

Computer Security

05. Confinement

Paul Krzyzanowski
Rutgers University
Spring 2018

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 1

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

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 2

Containers

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 3

What's the main problem?

- **Installing software packages can be a pain**
 - Dependencies
- **Running multiple packages on one system can be a pain**
 - Updating a package can update a library or utility another uses
 - Causing something else to break
 - No isolation among packages
 - Something goes awry in one service impacts another
- **Migrating services to another system is a pain**
 - Re-deploy & reconfigure

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 4

How did we address these problems?

- **Sysadmin effort**
 - Service downtime, frustration, redeployment
- **Run every service on a separate system**
 - Mail server, database, web server, app server, ...
 - Expensive! ... and overkill
- **Deploy virtual machines**
 - Kind of like running services on separate systems
 - Each service gets its own instance of the OS and all supporting software
 - Heavyweight approach
 - Time share between operating systems

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 5

What are containers?

Containers: created to package & distribute software

- Focus on services, not end-user apps
- Software systems usually require a bunch of stuff:
 - Libraries, multiple applications, configuration tools, ...
- Container = **image containing the application environment**
 - Can be installed and run on any system

Key insight:
Encapsulate software, configuration, & dependencies into one package

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 6

A container feels like a VM

- **Separate**
 - Process space, network interface, network configuration, libraries, ...
 - Limited root powers
- **But:**
 - All containers on a system share the same OS & kernel modules

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

7

How are containers built?

- **Control groups**
 - Meters & limits on resource use
 - Memory, disk (I/O bandwidth), CPU (set %), network (traffic priority)
- **Namespaces**
 - Isolates what processes can see & access
 - Process IDs, host name, mounted file systems, users, IPC
 - Network interface, routing tables, sockets
- **Capabilities**
 - Keep root access but restrict what it can do
- **Copy on write file system**
 - Instantly create new containers without copying the entire package
 - Storage system tracks changes
- **AppArmor**
 - Pathname-based mandatory access controls
 - Confines programs to a set of listed files & capabilities

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

8

Initially ... Docker

- First super-popular container
- Designed to provide Platform-as-a-Service capabilities
 - Combined Linux cgroups & namespaces into a single easy-to-use package
 - Enabled applications to be deployed consistently anywhere as one package
- **Docker Image**
 - Package containing applications & supporting libraries & files
 - Can be deployed on many environments
- **Make deployment easy**
 - Git-like commands: docker push, docker commit, ...
 - Make it easy to reuse image and track changes
 - Download updates instead of entire images
- **Keep Docker images immutable (read-only)**
 - Run containers by creating a writable layer to temporarily store runtime changes

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

9

Later Docker additions

- Docker Hub: cloud based repository for docker images
- Docker Swarm: deploy multiple containers as one abstraction

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

10

Container Orchestration

- We wanted to manage containers across systems
- Multiple efforts
 - Marathon/Apache Mesos (2014), Kubernetes (2015), Nomad, Docker Swarm, ...
- **Google designed Kubernetes for container orchestration**
 - Google invented Linux control groups
 - Standard deployment interface
 - Scale rapidly (e.g., Pokemon Go)
 - Open source (unlike Docker Swarm)

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

11

Container orchestration

- **Kubernetes orchestration**
 - Handle multiple containers and start each one at the right time
 - Handle storage
 - Deal with hardware and container failure
 - Add remove containers in response to demand
 - Integrates with the Docker engine, which runs the actual container

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

12

Containers & Security

Primary goal was software distribution, not security

- Makes moving & running a collection of software simple
 - E.g., Docker Container Format
- Everything at Google is deployed & runs in a container
 - Over 2 billion containers started per week (2014)
 - **Imctfy** ("Let Me Contain That For You")
 - Google's old container tool - similar to Docker and LXC (Linux Containers)
 - Then Kubernetes to manage multiple containers & their storage

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 13

Containers & Security

- **But there are security benefits**
 - **Containers use namespaces, control groups, & capabilities**
 - Restricted capabilities by default
 - Isolation among containers
 - **Containers are usually minimal and application-specific**
 - Just a few processes
 - Minimal software & libraries
 - Fewer things to attack
 - **They separate policy from enforcement**
 - **Execution environments are reproducible**
 - Easy to inspect how a container is defined
 - Can be tested in multiple environments
 - **Watchdog-based restarting**: helps with availability
 - Containers help with **comprehension errors**
 - Decent default security without learning much
 - Also ability to enable other security modules

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 14

Security Concerns

- **Kernel exploits**
 - All containers share the same kernel
- **Denial of service attacks**
 - If one container can monopolize a resource, others suffer
- **Privilege escalation**
 - Shouldn't happen with capabilities ... But there might be bugs
- **Origin integrity**
 - Where is the container from and has it been tampered?


