Introduction to Assurance Cases

Sam Redwine
James Madison University
redwinst@jmu.edu
OMG Software Assurance Workshop
March 2007
Agenda

- Introduction
- Security
- Some Software Security Principles
- Assurance
- Secure Software Assurance Case
- Conclusion

*Bonus Charts Not Covered*
Introduction

Sam Redwine
Introduction Outline

- Software Security versus Network Security
- Desirable Characteristics of a Software System Solution
- Specification
- Terms for Bad Things
Software Security versus Network Security

- The fields of network and information security
  - Usually assume software is bad (insecure)
  - Tries to mitigate this

- Software security tries to make software secure or at least more secure
Desirable Characteristics of a Software System Solution

Better
- Functionality
- Performance
  - Capacity, throughput, and speed
- Efficiency – Benefits and Costs
- (Less) Danger
- Opportunity
- Pleasingness
- Certainty

Compliance
- Contracts, laws and regulations, and policy
Desirable Characteristics of a Software System Solution

- Better
  - Functionality
  - Performance
    - Capacity, throughput, and speed
  - Efficiency – Benefits and Costs
  - (Less) Danger
  - Opportunity
  - Pleasingness
  - Certainty
  - Compliance
    - Contracts, laws and regulations, and policy
Specification

Statement of
- What software system guarantees under what conditions
- Behaviors and constraints on behaviors

Agreement/contract with system’s environment on interactions
- May have some assumptions about environment
Some Terms for Bad Things

- **Specification fault**
  - Violations of specification
    - Violation coming from outside
      - **Mistake, accident, mishap, act of nature, subversion, or attack**
    - Violation in static representation of system
      - **Fault, vulnerability** (or more casually “defect”)
    - Violation of constraints on dynamic system state
      - **Error**
  - Violation from inside crossing system boundary – visible outside
    - **Failure**
  - Result – inside or outside system
    - **Adverse consequence, cost, loss**
    - Disclosure, corruption, denial, repudiation, compromise
- Propensity toward Bad Things
  - **Weakness, bad practice**
Security
Levels of Security Requirements

Sam Redwine
Duality of Events and Conditions

- **Bad Events** to eliminate, limit, reduce, or manage
  - Bad “Real-World” Events
  - Bad System-Environment Interactions
  - Bad Computing Asset-related Events
  - Bad System Actions
  - Actions needed to prevent, avoid, ... respond to bad actions and events

- **Good conditions** to be established and preserved
  - “Real-World” conditions
  - Flow constraints
  - Asset properties (e.g. Confidentiality, Integrity, Availability)
  - Computing system states
  - Ability to establish and maintain condition
Examples

<table>
<thead>
<tr>
<th></th>
<th>Bad Event</th>
<th>Good Condition Always True</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Real World</strong></td>
<td>Reactor Meltdown</td>
<td>Reactor Temperature within limits</td>
</tr>
<tr>
<td><strong>System-Environment Interactions</strong></td>
<td>Dangerous human instruction</td>
<td>All human operators properly trained</td>
</tr>
<tr>
<td></td>
<td></td>
<td>All commands allowed to go to reactor are predicted to be benign</td>
</tr>
<tr>
<td><strong>Computing Assets</strong></td>
<td>Table of limits or History data corrupted</td>
<td>Table of limits changed only by authorized entities and history data never changes</td>
</tr>
<tr>
<td><strong>Software System</strong></td>
<td>Operation involving outside or asset is not logged</td>
<td>Every operation involving an asset has proper authorization and is logged</td>
</tr>
<tr>
<td><strong>Enabling Functionality</strong></td>
<td>Identification management lacking</td>
<td>Required functionality is available, correct, and tamper proof (and also needs to be not bypassable)</td>
</tr>
</tbody>
</table>
Requirements Frames+
Computing System-
Environment Interaction

Consequences

Computing System

Engineering Representation

Software and Hardware
Software in Danger All its Life

From
Non-malicious and malicious events
Attacks Possible during All Activities and in all Environments
Maliciousness

- Existence of maliciousness does not make non-malicious problems go away
- Performing to specification
  - Not probabilistic reliability and availability
  - Adversaries often attack where least expected
  - Anything could happen
- Adversary wants best thing for her/himself – may include wanting to make the worst possible thing happen to you at the worst time
Non-Malicious Events

