Computer Security

05. Confinement

Paul Krzyzanowski

Rutgers University

Spring 2017

Last Time

- *chroot*
- FreeBSD Jails
- Linux namespaces, capabilities, and control groups
  - Control groups
    - Allow processes to be grouped together – control resources for the group
  - Capabilities
    - Limit what root can do for a process & its children
  - Namespaces
    - Restrict what a process can see & who it can interact with:
      PIDs, User IDs, mount points, IPC, network

Containers

Containers

- Primary goal is software distribution, not security
  - Makes moving & running a collection of software simple
    - E.g., Docker Container Format
  - E.g., everything at Google runs in a container
    - Over 2 billion containers started per week (2014)
    - lmctfy ("*Let Me Contain That For You*")
    - Google's container tool – similar to Docker and LXC (Linux Containers
- But
  - Containers use namespaces, cgroups, & capabilities
    - Restricted capabilities by default
  - They separate policy from enforcement
  - Watchdog-based restarting: helps with availability
  - Helps with comprehension errors
    - Decent default security without learning much
    - Also ability to enable other security modules

Sandboxes

The sandbox

sand•box, ’san(d)-"bäks, *noun*. Date: 1688
: a box or receptacle containing loose sand: as **a:** a shaker for sprinkling sand on wet ink **b:** a box that contains sand for children to play in

- A restricted area where code can play in
- Allow users to download and execute untrusted applications with limited risk
- Restrictions can be placed on what an application is allowed to do in its sandbox
- Untrusted applications can execute in a trusted environment

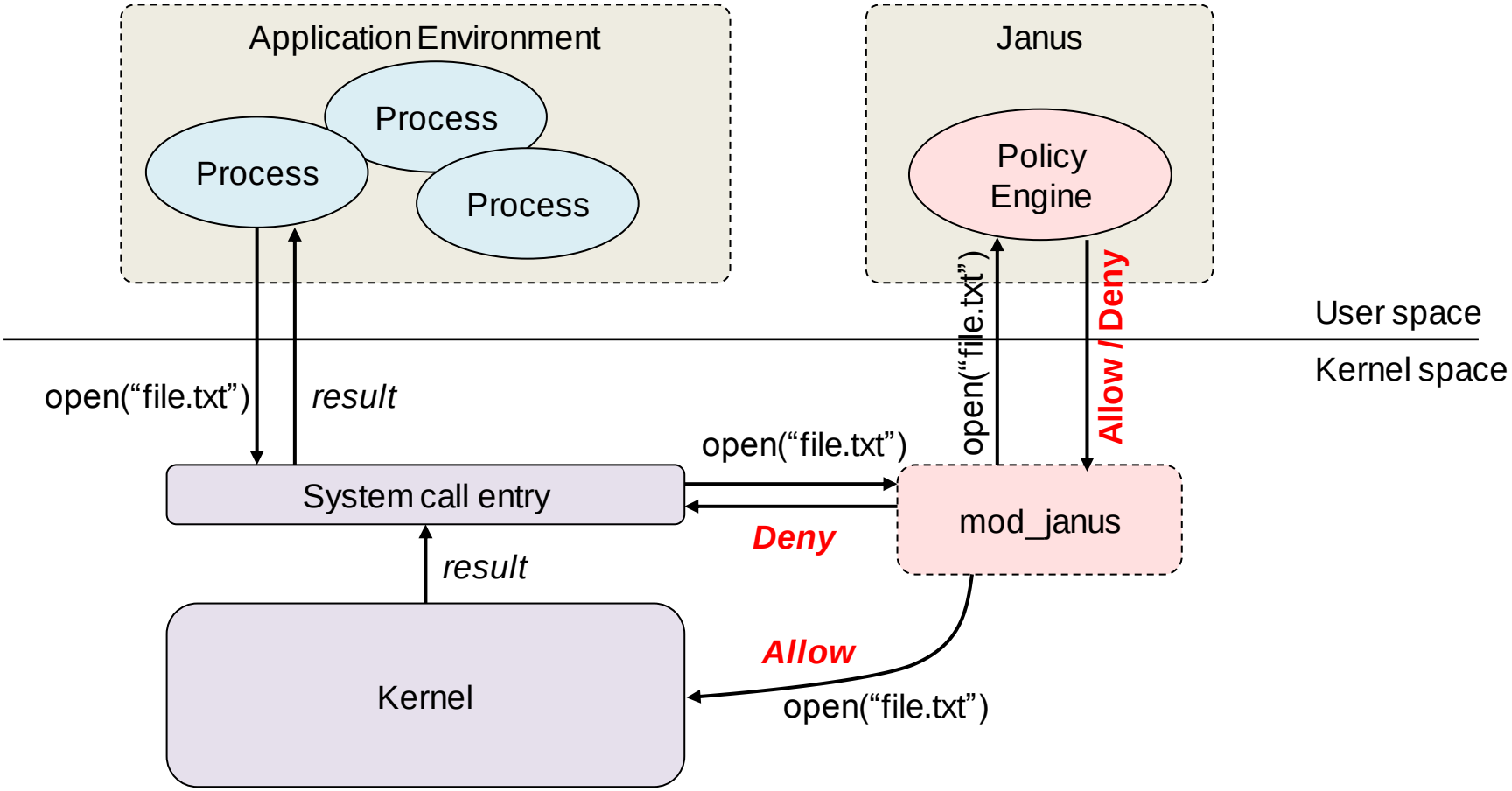
Jails & containers are a form of sandboxing
… but we want to focus on giving users the ability to run apps

