Xenon: High-Assurance

Xen

John McDermott
John.McDermott@NRL.Navy.Mil
Naval Research Laboratory
Center for High-Assurance Computer Systems
http://chacs.nrl.navy.mil
Beyond Buffer Overflows

* **Policy flaws**
  * Use the wrong product
  * Mis-configure the right product

* **Design flaws**
  * Majority of flaws are design flaws
  * Can be interface or architecture problems

* **Coding flaws**
  * e.g. buffer overflows
Beyond Assurance: Robustness

* NSA originated this useful concept
* Robustness = (strength of feature, implementation assurance)
* Assurance = how well did we build it?
* Strength = what flaws would be present, even if we had a perfect implementation?

It is pointless to build a high-assurance implementation of a low-strength feature.
Common Criteria

1. Define the security problem your product will solve.

2. By selecting from a framework of security requirements, define a security solution.

3. Choose a pre-defined assurance level.

4. Undergo independent evaluation to show that your product solves the problem, at the claimed level of assurance.
Independent Evaluation

- Actual evaluation is a contact sport.
- Lots of communication needed.
- Evaluator-developer relationship management.
- Following high-assurance practices without evaluation is beneficial, with much less pain.
- Actual evaluation is still possible.
Assurance Levels (EALs)

* Low (1-4):
  * Accepted internationally.
  * Does not review all source code.
  * No special security practices.
Assurance Levels (EALs)

- High (5-7):
  - Not accepted internationally.
  - Few examples.
  - Requires special high-assurance security development practices.
What is Suited to High-Assurance?

* Products that do not evolve rapidly.
* Products with a relatively small implementation.
* Products that are effective at key points in a larger architecture.
* Products that are strong mechanisms.
What security problem does a VMM solve ...

... that cannot be solved by another technology?

Strong separation of execution environments, per user community.

VMM’s are a strong mechanism for this problem.
The VMM Security Problem

What if we have 50 user communities?
Threat Model

- A threat is the goal of some threat actor.

- Four threat actors for Xenon:
  - T1 - malicious developer
  - T2 - malicious guest
  - T3 - network intruder
  - T4 - problematic operator
**T2 - Malicious Guest**

* We don’t care how it got to be malicious.

* **Initial access** - guest boot time access to platform (no human assistance at guest boot time).

* **Initial knowledge** - own configuration data, human sponsor has full source of guests and Xen.

* **Capabilities** - arbitrary sequences of instructions and hypercalls
Actor T2 Threats

* **T2.1 Unauthorized access:** access or cause another guest to access a resource contrary to configured policy.

* **T2.2 Service Denial:** degrade a resource or its availability to another guest.

* **T2.3 Information Leak:** leak information to another domain contrary to configured policy (may use residual data or covert storage channel).
Xenon high-assurance Xen

High-Assurance Work Products

- Security problem definition
- Assurance argument
- Security factored code base
- Policy-to-code modeling
- Model-based vulnerability analysis
- Evidence package for third-party evaluation.
Assurance Argument

* Shows why the final product should be trusted.

* Documented organization of evidence: (factoring, modeling, analysis, etc.)

* Allows planning and trade-offs in allocating resources to assurance tasks.
Xenon high-assurance Xen

Security Problem Definition

- Threats
- Regulations
- Assumptions about usage & environment
- Security policy that solves the problem

- Security features that enforce the policy
- Assurance plan
- Rationale connecting all of the parts
Xenon
high-assurance
Xen

Security Factored Code Base

* Refactor to meet **complexity** goals.
* A lot of Xen code is already there
* Refactor to meet **modularity** goals.
* Refactor to **separate** policy-enforcing code from other code.
* A lot of Xen code is already there
* Remove code/features to **reduce overall size**.
Policy-to-Code Modeling

* Security policy model (formal)
* Interface model (semi-formal)
* Design model (semi-formal)
* Must model all code that runs in the same address space
* Backward correspondence demonstration
Things Xen May Want to Do

* Keep writing small cohesive low-complexity functions.
* Maintain good high-level design.
* Strive for smaller files with simpler includes.
* Don’t spread concerns across multiple files.
* Don’t optimize just because you can.
* Never use `goto` when `break` or `continue` will do; never use `break` when `return` will do.
Things We Do for High Assurance

- Break up big modules into smaller modules.
- Apply secrets-oriented design rules.
- Change macros to inlines.
- Modify logic for case completeness.
- Remove optimization where it is not needed.
- Only support one kind of hardware.
- Sacrifice features to get security.
- Sacrifice features to get assurance.
Possible Open Community Process?

* Separate code & evidence base for high-assurance Xen?
  * What will be the minimal requirement for such code and evidence base?
  * Who will approve code & evidence?
  * How to keep up with main stream Xen?
Family Approach?

* Design Xen to have two family members:
  * Strong-security Xen with a simpler hypervisor.
  * Feature-rich Xen that adds/replaces modules of strong-security Xen.
Thank You