- Mistake, accident, mishap, act of nature
- Ignorance and incompetence
- Poor process
- Well intended but exploitable feature
- Recreational events – Easter eggs
- Not adequate as environment changes
- Natural disasters
Properties – Traditional Security
Definition in Red

- Validity
- Predictability
- Correctness
  - Agrees with Specification
- Availability
- Reliability
- Legitimacy
- Integrity
- Computational Difficulty
- Confidentiality
- “Multi-level Security”

- Violation Tolerance
- Damage Confinement
- Accountability
  - Non-repudiation
- Validatability and Verifiability
  - Analyzability
  - Testability
    - Observability
    - Controllability
- Maintainability
- Assurability
- Certainty
Security Properties – CIA+

- **Confidentiality**: preventing unauthorized disclosure
- **Integrity**: preventing unauthorized alteration
- **Availability**: preventing unauthorized destruction or denial of access or service
- **Accountability**: knowing what entity did what when
  - **Non-repudiation**: ensuring the inability to deny the ownership of prior actions

**Identification**: known identities for entities as much of security is about who can do what when

**Authentication**: verifying identity ensuring entity identified correctly
Specifications

- Functionality
- Constraints on behavior of functionality

Security properties; emergent system properties

For example, constraints in forms of
- Never allow anyone but good guys to change sensitive data
- Security functionality is not bypassable
Traditional Security and Safety

Security
- Adverse consequences
- Non-maliciousness
- Illegitimacy
- Maliciousness

Safety
- Adverse consequences
- Non-maliciousness

Behavior can be result of outside entity or inherent behavior of system or software.
Using Probability

Can malicious behavior be validly modeled probabilistically?

Reasonably frequent similar malicious behavior generally can be modeled probabilistically. E.g. car theft, household burglaries, script kiddies

However, serious adversaries tend to, “Attack where, when, and how their opponent least expects.”

That is, where probability is judged to be low.
Conclusion

- Security includes
  - Confidentiality,
  - Integrity,
  - Availability, and
  - Accountability,
- But ultimately it is about adverse consequences and their uncertainty

- A common way to think about security-related software behavior is with constraints
- Probability-based analysis may not work for maliciousness
Some Principles for Secure Software

Sam Redwine
Secure Software Principles

- Define explicit guidelines for certain characteristics of
  - System being produced
  - How software development activities should be performed.

- Principle essentially characterizes an ‘ideal’
- Apply to all phases of the software life cycle.
Saltzer-Schroeder Security Principles

- Economy of Mechanism – design simplicity
- Fail-safe defaults – deny access unless explicitly authorized
- Complete mediation – check every access
- Open design – visible
- Separation of privilege – multiple entities required
- Least privilege – only has privileges need to do its work at this time
Saltzer-Schroeder Security Principles, cont.

- Least common mechanism – avoid shared mechanisms
- Psychological acceptability – ease of use and operation
- Work factor – cost commensurate with risks
- Recording of compromises – trails of evidence
Conclusion

- Well-proven principles exist such as least privilege and complete mediation
- Many other principles exist
Secure Software Assurance

- Assurance
- Assurance Case
- Justified Confidence
- Chain to Usage of Product
Assurance

The term “Assurance” is used in several ways, but underlying concept is to reduce uncertainty.

To rationally decide to use software in dangerous situation one needs:

- The software
- Justified confidence in it
- Circumstances such that with this level of confidence use will have acceptable (or at least tolerable) potential consequences.
Justified Confidence

To have one’s uncertainty reduced so have basis for justified confidence in a security or safety need convincing

- Claim
- Evidence
- Arguments that tie evidence to claim

Implies valid evidence and arguments

Together these three – claims, arguments, and evidence – make the “assurance case”
Chain

Software System

Observations and Inferences

Reason and Calculate

Conclusion & Uncertainty

Appraise, Judge

Confidence in Conclusion

Decide what, where & when Circumstances would Warrant

Place Trust

Belief in Trustworthiness

Trust with Task

Benefits

Danger & Damage

Engineer

Decision Maker

Copyright 2005-2007 Samuel T. Redwine, Jr.
What Needs to be Shown

- Specified security objectives and properties are valid and meeting them will result in meeting real world intentions and expectations.
- System as designed, built, deployed, and executed will meet its specified security properties.
What Would be a Basis for Confidence?
Assurance Case Builds Arguments from Evidence to Claim