System Call Interposition

- System calls interface with resources
  - An application must use system calls to access any resources, initiate attacks … and cause any damage
    - Modify/access files/devices: *creat, open, read, write, unlink, chown, chgrp, chmod, …*
    - Access the network: *socket, bind, connect, send, recv*
- Interposition
  - Intercept & inspect an app's system calls

Example: Janus

App sandboxing tool implemented as a loadable kernel module



Example: Janus

- Policy file defines allowable files and network operations
- Dedicated policy per process
  - Policy engine reads policy file
  - Forks
  - Child process *execs* application
  - All accesses to resources are screened by Janus
- System call entry points contain *hooks*
  - Redirect control to `mod_Janus`
  - Module tells the user-level Janus process that a system call has been requested
    - Process is blocked
    - Janus process queries the module for details about the call
    - Makes a policy decision

Challenge

Janus has to mirror the state of the operating system

- If process forks, the Janus monitor must fork
- Keep track of the network protocol
  - socket, bind, connect, read/write, shutdown
- Does not know if certain operations failed
- Gets tricky if file descriptors are duplicated
- Remember filename parsing?
  - We have to figure out the whole dot-dot (..) thing!
  - Have to keep track of changes to the current directory too
- App namespace can change if the process does a *chroot*
- What if file descriptors are passed via Unix domain sockets?
  - *sendmsg, recvmsg*
- Race conditions: TOCTTOU

