Xenon Assurance Modifications to Xen Code

John McDermott, Naval Research Lab
Prologue: Security

- Strength of mechanism - any conceptual flaws?
- Assurance - any construction flaws?
Prologue: Security

* Security is always defined with respect to a threat model.

* A threat model has
  * threat
  * threat actor
  * capability
  * initial access
  * initial knowledge
The Xenon Project

Project Goals:

* What happens when you try to develop high assurance open source?

* Can you build a “separation kernel” that supports unmodified uninterpreted off-the-shelf software on commodity hardware (e.g. x86_64 with APIC)?
Xenon

A “separation hypervisor”

Not a separation kernel

Not a “secure” replacement for Xen

A new security-oriented product for specialized use

See the Apr. ‘07 slides at www.xen.org/xensummit
The Xenon Prototype

- Based on 3.1
- Some features removed
- Only supports x86_64
- Refactored code
- Evidence package
Simplicity

* McCabe cyclomatic complexity
* Halstead effort (sanity check)
* Size (sanity check)
* Xen 3.1 has approx. 3810 C functions
* Approx. 200 will have to be re-factored
* x86_emulate.c?
Simplicity: _evtchn_close

* _evtchn_close - 34
* _evtchn_close - 3
* _finish - 5
* _chk_state - 3
* _state_INTERDOMAIN - 17
* _state_VIRQ - 3
* _state_PIRQ - 2
* _get_port - 6
* _try_again - 3
* _init_state - 1
Modularity (cohesion)

Xen

PageAlloc

Xenon

PageAlloc

Map

XenHeap

Config

BootTime

DomHeap

Trigger

RunTime

Scrub
Encapsulating Global Data

* No global data allowed, e.g.
  * v->vcpu_id

* Information hiding instead, e.g.
  * loose: *define vcpu_id(v)
  * strict: static inline int vcpu_id
* get/set_vcpu_id
* larger but just as fast
* object size: 158744b vs. 157240b
* sort test: 0.121s vs. 0.122s
* search test: 0.836s vs. 0.854s
Xenon Construction Guidelines

* comments
* PDL files
* readme files
* tool-based formatting
* complexity, modularity, abstraction limits
* logical completeness
* coding practices
Guidelines: Comments

* minimal

* don’t comment anything that could be understood in a first reading of the source

* assume the reader has a modern, program-slicing based source code browser, e.g. CodeSurfer
Guidelines: PDL

* each source or header file has a companion Pseudocode Design Language file, e.g. `dom_event_channel_c.pdl`

* explains the **intent** of each line of code

* does not explain coding, algorithms or data structures

* see McConnel’s Code Complete, 2nd ed.
Guidelines: readme file

* each source, header, and assembler file has a companion “readme” file.

* everything not covered by the .pdl file
  
  * todo’s,

  * data structures (e.g. event channel buckets) and algorithms,

  * coding techniques (e.g. container_of)

  * workarounds, etc...
**Guidelines: formatting**

- All files formatted by a tool
- Public Xenon format is defined as GNU indent
  - `indent -kr -nut -i8 -l 28 file.c`
- Everyone can re-format to their preferred style for local work
Guidelines: structural limits

- complexity, modularity, and abstraction limits
- examples shown in earlier slides
- departures from these limits explained in the `.readme` file, i.e. the code is more complex, less modular on purpose
suppose code has conditions \((x < y)\) and \((v->task)\)

what about \((x \geq y)\) and \((v->task == NULL)\)?

should be dealt with in the code, covered by an assertion, or explained in the .readme file
Guidelines: coding practices

* how to avoid high-risk or low-assurance C coding techniques?
* could have used a safe subset of C, e.g. MISRA C,
* subsets can be “noisy” and irritating
* instead, adopt 18 high-assurance coding rules, mostly from Les Hatton
More Information

- email: john.mcdermott@nrl.navy.mil (Xenon)
- http://www.grammatech.com (CodeSurfer)