Arguments build on each other

Present to Stakeholders
Build Up Convincing Argument
Unacceptable System

Assurance Case either
1. Unconvincing
2. Convincing in showing system inadequate
Claims regarding Security can be Made about anywhere in Lifespan
Kinds of Arguments

- **Every significant** risk identified and adequately (together) mitigated

- **Every necessary** activity and item ok such that consequences ok
  
  - E.g. Specification is of a system with proper security properties and system built to agree with specification
  
  - And everything in system is in specification
From a negative viewpoint might argue

Premise: Target objectives are defined for the losses of interest

- Targets possibly set by the customer and/or socio-political environment

Conclusion: All target objectives demonstrated to have been achieved, as well as actually being achieved so altogether acceptable (or tolerable)

Casually: all significant weaknesses or risks known and altogether adequately mitigated
Overall Argument Structures - 2

From a positive viewpoint one might argue the proposition that

- Premise: A description of the software system exists (usually high-level specifications) in which an acceptable (or tolerable) confidence exists that it represents a software system that will establish and preserve the required security properties

- Conclusion: The implemented software system exactly agrees in all relevant ways with the description (no more and no less) to a level of confidence that when combined with the confidence level in the premise results in an acceptable (or tolerable) confidence level

- Typically: specification is ok and software built to specification
Evidence about Production

- Environment
- Intentions
- Process
- Means and resources
  - Includes quality of people
- Products
- Support
- Capabilities (possibly not yet exercised)

Uncertainty in these?

Characteristics, history, observations, measurements, evaluative results, analyses, inferences

Not knowing something may also be relevant

Everything you know is potentially relevant
Uses of Assurance Case

- Planned assurance case helps determine development plan and activities
- For developer: assurance case contents (so far) need to be adequate at each step
  - Especially release
- Assurance case helps decide purchase and use

Producer’s Assurance Case at release should be adequate to back sales claims and consumers’ needs for confidence.
Start of Chain

Software System

Observe, Analyze, Reason

Engineering Representation and System including Dynamic Behavior

Reason and Calculate

Observations and Inferences

Conclusion & Uncertainty

Appraise and Judge

Predicted Dangers
Assurance Case Contents and Quality

Samuel T. Redwine, Jr.
Assurance Case Contents

- Explicit set of **Claims**
- Systematic structure for **Evidence**
  - Gathering, preserving, analyzing and evaluating, reporting, and organizing
  - Including its metadata
- **Arguments** that link claims to evidence
  - Interpreting its meaning, relevance, uncertainty, and significance
- **Assumptions** and **Judgments** underlying arguments
- Possibly multiple **Viewpoints** and **Levels of Detail**
Draft Generic Software Security Assurance Case

Samuel T. Redwine, Jr.
Suggestive General Top-Level

TLS = Top-Level Spec

Draft

Copyright 2005-2007 Samuel T. Redwine, Jr.
Fragment – Agrees with All of Policy
All Behaviors?
Claims

Sam Redwine
Properties or Attributes and Claims

- Accountability
- Accuracy
- Availability
- Causation
- Confidentiality
- Consequence
  - Losses
- Consistency
- Correctness
- Integrity
- Knowledge of
- Non-repudiation

Claims of
- Value or Relationship
  - True or false
  - Numeric value
  - Other value or relation type
  - Unit of measure
  - “Substance” measured
- Timing
- Uncertainty
- Conditionality of value, relation, or uncertainty
Arguments

Sam Redwine
Kinds of Reasoning Systems

“Quantitative”
- Deterministic
  - E.g. formal proofs
- Non-deterministic formal systems for reasoning
  - Probabilistic
  - Game theoretic
    - E.g. minimax
- Other uncertainty-based formal systems of reasoning
  - E.g. fuzzy sets, others used in AI

Qualitative
- E.g. staff skill and experience, compliance with standard, qualitative statements of event causality
Mindsets