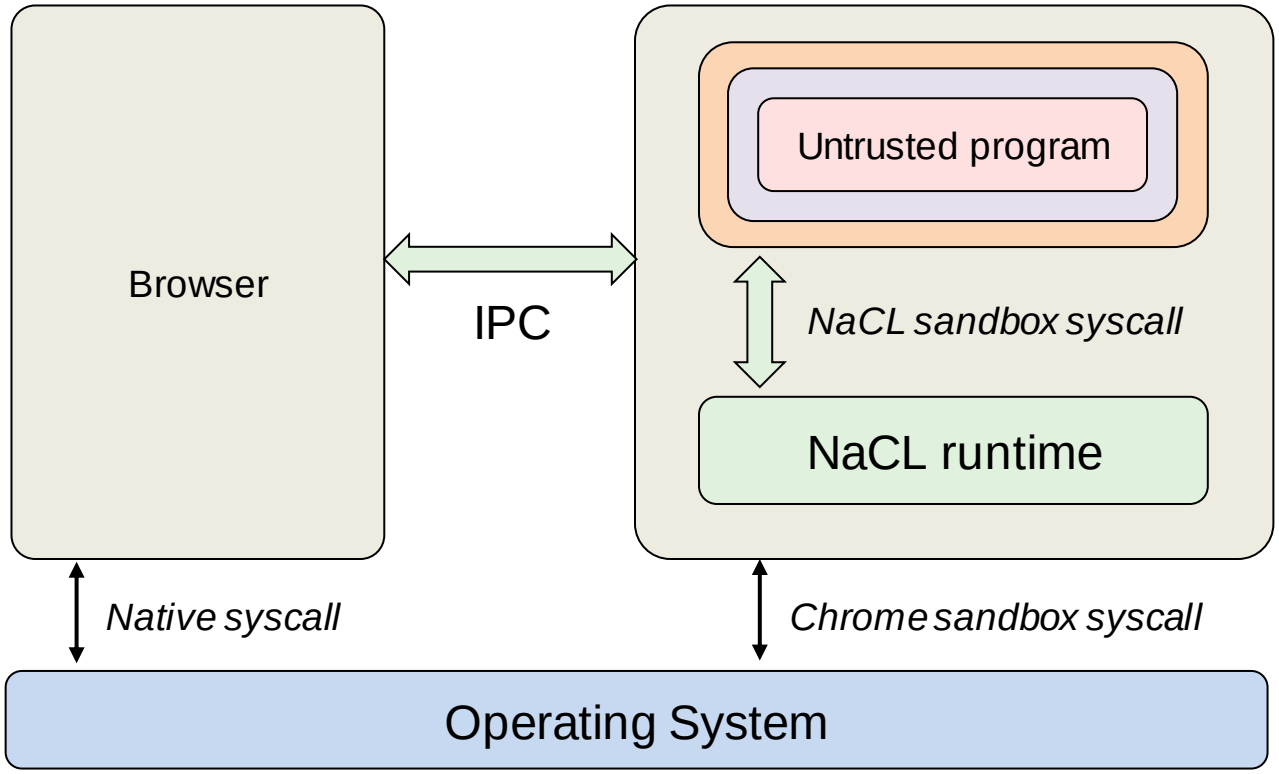
Chromium Native Client (NaCl)

- Designed for
  - Safe execution of platform-independent untrusted native code in a browser
  - Compute-intensive applications
  - Interactive applications that use resources of a client
- Two types of code: trusted & untrusted
  - Untrusted has to run in a sandbox
  - Pepper Plugin API (PPAPI): portability for 2D/3D graphics & audio
- Untrusted native code
  - Built using NaCL SDK or any compiler that follows alignmenr rules and instruction restrictions
  - NaCL statically verifies the code to check for use of privileged instructions

Chromium Native Client (NaCl)

Two sandboxes

- Outer sandbox: restricts capabilities using system call interposition
- Inner sandbox: uses x86 segmentation to isolate memory among apps



Java Language

- Type-safe & easy to use
  - Memory management and range checking
- Designed for an interpreted environment: JVM
- No direct access to system calls

Java Sandbox

1. Bytecode verifier: verifies Java bytecode before it is run
   - Disallow pointer arithmetic
   - Automatic garbage collection
   - Array bounds checking
   - Null reference checking
2. Class loader: determines if an object is allowed to add classes
   - Ensures key parts of the runtime environment are not overwritten
   - Runtime data areas (stacks, bytecodes, heap) are randomly laid out
3. Security manager: enforces *protection domain*
   - Defines the boundaries of the sandbox (file, net, native, etc. access)
   - Consulted before any access to a resource is allowed

JVM Security

- Complex process
- ~20 years of bugs … hope the big ones have been found!
- Buffer overflows found in the C support library
  - C support library buggy in general
- Generally, the JVM is considered insecure
  - But Java in general is pretty secure
    - Array bounds checking, memory management
    - Security manager with access controls
  - Use of native methods allows you to bypass security checks

