Nizza — a trustworthy, secure, open, and verifiable platform

Hendrik Tews
SoS group, Radboud University Nijmegen
http://www.cs.ru.nl/~tews

Supported by the European Union through PASR grant 104600.

Objective: Create an open robust computing platform

Enjoy the latest bells and whistles of the internet.
Without having to worry about the security of online banking.

4 Partners:

- Technical University Dresden (Germany)

  Development/Implementation of the open robust computing infrastructure
- Radboud University Nijmegen

  Formal methods: specification and verification of some parts
- Secunet Security Networks AG (Germany)

  Case study
- ST Microelectronics (France)

  Port the platform to an embedded system (PDA)

Sponsored by the EU through PASR

Conflict between Security and Usebility

- mobile phone/PDA
  - mobile webbrowser
  - store personal data, used for monetary transactions
- PC at home
  - Internetbanking, private correspondence
  - Internet access console

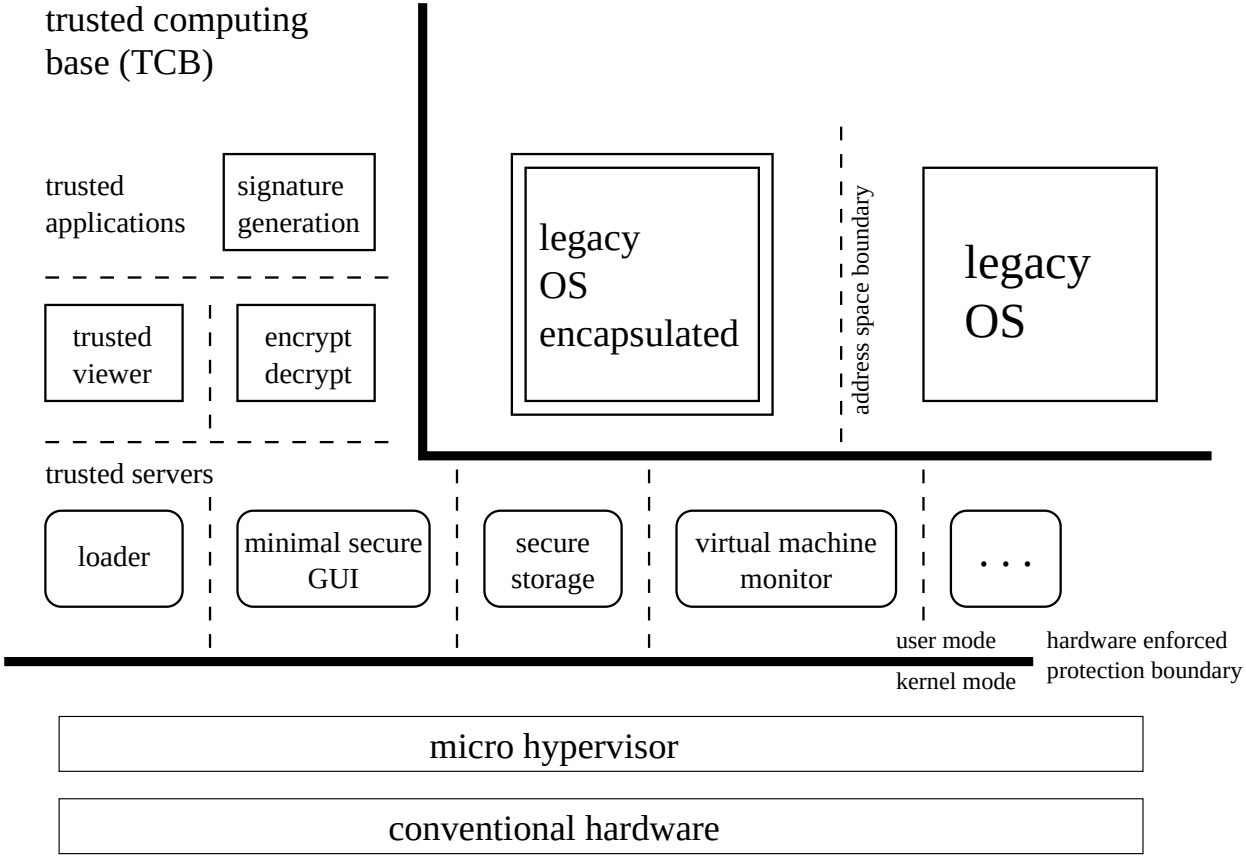
Security considerations:

- closed system
- minimal software

Usability considerations:

- supported OS with large application base (Windows, Linux)
- freely install/update software (from untrusted sources)

For private use: Disconnect from the internet or give up security.


trusted computing base (TCB)
trusted applications
signature generation
trusted viewer
encrypt decrypt
trusted servers
loader
minimal secure GUI
secure storage
virtual machine monitor
. . .
legacy OS encapsulated
address space boundary
legacy OS
user mode
kernel mode
hardware enforced protection boundary
micro hypervisor
conventional hardware

- Use several OS instances in parallel (web-browser instance, editor instance)
- Every OS instance has only limited access and (typically) cannot access other OS instances
- reboot web-browser instance if contaminated to badly
- editor instance can only talk to the encryption module
- Even if attacker compromises installation media he cannot do anything
- data typed in the editor OS is completely secured,
- trusted viewer protects against trojan horses in the editor instance
- Even most of the hardware can be driven by encapsulated lagacy OS instances
- denial of service attacks are the only problem
  (but it requires an extraordinary attacker to deny service
  for more than a few hours)

Some history

1997 MkLinux: *Linux on the the OSF Mach3 microkernel*, too slow

1997 L4Linux, paravirtualized Linux: *The Performence on micro-kernel-based Systems*
only 5% performance penalty

2003 XenoLinux: *Xen and the Art of Virtualization*

Comparison

L4, L4Linux	Xen
only Linux paravirtualisation, micro-hypervisor providing full virtualization underway	full virtualization
stand-alone application and OS guests	only OS guests
use case: many cooperating modules, RPC	several, mainly independent guest OS'es; no RPC
IPC latency heavily optimised	IPC throughput optimised
device drivers are separated by address space boundaries	Domain 0 controls all devices
sparse micro-kernel interface	rich hypervisor interface
lots of side channels	?

Verification

- treat program as mathematical object
- describe its behaviour in a precise way (semantics)
- prove properties about the behaviour

Verification is different from bug hunting

- verification proves some property and thus the absence of a certain class of errors
- complete verification is very costly and rarely ever performed (in the present)
- use other techniques to eliminate the largest number of bugs with limited resources

Different kinds of semantics

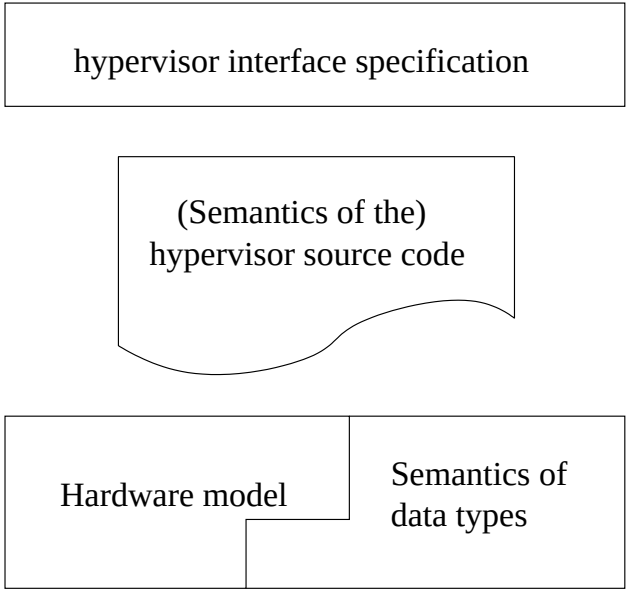
- operational semantics
- axiomatic semantics (Hoare Logic)
- denotational semantics
  - every piece of the program is translated into denotation
  - denotations are functions that capture all the behaviour
  - denotations are composed to get a denotation of the whole program
  - one reasons about the denotations

