30 Years of Software Assurance: What we have learned, and what we haven’t

Ronda R. Henning
rhenning@harris.com
Assurance -- Defined

- Pledge or promise – a declaration that inspires or is intended to inspire confidence.
- Confidence, in your ability or status
- Certainty, freedom from uncertainty
- Making something certain, overcoming doubt
- Insurance against certainty

-- Microsoft Encarta
But....

• It’s not a testable definition
  – How to test for intention?
  – How to provide certainty or freedom from doubt?
## The timeline of assurance

<table>
<thead>
<tr>
<th>Year</th>
<th>Purpose</th>
<th>Security Policy</th>
<th>Mechanisms</th>
<th>Philosophy</th>
<th>Tag Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1970</td>
<td>Specialized Uses</td>
<td>None needed</td>
<td>Physical Protection</td>
<td>Common Good</td>
<td>Woodstock</td>
</tr>
<tr>
<td>1980</td>
<td>Timesharing/ Early Internet</td>
<td>Userid + Password</td>
<td>Gold standard = verified code</td>
<td>Some were uncommon</td>
<td>“Mistakes don’t happen”</td>
</tr>
<tr>
<td>1990</td>
<td>Computer as Commodity</td>
<td>MAC and DAC w/labels</td>
<td>Pervasive TCSEC “C2 by 1992”</td>
<td>Painstaking Evaluation</td>
<td>Paranoia</td>
</tr>
<tr>
<td>2000</td>
<td>“Smart devices”</td>
<td>RBAC</td>
<td>Common Criteria</td>
<td>User Specified strength of countermeasures</td>
<td>Identity Theft</td>
</tr>
</tbody>
</table>
Why Assurance is Hard

- User data
- Enterprise data
- User Interface Utilities
- General purpose applications
- Network protocol stacks
- Operating system utilities
- Operating system kernals
- Firmware
- Motherboards
- Microprocessors
- Processor functions
- Logic gates

Assurance and Design Formalism

Complexity and size
The Reality

• The world changes
  – Requirements for protection change in response to threats
• Not all data is created or protected equally
  – Some is “more sensitive” than others
  – Some is more perishable than others
• When we treat security as static, we become obstacles and not enablers
The Orange Book
- Linked strength of mechanism with strength of assurance
- All or nothing concept

The Common Criteria
- User defines what functional and assurance objectives are
- Developer explains how they are met
- Independent lab verifies the claims
Standards (Continued)

• ISO/IEC 17799
  – Good policies and practices make good neighbors!
• Capability Maturity Model Integrated
  – Process is good, but not specific
• SSE-CMM
  – Process is not only good, but security and assurance bring additional processes to the framework.
• Failure – deviations from specified behavior
• Fault – failure that doesn’t necessarily impact the whole system
• Error – Impacts the operation of the system as a whole, and implies defects prevent correct operation

– Dobson and Randell
Specify Mission Requirements

Define System Architecture

Does it fulfill the business mission?

Context Scenarios That Threaten the Business mission

Generate

No, Redesign

Yes

Use results to define modifications for high-level system survivability

Evaluate the Architecture for Technological Soundness

Gather Evidence Of Assurance
- Product attributes
- Mission attributes
- Vendor(s) choice
- Vendor(s) Dev. Cycle
- Organization’s Risk Mgmt. Program

Annotate Architecture With Vendor provided Assurance data
- Openess of component interfaces
- Full access to source
- Engineering & design artifacts

Organization Accepts system risk or defines mitigation strategy

Source: Software Engineering Institute (V-Rate Methodology)
**The Microsoft Way**

**Requirements**
- Increase SW security
- Decrease disruption of plan & schedule
- Goals, challenges, plans

**Design**
- Requirements
- SW Structure
- Architecture & Design Guidelines

**Threat Model**
- Identify potential harms
- Identify probability of successful attack
- Determine: Critical features
  - Test points
  - Code modules

**Implementation**
- Code and test standards
- Special attention to threat model vulnerabilities
- Scanning Tools
- Code analysis

**Support & Service**
- Learn, detect, & correct flaws, exploits
- Evaluate advisories
- Take corrective actions

**Release**
- Final security review
- Independent team
- Examine flaws, test results
- Potential penetration testing

**Verification**
- Extra Security Review
- Focused Test
- Review what’s been developed, updated, and modified
- Priority to attack surfaces

**Supplemental Shipping Criteria**
- Status of evaluation
- Extra Test
- Configuration requirements

---

*Source: Lipner*
<table>
<thead>
<tr>
<th></th>
<th>Functionality</th>
<th>Speed</th>
<th>Fault-tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHY?</td>
<td>Does it work?</td>
<td>Is it fast enough?</td>
<td>Does it keep working?</td>
</tr>
<tr>
<td>WHERE?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Completeness</td>
<td>Separate normal and worst case</td>
<td>Safety first</td>
<td>End-to-end</td>
</tr>
<tr>
<td>Interface</td>
<td>Do one thing well:</td>
<td>Make it fast</td>
<td>End-to-end</td>
</tr>
<tr>
<td></td>
<td>Don't generalize</td>
<td>Split resources</td>
<td>Log updates</td>
</tr>
<tr>
<td></td>
<td>Get it right</td>
<td>Static analysis</td>
<td>Make actions atomic</td>
</tr>
<tr>
<td></td>
<td>Don't hide power</td>
<td>Dynamic translation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use procedure arguments</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Leave it to the client</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keep basic interfaces stable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td>Keep a place to stand</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Plan to throw one away</td>
<td>Cache answers</td>
<td>Make actions atomic</td>
</tr>
<tr>
<td></td>
<td>Keep secrets</td>
<td>Use hints</td>
<td>Use hints</td>
</tr>
<tr>
<td></td>
<td>Use a good idea again</td>
<td>Use brute force</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Divide and conquer</td>
<td>Compute in background</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Batch processing</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Summary of the slogans

Everything we ever needed, we learned in the early 1970's -- Lampson
Conclusions

What have we learned:

– Countermeasures are better
– Defense in Depth helps
– Process Improvement Initiatives institutionalize improvement
What we haven’t learned

• Discipline
  – Computer science and system design is still an art
  – Engineers that understand integration and allocation of assurance are hard to find
  – We substitute testing for early error detection – and pay the penalty.
In Summary

- Those who do not learn from the mistakes of the past are doomed to repeat them.

- Forums such as this capture our attempts to learn about assurance, and to learn how to implement it more effectively.