- Research
- Engineering
- Project Management
- Risk Management
- Financial
- Executive Management
- Regulatory
- Litigation/Liability
- Marketing
- Buyer
- User
- Intelligence analyst
- Attacker

- Safety
- Security
- Correctness
- Counterintelligence
- Crime
- Terror
- Industrial competitiveness
- Political or social activism
- Subversion
- War fighting
Kinds of Arguments

- Every significant risk identified and adequately (together) mitigated
- Necessary items done or exist such that (altogether) consequences will be acceptable (or tolerable)
  - E.g. Specification is of a system has proper security properties and system built to agree with specification (and nothing more built than is in specification)
Top-Level Claim and Argument

Significant potential losses are adequately limited

- Losses Sufficiently Identified
- Altogether Identified Losses Adequately Limited by the
- Individual Requirements Specified
  These
- Individual Requirements are Well-Specified and
- System Meets Individual Requirements

Derives from the structure in SafSec Standard
Losses Sufficiently Identified

- Losses are sufficiently identified
  - The causal basis of the identified losses is established
  - Loss-related causal analysis is sound
  - All identified losses and other significant causal steps have associated size/severity and uncertainties/likelihoods defined
Altogether Losses Adequately Limited

- Argument exists that individual requirements/policies adequately limit overall losses based on design of actual system (e.g. preventive measures, constraints, tolerance, local requirements, characteristics, or countermeasures)

- Causal analysis reflects the aggregate affects of Individual reqts.
  - A statement exists that explains how Individual reqts. limit losses
  - Individual reqts. are together sufficient to achieve overall limit on losses
Individual Requirements Specified

Individual requirements are specified (that adequately limit all the identified losses)

E.g. Acceptability of losses related to each individual reqt. defined by setting budgeting targets for each
Requirements Well-Specified

Each individual requirement is sound capturing all the key dependability characteristics

Individual reqts. are realizable specifications

Appropriate Assurance Requirements defined for each Individual reqt.
System Meets Individual Requirements

Actual system is shown to meet all Individual requirements

- Arguments exists that implementation actually meets Individual reqts. based on appropriate evidence

  - Actual system is complete
  - Evidence relates to actual system
  - Evidence shows that the system meets the Individual requirements.
  - Evidence meets assurance requirement being valid and adequately limiting uncertainties
Use of Formal Methods

Formal Methods Evaluation Red Dashed

Modified from a diagram by Praxis Critical Systems

Copyright 2005-2007 Samuel T. Redwine, Jr.
Faults, Errors, and Dangers View

System States

- Safe State
- Error State
- Safe and Secure
- Failure and Possible Violation
- Unsafe and Secure
- Unsafe and Insecure
- Safe and Insecure
- Recover

Fault activated

Copyright 2005-2007 Samuel T. Redwine, Jr.
Chain

Software System

Observations and Inferences

Observe, Analyze, Reason

Conclusion & Uncertainty

Appraise, Judge

Reason and Calculate

Confidence in Conclusion

Decide what & when Circumstances would Warrant

Belief in Trustworthiness

Place Trust

Trust with Task

Danger & Damage

Engineering Representation and Dynamic Behavior

Predicted

Copyright 2005-2007 Samuel T. Redwine, Jr.
Coverage of Arguments and Evidence

- Entire lifecycle and across supply chain
- Evidence for and against (counter-evidence)
- All aspects
  - Environment
  - Intentions
  - Process
  - Means and resources (including people and tools)
  - Products
  - Support
  - Capabilities (possibly not yet exercised)
Quality of Arguments

- Rigorous
  - Based on proven scientifically established methods and principles
- Diverse and independent
- Insensitive to small changes in values
- Robust

Assumptions and Judgments
- Few
- With low criticality
- Reflect high relevant expertise and experience
- Reviewed and evaluated
- Reflect wide agreement among knowledgeable
- Who also state they have low uncertainty

Result in low uncertainty
Assurance Relevant Evidence and Characteristics

Sam Redwine
Evidence about Production

- Environment
- Intentions
- Process
- Means and resources - Includes quality of people
- Products
- Support
- Capabilities (possibly not yet exercised)

Uncertainty in these?