OS-Level Sandboxes

Example: the Apple Sandbox

- Create a list of rules that is consulted to see if an operation is permitted
- Components:
  - Set of libraries for initializing/configuring policies per process
  - Server for kernel logging
  - Kernel extension using the TrustedBSD API for enforcing individual policies
  - Kernel support extension providing regular expression matching for policy enforcement
- sandbox-exec command & sandbox_init function
  - sandbox-exec: calls *sandbox_init()* before *fork()* and *exec()*
  - `sandbox_init(kSBXProfileNoWrite, SANDBOX_NAMED, errbuf);`

Apple sandbox setup & operation

sandbox_init:

- Convert human-readable policies into a binary format for the kernel
- Policies passed to the kernel to the TrustedBSD subsystem
- TrustedBSD subsystem passes rules to the kernel extension
- Kernel extension installs sandbox profile rules for the current process

Operation: intercept system calls

- System calls hooked by the TrustedBSD layer will pass through Sandbox.kext for policy enforcement
- The extension will consult the list of rules for the current process
- Some rules require pattern matching (e.g., filename pattern)

Apple sandbox policies

Some pre-written profiles:

- Prohibit TCP/IP networking
- Prohibit all networking
- Prohibit file system writes
- Restrict writes to specific locations (e.g., /var/tmp)
- Perform only computation: minimal OS services

Virtual Machines

Virtual CPUs (sort of)

What time-sharing operating systems give us

- Each process feels like it has its own CPU & memory
  - But cannot execute privileged CPU instructions (e.g., modify the MMU or the interval timer, halt the processor, access I/O)
- Illusion created by OS preemption, scheduler, and MMU
- User software has to "ask the OS" to do system-related functions

- Containers, BSD Jails, namespaces give us operating system-level virtualization

Process Virtual Machines

CPU interpreter running as a process

- Pseudo-machine with interpreted instructions
  - 1966: O-code for BCPL
  - 1973: P-code for Pascal
  - 1995: Java Virtual Machine (JIT compilation added)
  - 2002: Microsoft .NET CLR (pre-compilation)
  - 2003: QEMU (dynamic binary translation)
  - 2008: Dalvik VM for Android
  - 2014: Android Runtime (ART) – ahead of time compilation
- Advantage: run anywhere, sandboxing capability
- No ability to even pretend to access the system hardware
  - Just function calls to access system functions
  - Or “generic” hardware

Machine Virtualization

Machine Virtualization

Normally all hardware and I/O managed by one operating system

Machine virtualization

- Abstract (virtualize) control of hardware and I/O from the OS
- Partition a physical computer to act like several real machines
  - Manipulate memory mappings
  - Set system timers
  - Access devices
- Migrate an entire OS & its applications from one machine to another

1972: IBM System 370

- Allow kernel developers to share a computer

Why are VMs popular?

- Wasteful to dedicate a computer to each service
  - Mail, print server, web server, file server, database
- If these services run on a separate computer
  - Configure the OS just for that service
  - Attacks and privilege escalation won't hurt other services

Hypervisor

- Hypervisor: Program in charge of virtualization
  - Aka Virtual Machine Monitor
  - Provides the illusion that the OS has full access to the hardware
  - Arbitrates access to physical resources
  - Presents a set of virtual device interfaces to each host

Machine Virtualization

An OS is just a bunch of code!

