Automated Methods for Cyber Test and Evaluation

V. Berzins

The views presented is this paper are those of the author and do not necessarily represent the views of DoD or its Components.

Cyber Testing Challenges

- ICD 503: Manage Risk
- Paradigm Shift: Cyber Failures Are Not Random
  - In uncontested environments, failures act like random processes
    - Statistical models of risk apply
    - Goal is to mitigate expected loss
  - In contested environments, adversaries maximize your loss
    - Need game theoretic models of risk
    - Goal is to mitigate worst case loss
- Risk exposure depends on variable circumstances
  - Are we at war?
  - How much profit/military advantage/political value would a successful attack provide to adversaries?
  - Are sufficient resources available for a successful attack?
  - How much risk of prosecution or counterattack is there?

Cyber Testing Challenges

- Causes and Effects Will Be Hidden
  - Rice's theorem: perfect cyber certification is impossible
    - Perfect solution processes will not always terminate
    - Certification must operate within reasonably short bounded time
  - Attacks are designed to make them difficult to find
    - Small footprint - one of a huge number of possible conditions.
    - Fragmentation – interaction of widely separated parts of code,
    - Delayed manifestation – no effect behavior until much later
    - Timing – correct behavior delayed sufficiently causes failures.
    - Parasitic effects – breaking the model of computation so that logically correct source code can produce damaging behavior.
- Consequences are physical
- Threats can morph

Types of Solutions

- Expand scope of risk management
  - Mitigations address both software and adversary
  - Make attacks less profitable / more risky
- Improve software analysis
  - Use software dependencies to find weaknesses
  - Runtime monitoring
- Recover from or mitigate mishaps
  - Self healing and fail safe systems
- Incorporate solutions in **architecture**
  - The part of the system that does not change

Architectural Solutions

- Resiliency via architecture
  - Runtime testing and recovery infrastructure
  - Monitor code/data integrity and physical effects
- Standardized modular security services
  - Authenticated distribution of software updates
  - Runtime monitoring of executable code to detect unauthorized changes
  - Restoring corrupted code
  - Restoring execution state to a valid configuration
  - Resuming execution with restored code

Insider Threats - Turn-Key Malware

- Statistically invisible = impossible to detect by black box testing
- Clear box testing can do better
  - Use constraint solvers to synthesize test inputs for majority of cases

Outsider Threats – Runtime Code Modification

- **Static and Dynamic Detection**
  - Software update service analysis
  - Architecture conformance checking
  - Memory allocation checking
  - Memory reference checking
  - Runtime monitoring of executable code
  - Runtime monitoring of data integrity constraints
  - Runtime monitoring of physical states

Outsider Threats – Runtime Code Modification

- Mitigations for defense in depth
  - Using pure code segments in read-only hardware
  - Restoration of code from ROM
  - Disabling reflective language capabilities
  - Use garbage collecting programming languages to reduce hazards of code and data corruption
  - Intensively analyze memory allocation and recycling facilities for memory corruption hazards
    - compilers, runtime libraries, linkers, loaders, etc.
  - OS and hardware level memory protection

Architecture Testability Levels

	Level	Testability Level Description
0	inadequate	Does not meet requirements for any of the higher levels
1	syntactic	All services and data elements provided by each procurable component have published interfaces/data models that provide names and type signatures.
2	semantic	Published interfaces include precise definitions of the meaning of the services/data, including units, connection to real world objects, and requirements on outputs and final states resulting from all services
3	robust	Published interfaces include all assumptions and restrictions on inputs and states, triggering conditions for all exceptions, and expected results after exceptions
4	observable	All system attributes relevant to checking the requirements are observable either via the published operational interfaces or published augmented testing interfaces
5	measurable	All properties needed to check the requirements have clearly defined measurement and evaluation procedures
6	decidable	Pass/fail decisions for all test cases can be made entirely by automated procedures, without need for subjective human judgment
7	unbounded	Any number of random test inputs can be automatically generated and corresponding test results can be automatically checked for all services

QA for Architectures

- QA for architectures should assess their testability levels
  - Levels 5-7 appropriate for secure architectures
- Testability levels 6 and 7 can be augmented with continuous Built-in-Test capabilities
  - Enables checking system integrity in the field
  - Corrupted software: e.g. re-image OS
  - Prognostics: e.g. replace battery soon
  - Device failure: e.g. replace hard drive
- Conform to a TRF for code integrity services

Conclusions

- No silver bullet for cyber security.
- Best practical solutions integrate a layered set of defenses and mitigations
- Need runtime monitoring / recovery in addition to static analysis and dynamic testing
- Security QA procedures for architectures should be part of OSA processes.

Recommendations

- Increase the time and effort it will take an adversary to compromise our systems.
  - Make countermeasures part of OSA/TRF.
- Decrease the time to detect a compromise and restore dependable operation.
  - Runtime monitoring and self-healing.
- Make system compromise prohibitively expensive for potential attackers.
  - Integrate software, hardware, network, legal, political, and military countermeasures.

Thank you