Characteristics, history, observations, measurements, evaluative results, analyses, inferences

Much more in Bonus Charts

Everything you know is potentially relevant

Not knowing something may also be relevant
Common Characteristics of Successful High-Security Projects

- Adequate time and resources
- High-quality people with the right expertise (and experience)
- Extensive review
- Specification whose ambiguities have been resolved
- Clean, relatively stable architecture

Can they answer question, “How do you know it is secure?”
Conclusion

- Security includes Confidentiality, Integrity, Availability, and Accountability, ultimately it is about adverse consequences and their uncertainty.
- Probabilities may be poor for maliciousness.
- Principles such as “Least Privilege” exist.
- Must have Software and Justified Confidence.
  - Grounds for the latter is Assurance Case with claims, arguments, and evidence.
Questions

- Hopefully, you have background for remainder of tutorials and workshop
- Questions
Bonus Charts

Sam Redwine
Kinds of Evidence

- The quality and history of people
- The characteristics, suitability, and history of processes used
- Data on the quality and fidelity of use of process
- The quality and suitability of the development environment
- Production organizational structure characteristics and suitability
- Security of production
- The realism of the assumptions made
- Quality of safety or security policy
- Agreement of informal and formal statements of security properties (and/or other properties, e.g. safety)
- Specification quality and consistency with these properties
- Consistency from representations have confidence in (e.g. specification) through intermediate representations (e.g. design, source code) to the released software
- Security of deployment
- Security of installation

Source: Sam Redwine
Kinds of Evidence (Cont’d)

- Security of updating the software
- Software executed is same as what was installed (and updated)
- And on across lifespan and possible situations (e.g. stolen laptop)
- Lack of non-required or unspecified functionality
- Analysis of structure of system and its interactions
- Chain of evidence: valid creation and collection of evidence with its integrity maintained until and during its use
- Proper use of assurance argument and evidence
- Design including defense in depth and tolerance
- Characteristics of the code including good practices and use of a safe programming language (subset)

Source: Sam Redwine
Kinds of Evidence (Cont’d)

- Static analysis of code for relevant aspects such as
  - Vulnerability-prone features
  - Exceptions
  - Information flow
  - Conformance to coding standards
  - Correctness of proofs

- Nature and results from
  - Reviews
  - Audits
  - Tests
  - Measurement (direct or surrogate)
  - Analyses
  - Simulations
  - Proofs

Source: Sam Redwine
Kinds of Evidence (Cont’d)

- Data on the execution history of the software itself – particularly for post-update assurance
- Ease of recovery and repair
- Quality of support (particularly response to discovery of defects or vulnerabilities)
- Evidence from suppliers or other sources
- Warranty or risk sharing agreement and its creditability

Source: Sam Redwine
Characteristics

- Analyzable: Capable of being checked
- Attentive: Alert, vigilant, observant, watchful
- Capable: Having required or wanted skills or faculty
- Complete: Having all the necessary parts or providing a total solution
- Consistent: Uniform and steady
- Controlled: Kept within defined limits
- Correct: Free from error, defect, or fault with respect to a higher level specification
- Defined: Described by a fixed set of parameters
- Documented: Committed to writing
- Easy-to-Use: Capable of being put in service, performed, or maintained without difficulty
- Effective: Produces the desired result
- Efficient: Performs with a minimum of waste, expense, or unnecessary effort
- Evaluated: Tested against a standard
- Experienced: Having performed similar events or activities
- Fault Tolerant: Tolerant of mistakes or errors

Characteristics

- Knowledgeable: Possessing requisite information
- Learning: Capable of acquiring knowledge as result of experience
- Managed: Operates according to a plan; an organized effort
- Mature: Well seasoned; time-tested
- Measurable: Capable of having dimensions, quantity, or capacity ascertained
- Optimized: Designed and built with efficiency in mind; fine tuned for performance
- Predictable: Anticipated, expected, foreseen, prepared for
- Profiled: Described in a way prescribed by a standard criteria
- Quality Focused: Very concerned about quality
- Rated: Scored against a standard
- Recoverable: Able to be repaired or brought back from harm
- Repeatable: Capable of being performed, experienced, or produced again
- Reputable: The estimation in which a person or organization is held by the public
- Robust: Able to continue; resistant to undesirable change
- Scaleable: Scope or granularity can be adjusted to meet changing circumstances
- Stable: Unwavering; not subject to excessive variation
- Strong: Capable of enduring or being defended
- Successful: Possessing a high rate of past success
- Tested: Subjected to a regimen of testing
- Trustworthy: Deserving of confidence that a responsibility will be fulfilled
- Well Understood: Universally comprehended across the entire enterprise