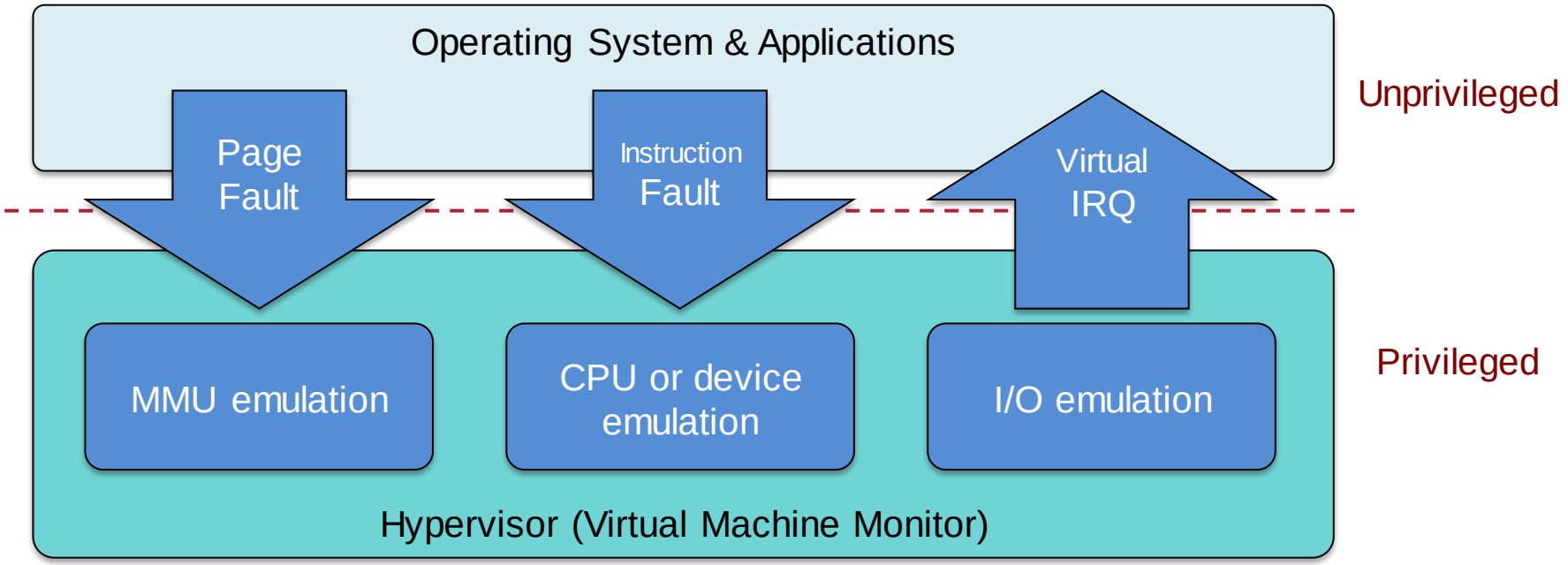
- Privileged vs. unprivileged instructions

- If regular applications execute privileged instructions, they trap
- Operating systems are allowed to execute privileged instructions
- If running kernel code, the VMM catches the trap and emulates the instruction
  - Trap & Emulate

Hypervisor

Application or Guest OS runs until:

- Privileged instruction traps
- System interrupts
- Exceptions (page faults)
- Explicit call: VMCALL (Intel) or VMMCALL (AMD)



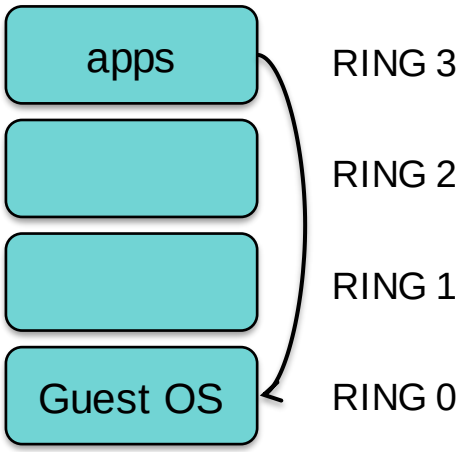
Intel & ARM Didn't Make VM Easy

- Intel/AMD systems prior to Core 2 Duo (2006) did not support trapping privileged instructions
- Most ARM architectures also did not trap on certain privileged instructions
  - Hardware support added in Cortex-A15 (ARMv7 Virtualization Extension): 2011
- Two approaches
  - Binary translation (BT)
    - Scan instruction stream on the fly (when page is loaded) and replace privileged instructions with instructions that work with the virtual hardware (VMware approach)
  - Paravirtualization
    - Don't use non-virtualizable instructions (Xen approach)
    - Invoke hypervisor calls explicitly

