



***Balancing Cost and Assurance in System
Development
A Sample Case and Some Thoughts***

Rick Schantz
Jérôme Hugues

Monterey Workshop
October 17, 2006

Some Starters

In the context of Information Systems, Assurance deals with

1. Doing something to prevent, identify, recover from, tolerate and/or deal with undesirable outcomes
2. Establishing confidence that the measures taken in 1 are adequate and that they work (only) as intended

There are many, diverse categories of cost

There are many, diverse dimensions of assurance (including trust); its not just about software bugs

There are many stakeholders who make different tradeoffs with respect to balance

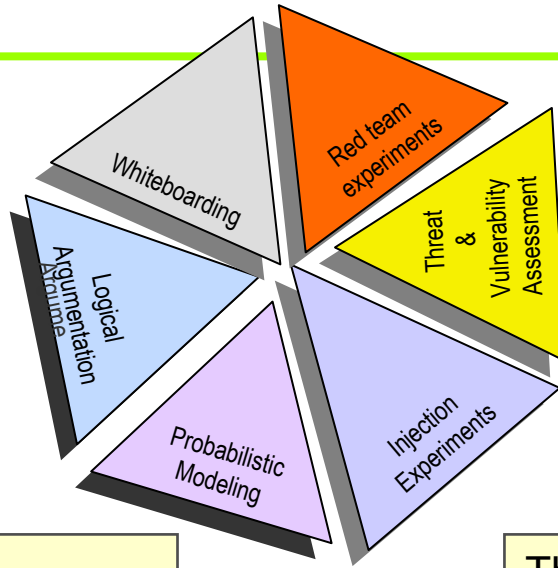
Costs are real and countable

Assurance is subjective (in the eye of the beholder) and infinite

Assurance consists of process, argument & evidence

Validating DPASA Survivability Architecture

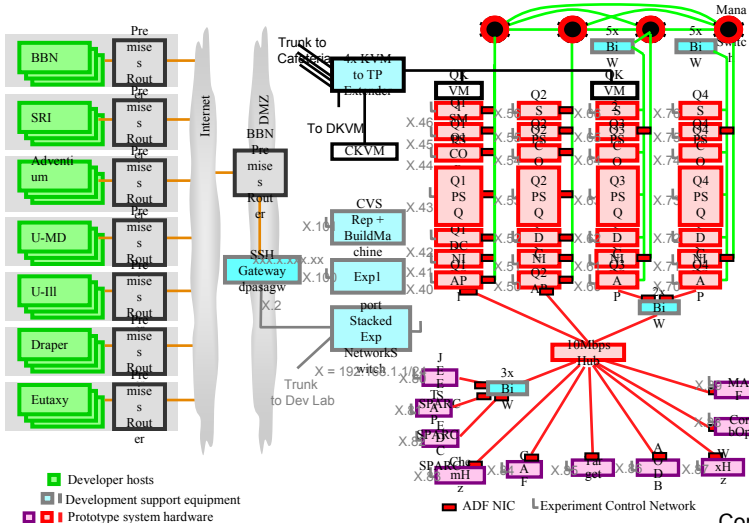
Current best practice for evaluating security and survivability: red team experiments and white boarding



DPASA employed multiple synergistic validation techniques to gain **confidence** that the architecture achieves the desired high level of assurance.

100s of attack possibilities considered threats and attacks are managed

The DPASA IT-JBI design provides critical functionality with high probability even when under heavy successful attack.



Demonstration at Rome Labs March 2005 where the defense-enabled system faced class A red team attacks from two different groups

Survivability Assurance in Practice

- Undesirable outcome is loss of service availability/integrity under cyber-attack
- Desired assurance level: 12 hours of continuous service under sustained attack, from advantaged starting points
- Various cost dimensions include redundancy, diversity, cryptography, coordination, system, performance goals, usability, in addition to hardware, software, development, maintenance, upgrade (budget)
- Experience led to partitioned techniques:
 - Fix an initial calibration for assessing confidence
 - Try varying measures to avoid the undesirable
 - Repeated iteration
- Fixed calibration (initial starting point) :
 - Attacker objectives, and attack paths derived there of
 - Look for existence of overlapping defense against each attack path
 - Enough if each attack path has at least two
- Means to prevent the undesirable:
 - Redundancy (eliminate single point failures)
 - Heterogeneity (prevent common mode failures)
 - Reconfiguration (recover, degrade from attack induced failures)
 - Uncertainty (thwart planned and staged attacks)
- Iteration:
 - Did we find at least 2 overlapping defenses in each attack path? (if not add more redundancy, reconfiguration etc...)
 - Should we consider more than 2 overlapping defense?
 - Did we consider all attacker objectives?

Observations Over Collections of Real Development Practices (Controversial)

- With exceptions, assurance takes a back seat to cost considerations
- Our commonly accepted and practiced concepts, practices, tools, techniques, etc for assurance of large scale software systems dramatically lags our concepts, practices, tools, ..., for (almost) every other aspect of software engineering
- Assurance catches up typically only after a disaster of some sort
- We're quickly making matters worse by demonstrating the construction of much more complex systems
- Assurance is a total lifecycle, significant recurring cost, repeated with each upgrade change
- There does not seem to be (except perhaps in specific domains) any established basis for transforming a potentially infinitely continuous notion of assurance for software intensive projects) into commonly accepted and quantized stopping points (e.g. discrete pay more, get (how much) more) (like Orange book or CMM); Perhaps there should be
- Using COTS vs. custom
- Immediate vs. upstream costs
- Metrics are often misleading and squishy (despite much attempted rigor) and requirements (specifications) woefully incomplete