

Hypervisor Verification Project

Center for Information Assurance and Security
University of Texas at Austin

November 29, 2011

Encapsulation via Virtualization

Virtual Machines provide a path toward software encapsulation, decoupling software from the underlying hardware by introducing a level of indirection.

- VMM controls how software utilizes hardware resources;
- offers encapsulation of the software state;
- permits isolation of different virtual machines;
- mediates communication between virtual machines;
- potentially protects software integrity;
- aids *portability*—subsystems can run on any platform that supports the VMM;
- enhances *certifiability*—isolation and communication properties can be verified once and leveraged across applications.

The work we are currently doing is predicated on the following:

- 1 Virtualization will continue to grow in importance as a mechanism for enhancing the flexibility, efficiency, and security of computing systems.
- 2 The reliability and security benefits that virtualization promises depend on certain guarantees of data protection and operational isolation of virtual machines.
- 3 Providing high assurance of such properties requires a rigorous formal analysis, stretching from the desired security properties at the top, down to the physical hardware and implementation code at the bottom.

The Payoff

- Security guarantees are provided *by the hypervisor*, not by the guest operating system.
- From the certifier's perspective, if the hypervisor is reliable, then we gain the security guarantees *regardless of the reliability of the guest software*.
- The Virtual Machine can protect the machine from a malicious guest.
- The hypervisor may be accessible to verification—a few thousand lines of code, compared to millions for the guest OS.

A Verified Hypervisor

How do we obtain a highly reliable hypervisor? Two options:

- 1 Verify an existing hypervisor;
- 2 Build and verify one from scratch.

Commercial-scale hypervisors (e.g., Xen and VMWare) are extremely large and complex, involving millions of lines of code, and some are proprietary.

Small research hypervisors (e.g., SecVisor, Nova) are small and potentially accessible to formal analysis.

- What assurances are provided?
- Is the design/implementation correct?
- What is the verification engine?
- Does the implementation language have a formal semantics?
- Are there formal models for the target platform?

The Bottom-Up Approach

An alternative is to

- 1 build a small, robust hypervisor that ensures isolation;
- 2 verify its correctness via rigorous formal methods.

This is a significant research effort:

- Will it be small enough to verify?
- Will it be big enough to accomplish the goals?
- Is it efficient enough to be useful?
- What is the implementation language?
- What is the verification engine? ACL2? Model checking?
- Can we obtain a formal model of the target platform (e.g., x86 ISA)?

A Possible Path

Rather than view the project monolithically, it's probably better to attack the problem incrementally.

- 1 Build a simple, robust VMM sufficient to host an OS.
- 2 Build a simple VMM that is provably protected from the guest OS.
- 3 VMM can run two guests serially, with separated address spaces.
- 4 Guests can run concurrently.
- 5 Guests can communicate in controlled fashion.
- 6 Guests have more realistic IO.
- 7 And so on...

In the process, we address the serious research challenges and discover how to do this.

In addition, we believe that a small verified hypervisor will be a crucial component of Raytheon's vision of a *highly assured glovebox* for use in the National CyberRange.

A glovebox would allow execution of arbitrary (potentially malicious) code in a controlled environment with guarantees that:

- the code could not damage the underlying machine;
- the user can always reliably and consistently regain control of the machine.

This is actually quite challenging on modern machines and will require considerable research.

Thus far we have:

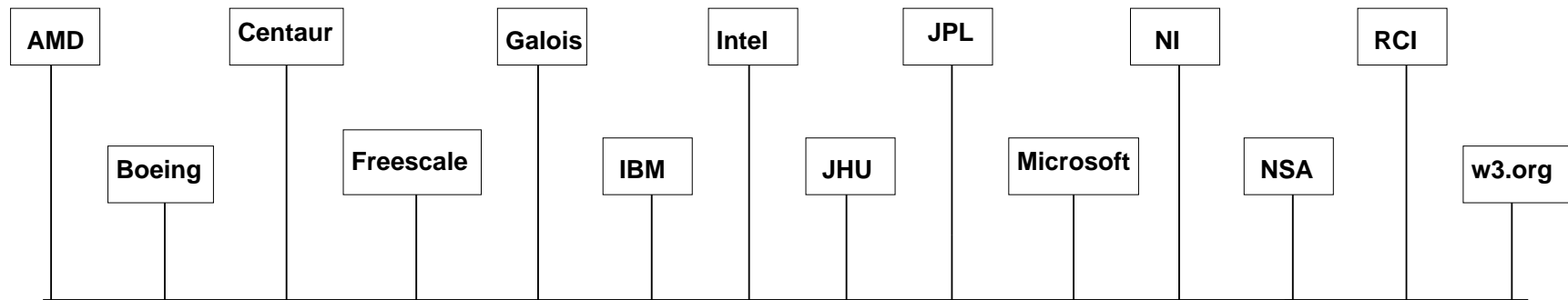
- Developed a small hypervisor, called MinVisor, adequate for booting Linux
- Developed an understanding of the initial state and invariants necessary to provide its assurance guarantees
- Begun modeling this in the ACL2 logic
- Developed within ACL2 a small subset of the ISA of the X86 processor, and proven some critical MinVisor code
- Developed an initial high-level model of correctness with respect to the MinVisor invariants

ACL2 stands for *A Computational Logic for Applicative Common Lisp* developed by Boyer, Kaufmann, and Moore

- a first-order logic with mathematical induction;
- a functional programming language;
- a computer program based on these, which can prove and organize theorems.

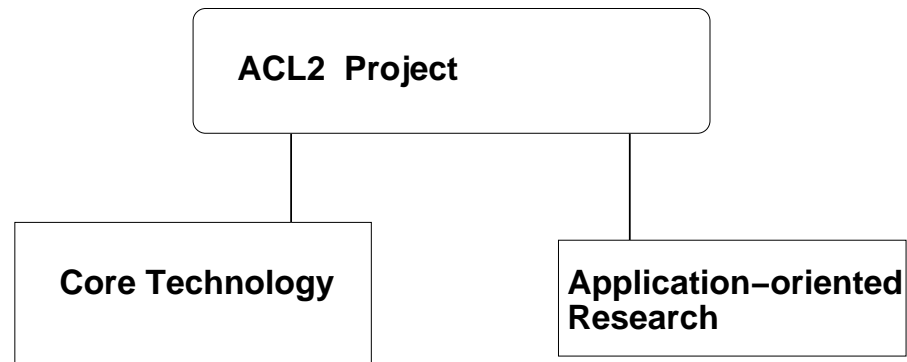
The ACL2 sources are 8.4M code, programmed primarily in itself.

Use of ACL2



Customers

Our Program



Challenges

- Hypervisors interacts directly with the hardware, which means that a high-fidelity model of the ISA is desirable
- Subsetting the ISA needs to be done with care if the proof is to be “meaningful”
- Software can damage a modern processor, including hardware and bios; a glovebox design would have to identify and contain such effects
- MinVisor is simple relative to commodity hypervisors, but still a fairly complex artifact
- Managing the proof and managing the distributed proof effort introduces challenges
- Device drivers will be a real challenge

This effort is joint work by:

- Dr. Mike Dahlin
- Dr. Bill Young
- Deepak Goel
- Ryan Johnson
- Robert Krug
- Michael McCoyd

Related work is being done on the formalization of the X86 ISA and efficient proofs of X86 programs by:

- Dr. Warren Hunt
- Dr. J Moore