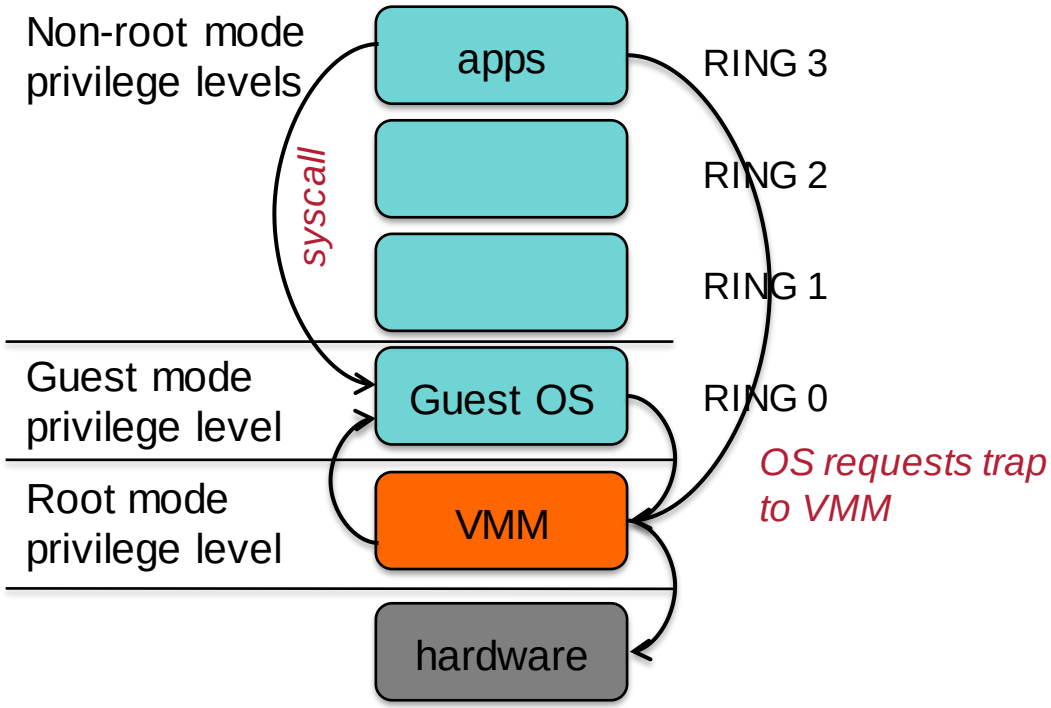
Hardware support for virtualization

Root mode (Intel example)

– Layer of execution more privileged than the kernel



Without virtualization



Architectural Support

- Intel Virtual Technology
- AMD Opteron

Guest mode execution: can run privileged instructions directly

- E.g., a system call does not need to go to the VM
- Certain privileged instructions are intercepted as VM exits to the VMM
- Exceptions, faults, and external interrupts are intercepted as VM exits
- Virtualized exceptions/faults are injected as VM entries

CPU Architectural Support

- Setup
  - Turn VM support on/off
  - Configure what controls VM exits
  - Processor state
    - Saved & restored in guest & host areas
- VM Entry: go from hypervisor to VM
  - Load state from guest area
- VM Exit
  - VM-exit information contains cause of exit
  - Processor state saved in guest area
  - Processor state loaded from host area

Two Approaches to Running VMs

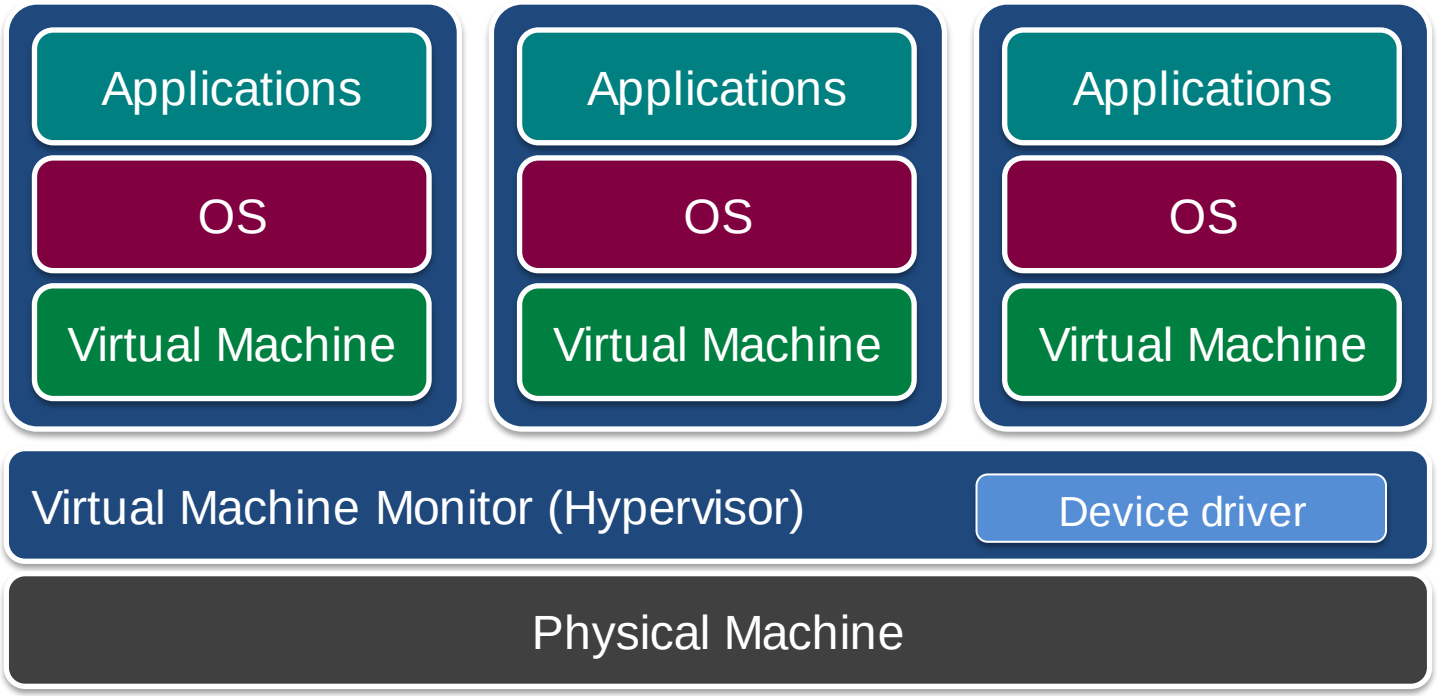
1. Native VM (hypervisor model)
2. Hosted VM

Native Virtual Machine

Example:
VMware ESX

Native VM (or *Type 1* or *Bare Metal*)

- No primary OS
- Hypervisor is in charge of access to the devices and scheduling
- OS runs in “kernel mode” but does not run with full privileges