Source: Williams, 1998
### Attributes of Trustworthy Systems

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Method</th>
<th>Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptable</td>
<td>n-tier architecture</td>
<td>value of n</td>
</tr>
<tr>
<td>Available</td>
<td>redundant</td>
<td>hours of operation /total hours</td>
</tr>
<tr>
<td>Complete</td>
<td>inspected</td>
<td>traceable shalls delivered/shalls required</td>
</tr>
<tr>
<td>Conforming</td>
<td>inspected</td>
<td>shalls delivered/shalls required</td>
</tr>
<tr>
<td>Consistent</td>
<td>inspected</td>
<td>standards compliance</td>
</tr>
<tr>
<td>Correct</td>
<td>proved</td>
<td>defects in satisfying intended function</td>
</tr>
<tr>
<td>Defect free</td>
<td>inspected</td>
<td>defects detected</td>
</tr>
<tr>
<td>Dependable</td>
<td>aged</td>
<td>user feedback survey</td>
</tr>
<tr>
<td>Extensible</td>
<td>architected</td>
<td>plug-in connections</td>
</tr>
<tr>
<td>Failure free</td>
<td>tested</td>
<td>failures detected</td>
</tr>
<tr>
<td>Fault free</td>
<td>white box tested</td>
<td>faults detected</td>
</tr>
<tr>
<td>Interoperable</td>
<td>architected</td>
<td>parameter list alignment</td>
</tr>
<tr>
<td>Low complexity</td>
<td>designed well</td>
<td>cyclomatic and essential complexity</td>
</tr>
<tr>
<td>Maintainable</td>
<td>architected</td>
<td>standards compliance</td>
</tr>
<tr>
<td>Modifiable</td>
<td>architected</td>
<td>adaptation parameters</td>
</tr>
<tr>
<td>Open</td>
<td>architected</td>
<td>TBD</td>
</tr>
<tr>
<td>Performance</td>
<td>architected</td>
<td>deadlines met/total deadlines</td>
</tr>
<tr>
<td>Predictable</td>
<td>architected</td>
<td>variance</td>
</tr>
<tr>
<td>Private</td>
<td>policy enforcement</td>
<td>incidents</td>
</tr>
<tr>
<td>Reliable</td>
<td>black box tested</td>
<td>mean time to failure</td>
</tr>
<tr>
<td>Resilient</td>
<td>survivable function</td>
<td>availability</td>
</tr>
<tr>
<td>Safe</td>
<td>due diligence</td>
<td>standards compliance</td>
</tr>
<tr>
<td>Scalable</td>
<td>architected</td>
<td>loops within loops</td>
</tr>
<tr>
<td>Secure</td>
<td>security function</td>
<td>function present</td>
</tr>
<tr>
<td>Stable</td>
<td>architected</td>
<td>change history</td>
</tr>
<tr>
<td>Survivable</td>
<td>survivable functionality</td>
<td>function present</td>
</tr>
<tr>
<td>Tested</td>
<td>due diligence</td>
<td>change history matches test history</td>
</tr>
<tr>
<td>Traceable</td>
<td>requirements managed</td>
<td>none, simple, complex</td>
</tr>
<tr>
<td>Understandable</td>
<td>architected</td>
<td>maintenance personnel feedback</td>
</tr>
<tr>
<td>Usable</td>
<td>user friendly</td>
<td>user feedback</td>
</tr>
</tbody>
</table>

Source: Don O’Niell
Some References

- Common Criteria v. 3.1, 2006.
- CJCSM 6510.01, Defense-In-Depth: Information Assurance (IA) and Computer Network Defense (CND), DoD Joint Staff, 2004. (Official Use Only)
References – Safety and Security

- Redwine, S. (Editor). *Secure Software Assurance: A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure*