OS Security

Virtualization

Radboud University Nijmegen, The Netherlands

Winter 2015/2016

Announcement

- No lecture on January 5, 2016
  - Werkcollege will take place as usual (Wednesday, January 6)
- Next lecture will be on January 12
- Enjoy the holidays!
  - Post-Snowden Crypto workshop[1], Dec 9-10, Brussels
  - 32C3[2], Dec 27-30, Hamburg

[1] https://hyperelliptic.org/PSC/
[2] https://events.ccc.de/congress/2015/wiki/Main_Page

A short recap

- Last 2 lectures: Malware
- Malware evolution from PC to smartphone
- Early days: malware targeting Symbian OS users (`Cabir`, `Pbstealer`)
- Popular smartphone platforms affected
  - Android: first proof-of-concept malware released in 2008
  - iOS: `WireLurker`
  - Windows Phone: `FinSpy Mobile`
  - Blackberry: Trojans using the 'Backstab' technique
- Intrusion detection system
  - NIDS, HIDS
  - NIDS: (i) string, (ii) port and (iii) header condition signatures
  - HIDS: signature- and behaviour-based
- Intrusion prevention system
  - NIPS, HIPS
  - (i) signature-based detection, (ii) anomaly-based detection and (iii) protocol state analysis detection

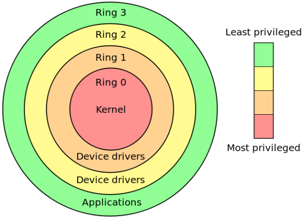
Role of the OS

- A major job of the OS is to enforce **protection**
- Prevent malicious (or buggy) programs from:
  - Allocating too many resources (denial of service)
  - Corrupting or overwriting shared resources (files, shared memory,...)
- Prevent different users, groups, etc. from:
  - Accessing or modifying private state (files, shared memory,...)
  - Killing each other's processes
- Prevent viruses, worms, etc. from exploiting security holes in the OS
  - Overrunning a memory buffer in the kernel can give a non-root process root privileges
- *How does the OS enforce protection boundaries?*
  - 2-level protection: kernel and user mode
  - Multilevel protection: Ring 0-3

Kernel and User mode

- What makes the kernel different from user mode?
  - Kernel can execute special *privileged instructions*
- Examples of privileged instructions are:
  - Access to I/O devices
  - Manipulate memory management: set up page tables, load/flush the CPU cache, etc
  - Call halt instruction: put CPU into low-power or idle state until next interrupt

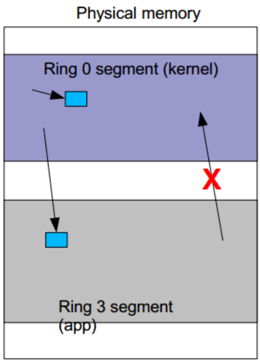
Multilevel Protection: Ring 0-3



- **Ring 0**: kernel
- **Rings 1-2**: third-party drivers (less privileged OS code)
- **Ring 3**: application code

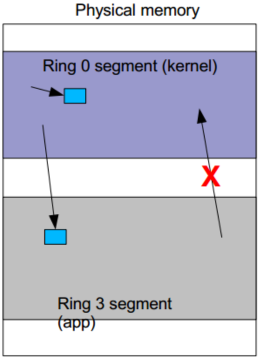
More on Protection Rings - I

- Each memory segment has an associated privilege level (0 through 3)
- The CPU has a Current Protection Level (CPL)
  -> Usually the privilege level of the segment where the program's instructions are being read from



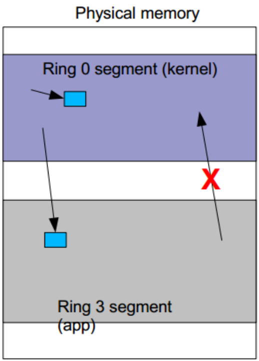
More on Protection Rings - I

- Each memory segment has an associated privilege level (0 through 3)
- The CPU has a Current Protection Level (CPL)
  -> Usually the privilege level of the segment where the program's instructions are being read from
- Program can read/write data in segments of *lower privilege* than CPL
  -> e.g. Kernel can read/write user memory



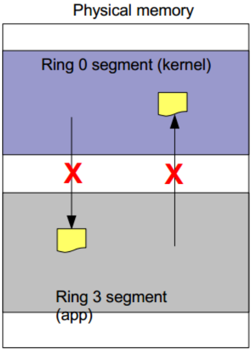
More on Protection Rings - I

- Each memory segment has an associated privilege level (0 through 3)
- The CPU has a Current Protection Level (CPL)
  -> Usually the privilege level of the segment where the program's instructions are being read from
- Program can read/write data in segments of *lower privilege* than CPL
  -> e.g. Kernel can read/write user memory
  -> But user cannot read/write kernel memory.... Why?