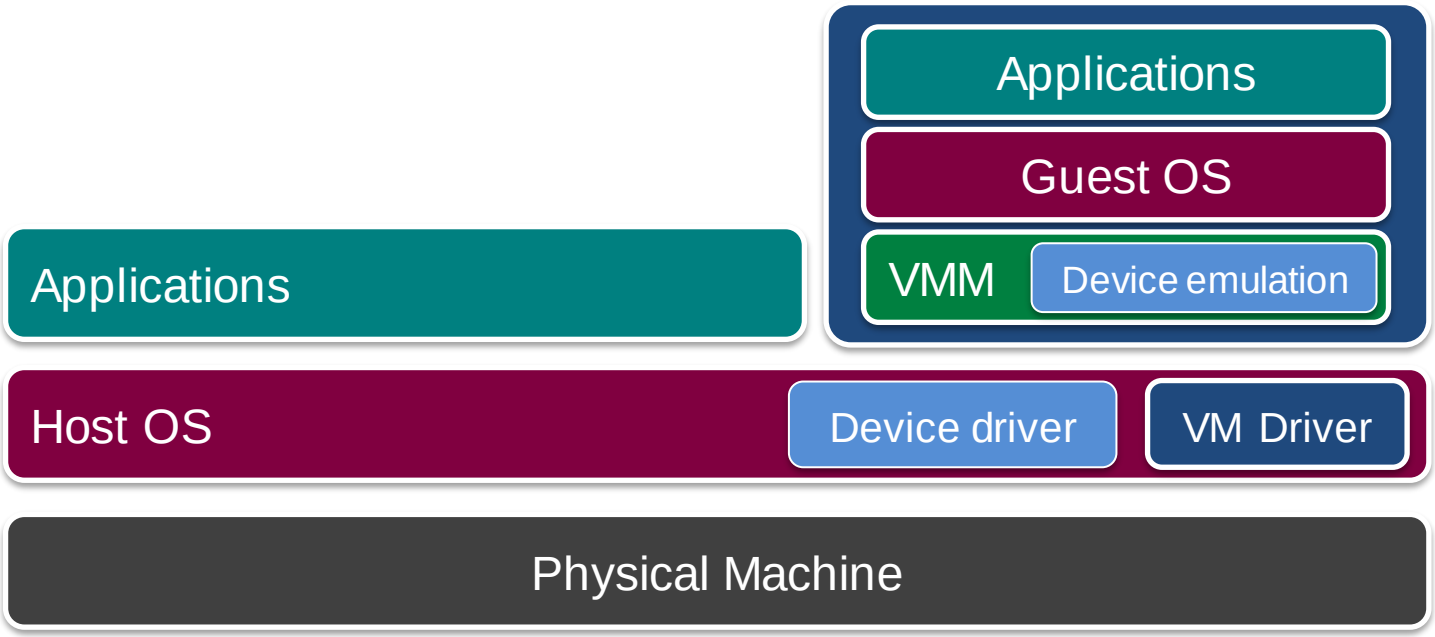


Hosted Virtual Machine

Example: VMware Workstation

Hosted VM

- VMM runs without special privileges
- Primary OS responsible for access to the raw machine
  - Lets you use all the drivers available for that primary OS
- Guest operating systems run under a VMM
- VMM invoked by host OS
  - Serves as a proxy to the host OS for access to devices

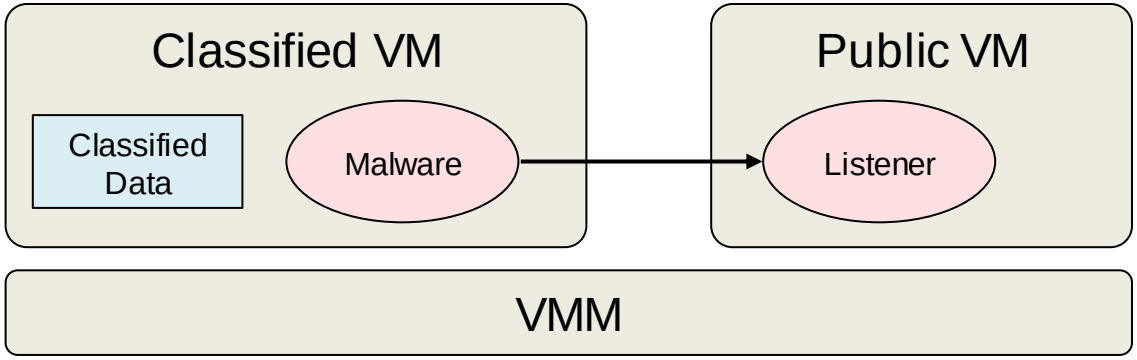


Security Assumptions

- Attacks & malware can target the guest OS & apps
- Malware cannot escape from the infected VM
  - Cannot infect the host OS
  - Cannot infect the VMM
  - Cannot infect other VMs on the same computer

Covert Channels

- Covert channel
  - Secret communication channel between isolated components
- Can be used to leak classified data



1. Malware can perform CPU-intensive task at specific times
2. Listener can do CPU-intensive tasks and measure completion times

This allows malware to send a bit pattern:

malware working = 1 = slowdown on listener

Depends on scheduler but there are other mechanisms too… like memory access

The end