CS 591: Introduction to Computer Security

Lecture 11: Assurance

James Hook

Objectives

- Introduce Assurance as a concept/goal
- Introduce methods to increase assurance

Why do you trust an Airplane?

- Which of these do you trust more? Why?

NASA images from web site: http://www.dfrc.nasa.gov/Gallery/Photo/

Boeing images from web site: http://www.boeing.com/companyoffices/gallery/flash.html

Discussion points:

- Who’s flying?
- How long have the airframes been in service?
- Risk/benefit: If you want to go into space you don’t have a lot of choices
  - Best to limit to “apples to apples”

Trusting Commercial Aircraft

- Specification integrity
  - Clear scope of project; goal of aircraft
- Design integrity
  - State of the art engineering analysis of design
  - Extensive modeling (physical and simulation) based on established best-practices of a mature engineering discipline
  - FAA review
- Manufacturing integrity
  - Extensive process controls and tests for all components
  - Rigor appropriate to risk (entertainment system vs. autopilot)

Trusting Commercial Aircraft

- Operational integrity
  - Maintenance is performed by certified mechanics
  - Maintenance performed on schedule
  - Maintenance includes diagnostic measurements confirming conformance to design specifications
  - Pilot is licensed to fly
  - Pilot inspects aircraft prior to flight (and she’s on the plane!)
  - Pilot does not perform maintenance (Separation of duty)
- Feedback
  - Independent investigation of failures
  - If design defects or manufacturing defects are identified the entire fleet can be grounded or repaired

Are all Aircraft Trustworthy?

- Federal regulations reflect risk
- Crudely: Level of assurance increases as potential cost of failure increases
- Commercial aviation is “high assurance”

Can you trust systems that include software?

- Some modern aircraft are “fly by wire”
- How do we trust them?
- FAA
  - Lots of testing
  - Lots of review
  - Lots of process-based controls of both
- Techniques that work for high assurance embedded systems are hard to scale

Trusting Information Systems

- How can we trust an information system?
- What can we trust it to do?
- Can we trust a mechanism to implement a policy?
- How well does the analogy to aviation apply?

The Analogy

- Key factor of trust of commercial airplanes is that we trust the engineering processes used to design, build, maintain, and improve them
- Assurance techniques for information systems are predicated on software engineering practices
  - Is our discipline a sufficiently mature engineering discipline to earn the trust that the public has placed in us?
- Sullivan and Bishop’s presentation builds on what are accepted as best practices in Software Engineering
- Anderson’s presentation is a little more skeptical

Assurance & Trust

- Sullivan builds on three related ideas:
  - *Trustworthy*: sufficient credible evidence that the system will meet … requirements
  - *Trust*: a measure of trustworthiness
  - *Security Assurance*: confidence that an entity meets its security requirements, based on evidence provided by the application of assurance techniques
    - E.g.: development methodology; formal methods; testing; …
- So what’s the difference between “trustworthy” and “security assurance”?
  - Does a system have to be correct to be secure?

Ross Anderson on Assurance

- “Fundamentally, assurance comes down to the question of whether capable, motivated people have beat up on the system enough. But how do you define enough? And how do you define the system? How do you deal with people who protect the wrong thing, … out of date or plain wrong? … allow for human failures?”

Engineers: Avoid Previous Failures

- Sullivan proposes 9 classes of failures
  1. Requirements definition, omissions, and mistakes
  2. System design flaws
  3. Hardware implementation flaws (wiring, chip)
  4. Software implementation errors (bugs, compiler bugs)
  5. System use and operation errors
  6. Willful system misuse
  7. Hardware, communication, or equipment malfunction
  8. Environmental problems, natural, acts of God
  9. Evolution Maintenance, faulty upgrades, decommissions

Study Previous Failures

- RISKs community documents failures
- Sullivan presents three “war stories” to support that the list is reasonable
- We will never prove such a list is sufficient
- As a mature discipline, we will be able to change “best practices” if list is insufficient
  - Tacoma Narrows Bridge

Relevant Tools and Techniques

- Design Assurance: 1, 2, and 6
- Implementation Assurance:
  - Hardware/software errors 3, 4, 7
  - Maintenance & upgrades 9
  - Willful misuse 6
  - Environment 8
- Operational Assurance
  - Operational errors 5
  - Willful misuse 6

1. Requirements definition, omissions, and mistakes
2. System design flaws
3. Hardware implementation flaws
4. Software implementation errors (bugs, compiler bugs)
5. System use and operation errors
6. Willful system misuse
7. Hardware, communication, or equipment malfunction
8. Environmental problems, natural, acts of God
9. Evolution Maintenance, faulty upgrades, decommissions

Software Engineering

- Taxonomy of failures and design methods presupposes Software Engineering Principles
  - Classic lifecycle view of SE posits:
    - Requirements
    - Design
    - Implementation
    - Integration and Test
    - Operation and Maintenance

Design Assurance (broad)

- Requirements: statements of goals that must be satisfied
- For Security assurance, requirements should determine the security policy, or the space of possible security policies (*security model*), for the system
  - E.g. What is the access control mechanism? What are the subjects? What are the objects? What are the rights?
  - Is the access control policy mandatory? Discretionary? Originator controlled?
- The tools introduced in class to date provide a vocabulary for expressing security models, policies, and mechanisms

Policy Assurance

- Evidence that the set of security requirements is complete, consistent and technically sound
  - Complete:
    - Logic: complete means every sentence is either true or false
    - Security: every system state can be classified as “safe” or “unsafe”
  - Consistent:
    - Logic: there is no sentence that is both true and false, or, equivalently that the sentence “false” is not a theorem
    - Security: no system state is both “safe” and “unsafe”.
  - Technically sound:
    - Logic: a rule is sound if it does not introduce inconsistencies
    - ? I think the author intends a necessarily informal notion that the model is appropriate to the situation

Policy Assurance Examples

- The original BLP papers show that the model is complete and consistent
- The Volpano, Irvine and Smith paper shows that the Denning and Denning Information Flow Security concepts can be made sound
  - That analysis is necessarily incomplete (halting problem)
- Many Policy Assurance arguments are carried out using
  - “rigorous mathematics” (I.e. pencil and paper proofs)
  - some use theorem provers (machine checked proofs)

Design Assurance (strict)

- Design is sufficient to meet the requirements of the policy
  - What is a design?
    - Architecture
    - Hardware software components
    - Communication mechanisms
    - Use-cases?
    - Threat profile?

Implementation Assurance

- Evidence establishing the implementation is consistent with the requirements and policy
  - Generally this is done by showing the implementation is consistent with the design, which is consistent with requirements and policy…
- Considerations
  - Design implemented correctly
  - Evidence that appropriate tools and practices used to avoid introducing vulnerabilities (e.g. code insertion/buffer overflow)
  - Testing
  - Proof of correctness
  - Documentation

Operational Assurance

- Evidence the system sustains the security policy requirements during installation, configuration, and day-to-day operation
- Text mentions documentation
- Usability testing is also key
  - Human-Computer Interaction studies are underutilized in mainstream assurance practices!

Coverage?

- Design Assurance: 1, 2, and 6
- Implementation Assurance:
  - Hardware/software errors 3, 4, 7
  - Maintenance & upgrades 9
  - Willful misuse 6
  - Environment 8
- Operational Assurance
  - Operational errors 5
  - Willful misuse 6

All are tasked with 6, do any do an adequate job?

1. Requirements definition, omissions, and mistakes
2. System design flaws
3. Hardware implementation flaws
4. Software implementation errors (bugs, compiler bugs)
5. System use and operation errors
6. Willful system misuse
7. Hardware, communication, or equipment malfunction
8. Environmental problems, natural, acts of God
9. Evolution Maintenance, faulty upgrades, decommissions

Bishop

- Chapter 17 (Contributed by Elizabeth Sullivan)

Assurance

- Myth or Reality?
- Are we behaving like good engineers and avoiding the Failures of Past?
  - Or are we alchemists promising to make gold out of manure?
- If we really cared about code insertion attacks would we use C for routine programming 18 years after the Morris worm?

Confounding Issue

- In Software Engineering which matters more:
  - People
  - Tools
  - Process
- All evidence of which I am aware says people matter more than tools or process
- Given this, can we achieve assurance by mandating tools and process?

Looking Forward

- Next Lecture:
  - Evaluation
  - Reading:
    - Bishop Chapter 18
    - RA Chapter 23
    - NB: The two texts have strongly contrasting views; please review both!