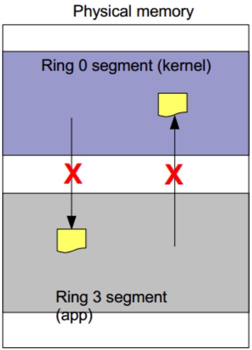
More on Protection Rings - II

- Each memory segment has an associated privilege level (0 through 3)
- The CPU has a Current Protection Level (CPL)
    -> Usually the privilege level of the segment where the program's instructions are being read from
- Program cannot (directly) call code in *higher privilege* segments
    -> Why?



More on Protection Rings - II

- Each memory segment has an associated privilege level (0 through 3)
- The CPU has a Current Protection Level (CPL)
  -> Usually the privilege level of the segment where the program's instructions are being read from
- Program cannot (directly) call code in *higher privilege* segments
  -> Why?
- Program cannot (directly) call code in *lower privilege* segments
  -> Why?



Types of Virtualization

- OS-level virtualization
- Application level virtualization
- Full/native virtualization
- Paravirtualization
- Emulation

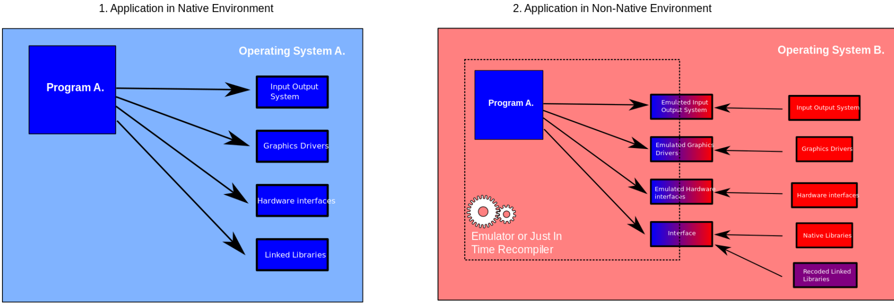
OS-level virtualization

- OS allows multiple secure virtual servers to be run
- Makes the subsystem thinks it is running in its own operating system
- Abstracts the services and kernel from an application
- Guest OS is the same as the host OS, but appears isolated; apps see an isolated OS
- For example: Solaris Containers, FreeBSD Jails, Linux Vserver

Application level virtualization

- Application behaves at runtime in a similar way when directly interfacing with the original OS
- Application is gives its own copy of components that are not shared
- For instance: own registry files, global objects
- Application virtualization layer replaces part of the runtime environment normally provided by the OS
- Example: Java Virtual Machine (JVM)

Application level virtualization



Full/native virtualization

- VM simulates “enough" hardware to allow an unmodified guest OS to be run in isolation
- Any software capable of execution on the hardware can be run in the virtual machine
- Example: VMWare Workstation/Server, Mac-on-Linux etc.
- Challenge: Interception and simulation of privileged operations (I/O operations)
- Every operation performed within a given virtual machine must be kept within that virtual machine; virtual operations cannot be allowed to alter the state of any other virtual machine, control program or hardware.

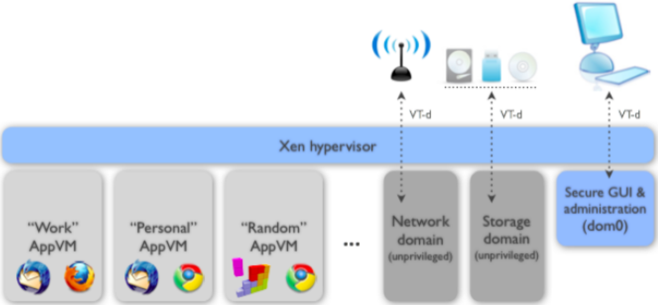
Paravirtualization

- VM does not simulate hardware
- Is a technique that presents a software interface to VMs that is similar but not identical to that of the underlying hardware
- Use special API (para-API) that a modified guest OS must use
- Hypercalls trapped by the Hypervisor and serviced
- Provides specially defined 'hooks' to allow the guest(s) and host to request and acknowledge operations, which would otherwise be executed in the virtual domain
- Hence, reduces the portion of the guest's execution time spent performing operations which are substantially more difficult to run in a virtual environment compared to a non-virtualized environment
- For example: Xen, VMWare ESX Server