C++

- OS kernels are typically written in C or C++ enriched with assembly
- standard is very vague (are there negative numbers in signed int?)
- type system is not sound (even without typecasts)

Specifics of kernel Verification

- need type casts (for memory management)
- has to deal with hardware registers that modify the behaviour of the CPU
  - CR3 (page directory base register
  - EFLAGS
  - global descriptor table, interrupt descriptor table
  - task segment
  - feature control register CR0, CR4
- need for assembly (IRET, INVLPG, ...)
- strange programming environment
  - virtual memory, but the same piece of memory might be visible at different addresses
  - virtual memory mapping is manipulated by the kernel itself (even for kernel memory)



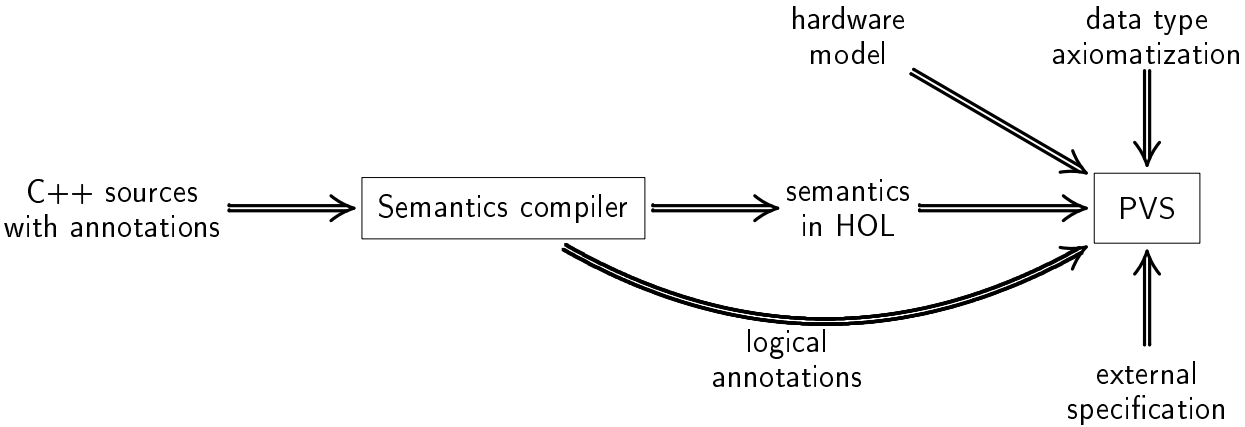
$$\Phi_{data\_types}, \Phi_{hardware} \quad \vdash \quad \varphi(hypervisor)$$

Semantics of data types

- as it says, semantics of `unsigned`, `void *`, ...
- supports type casts in a very modular way
- let verification fail on wrong type casts (or virtual memory errors)

Hardware model

- abstract model of IA32 architecture
- provide basic operations for program semantics (reading/writing typed variables in virtual memory)
- models protected mode with paging enabled (including all details that might cause programming errors, such as the TLB)
- don't model unnecessary stuff (virtual x86 mode, physical address extension, ...)
- however, do monitor all relevant flags and switches (let the verification fail if, e.g., paging is disabled)
- use the semantics of data types for hardware data types (such as page directory entries)
- stricter check for nonsense/errors than the real hardware (e.g., fail when a string is encountered in the page directory)



- PVS: an interactive theorem prover for higher-order logic
- semantics compiler translates C++ sources into PVS
- program semantics "*is evaluated*" on top of the hardware and the data type model
- verification goals are handwritten or included in the sources as special annotations

Goals that we would like to attempt

- absence of the following hardware errors
  - reserved bit violations
  - accessing features not present in the model (such as physical address extension)
  - TLB inconsistency
  - unaligned access to memory mapped hardware devices (such as the Advanced Programmable Interrupt Controller)
- dynamic type correctness
- only kernel code runs in kernel mode

Goals currently out of reach

- address space separation
- attacker does not get access to data in a different address space

- Nizza architecture solves conflict between security and usability
- verification of the underlying micro-hypervisor is tackled at Radboud University
- use denotational semantics of (a subset of) C++
  to prove simple correctness properties