March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 15

Sandboxes

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 16

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

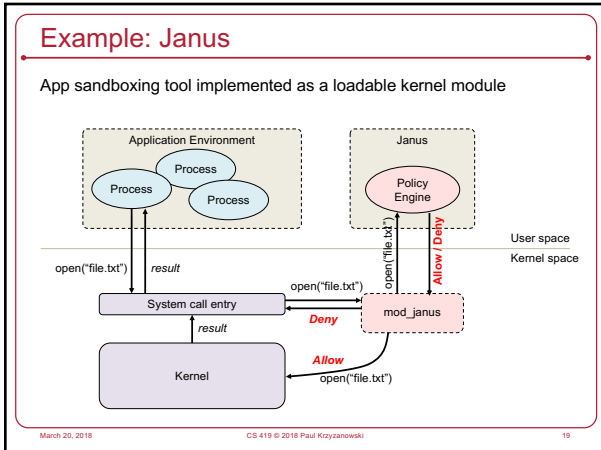
*Jails & containers are a form of sandboxing
 ... but we want to focus on giving users the ability to run apps*

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 17

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

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 18

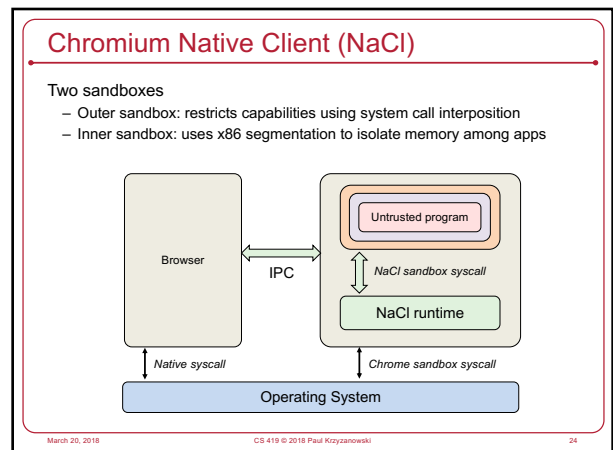


- ### Example: Janus
- **Policy file** defines allowable files and network operations
 - **Dedicated policy per process**
 - Policy engine reads policy file
 - Forks
 - Child process `execs` application
 - All accesses to resources are screened by Janus
 - **System call entry points contain hooks**
 - 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
- March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 20

- ### Implementation 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**
- March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 21

- ### Web plug-ins
- External binaries that add capabilities to a browser
 - Loaded when content for them is embedded in a page
 - Examples: Adobe Flash, Adobe Reader, Java
- March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 22

- ### 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 alignment rules and instruction restrictions
 - GNU-based toolchain, custom versions of gcc/binutils/gdb, libraries
 - 32-bit x86 support
 - NaCl statically verifies the code to check for use of privileged instructions
- March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 23



Java Language

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

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

25

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

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

26

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

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

27

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);`

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

28

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)

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

29

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

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

30

Virtual Machines

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

31

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**

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

32

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

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

33

Machine Virtualization

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

34

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

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

35

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

March 20, 2018

CS 419 © 2018 Paul Krzyzanowski

36

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

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 37

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**

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 38

Hypervisor

Application or Guest OS runs until:

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

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 39

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

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 40

Hardware support for virtualization

Root mode (Intel example)

- Layer of execution more privileged than the kernel

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 41

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

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 42

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

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 43

Two Approaches to Running VMs

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

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 44

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

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 45

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
 - Serves as a proxy to the host OS for access to devices

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 46

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

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 47

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

March 20, 2018 CS 419 © 2018 Paul Krzyzanowski 48