Emulation

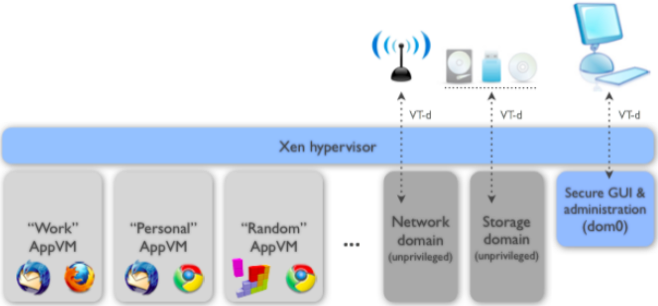
- VM emulates complete hardware and software
- Emulator is a hardware/software enabling a system (i.e. host) to behave like another system (i.e. guest)
- Unmodified guest OS for a different system can be run
- Useful for reverse engineering, malware analysis, forensics (taint tracking)
- For example: QEMU, VirtualPC for Mac, Android

Qubes OS



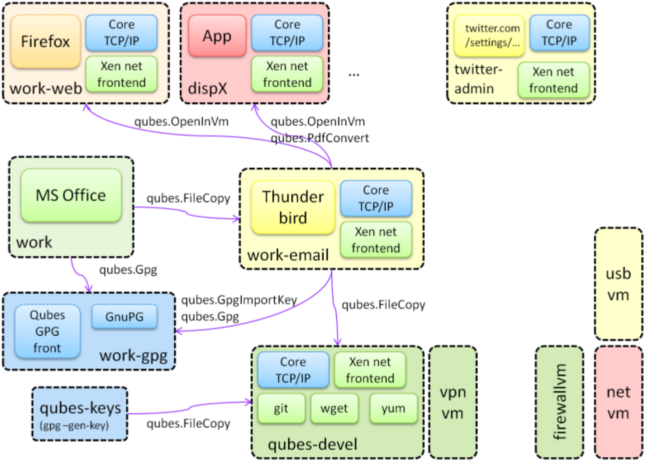
- Based on a secure bare-metal hypervisor (Xen)
- Networking code sandboxed in an unprivileged VM (using IOMMU/VT-d)
- USB stacks and drivers sandboxed in an unprivileged VM
- No networking code in the privileged domain (dom0)

Qubes OS

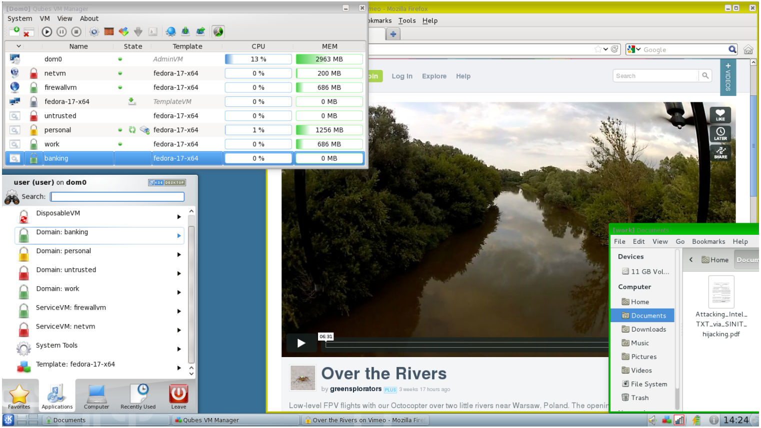


- All user applications run in "AppVMs," lightweight VMs based on Linux
- Centralized updates of all AppVMs based on the same template
- Qubes GUI virtualization presents applications as if they were running locally
- Qubes GUI provides isolation between apps sharing the same desktop
- Secure system boot

Compartmentalization in Qubes OS



Qubes OS Live



TUDOS - TU Dresden OS

- Demo
- Can be downloaded from:
  http://demo.tudos.org/eng_download.html

VM Vulnerabilities

- Hardware oriented attacks
- Management interface exploits
- Break out of jail attacks (VM escape)
- Virtual-machine based rootkits (Blue Pill)
- Application privilege escalation
- JIT spraying
- Untrusted native code execution