:
 - Testing
 - Validation
 - Formal Verification

Assurance – Domain

- Policy Assurance

Provide evidence that shows the security requirements of the selected policy is consistent, complete, and sound.

- Design Assurance

Provide evidence that shows the design is good enough to meet the security requirements and security policy.

Assurance – Domain

- Implementation Assurance
Provide evidence that shows that the implementation of the design meets the security requirements and the security policy.
- Operational [or Administrative] Assurance
Provide evidence that shows that the system maintains the security policy throughout installation, configuration, and everyday usage.

Assurance – Features

- Access mediation
Provide assurance that all accesses are mediated (checked).
- Tamper-resistance
Provide assurance that the security mechanisms can't be modified without authorization.
- Enforcement of security policy
Provide assurance that no aspect of the policy can be avoided at any time.

Assurance – Rigor

- Informal
 - Use natural language specification & justification.
 - Minimal rigor.
- Semiformal
 - Use natural language specification & justification.
 - Impose *some* rigor on the process.
- Formal
 - Use mathematical/machine-parsable language.
 - Use rigorous techniques [e.g., mathematical proofs]

Assurance – Approach

- Testing
 - Tiger Teams (penetration testing)
 - Software Testing
- Formal Verification
 - Formal Methods
 - Design and/or Implementation
- Validation
 - Less formal than verification
 - More formal than testing
 - The “happy middle ground”

Assurance – Testing

- Often considered a black art.
- Can't test *everything* – must pick and choose.
- Can only prove that there *are* flaws.
- Provides assurance about:
 - the actual product
 - in real-world situations
 - with actual users

Assurance – Testing

- Often an after-the-fact attempt to inject some quality / create some assurance.
- Often used by snake-oil TCB vendors.
- Are “hacker challenges” worth anything.
- The most widely *accepted* technique.

Assurance – Verification

- Start with a valid security policy [model].
- Verify that the design is consistent with the policy (e.g., MAC, virtualization, covert channel mitigation, etc.).
- Verify that the implementation is consistent with the design (via “formal methods”).

Design Verification

- Model all objects and operations, typically as a (very large) finite state machine.
- Define the entire system's data as one (very large) state vector.
- Define all possible state transitions.
- Identify the “secure states” in terms of the state vector.

Design Verification

- Start in a “secure” state.
- Prove that each possible state transition leaves the system in a secure state.
- Very ugly, very difficult.
- Takes a long time, even with theorem provers.
- Expensive.

Implementation Verification

- Verify correspondence between the design (and/or security policies) and implementation.
- Need two definitions of each procedure:
 - a mathematical (non-procedural) one
 - an implementation (procedural) one
- Define control points and assertions.

Implementation Verification

- Prove that assertions are true at *all* control points.
- Still need those theorem provers.
- Slow and expensive.
- Also a black art.

Assurance – Validation

- Something in between informal and formal.
- Less rigorous than verification.
- More structured than just testing.
- Use “good” software engineering techniques and development methodologies.

Validation Tools

- Requirements
- Design Review.
- Code Reviews
 - peer review
 - formal reviews
- Module Testing.
- Traceability.

Software Life Cycles

- Typical software lifecycle consists of:
 - conception
 - manufacture
 - deployment
 - maintenance
- There are several models for (good) software development.

Software Development

- Waterfall
- Exploratory programming
- Prototyping
- Formal transformation
- Components
- Extreme programming

Waterfall Model

Requirements Definition

Requirements Analysis

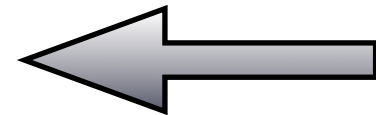
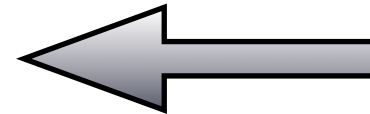
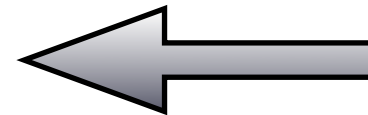
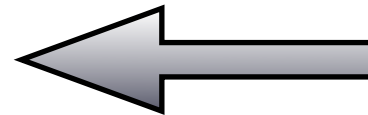
System Design

Implementation

Integration

Acceptance

Maintenance



Other Models

- Exploratory Programming
Start quick and get something minimal running, then keep modifying it until it works correctly, or at least good enough.
- Prototyping
Quickly implement an ugly “first stab”, then analyze the prototype to obtain requirements.

Other Models

- Formal Transformation
Start with a formal (proven) specification, and then repeatedly apply correctness-preserving transformations until you get code.
- Components
Assemble pre-built usable components.
- Extreme Programming
Fast, business-decision driven development.

Open Source

- Arguments against
 - Gives the “bad guys” easy access.
- Arguments for
 - “Many eyes make all bugs shallow”.
 - Follow the crypto community’s lead.
- In practice
 - Most studies show # of faults ~ same.
- My Opinion
 - You’re looking at the wrong thing.

End Lecture

Lecture References

- Pfleeger, *Security in Computing*, 3rd & 4th edition
- Bishop, *Introduction to Computer Security*
- Anderson, *Security in Open versus Closed Systems - The Dance of Boltzmann, Coase and Moore*
- *Department of Defense Trusted Computer System Evaluation*, DoD 5200.28-STD

Reading

- Finish Chapter 5

Finis