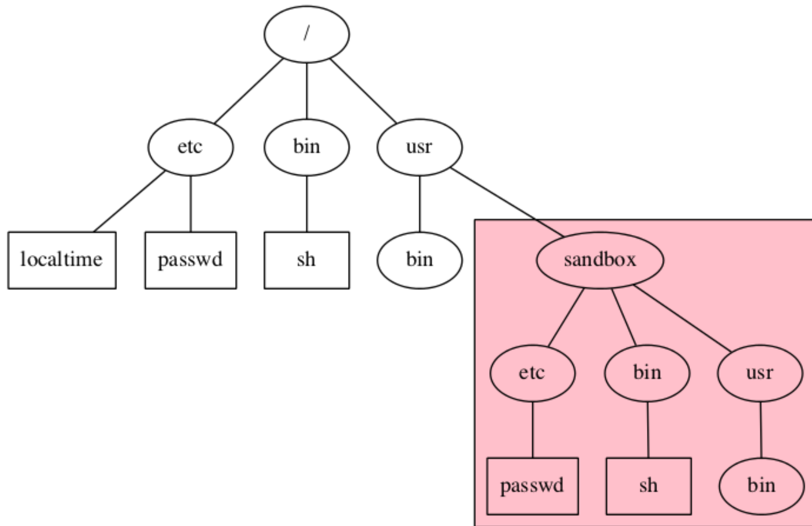
Sandboxing 2

Change Root: `chroot()`

- Oldest Unix isolation mechanism
- Make a process believe that some subtree is the entire file system
- File outside of this subtree simply don't exist
- Sounds good, but...

Chroot



Limitations of Chroot

- Only root can invoke it. (Why?)
- Setting up minimum necessary environment can be painful
- The program to execute generally needs to live within the subtree, where it's exposed
- Still vulnerable to root compromise
- Doesn't protect network identity

Root versus Chroot

- Suppose an ordinary user could use `chroot()`
- Create a link to the sudo command
- Create `/etc` and `/etc/passwd` with a known root password
- Create links to any files you want to read or write
- Besides, root can escape from `chroot()`

Escaping Chroot

- What is the current directory? If it's not under the `chroot()` tree, try `chdir("../../..")`
- Better escape: create *device files*
- On Unix, all (non-network) devices have filenames
- Even physical memory has a filename
- Create a physical memory device, open it, and change the kernel data structures to remove the restriction
- Create a disk device, and mount a file system on it. Then `chroot()` to the real root
- (On Unix systems, disks other than the root file system are "mounted" as a subtree somewhere)

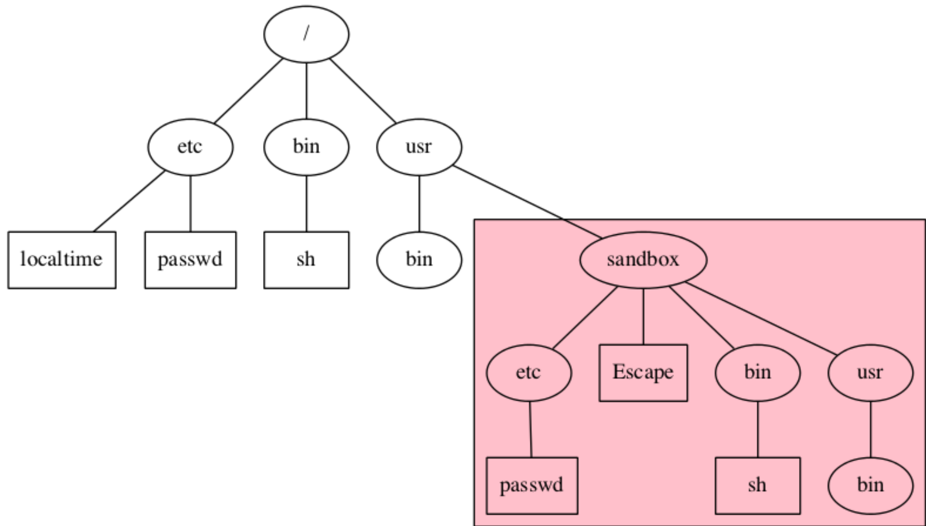
Trying Chroot

```
# mkdir /usr/sandbox /usr/sandbox/bin
# cp /bin/sh /usr/sandbox/bin/sh
# chroot /usr/sandbox /bin/sh
chroot: /bin/sh: Exec format error
# mkdir /usr/sandbox/libexec
# cp /libexec/ld.elf_so /usr/sandbox/libexec
# chroot /usr/sandbox /bin/sh
Shared object "libc.so.12" not found
# mkdir /usr/sandbox/lib
# cp /lib/libc.so.12 /usr/sandbox/lib
# chroot /usr/sandbox /bin/sh
Shared object "libedit.so.2" not found
```

Trying Chroot (Continued)

```
# cp /lib/libedit.so.2 /usr/sandbox/lib
# chroot /usr/sandbox /bin/sh
Shared object "libtermcap.so.0" not found
# cp /lib/libtermcap.so.0 /usr/sandbox/lib
# chroot /usr/sandbox /bin/sh
# ls
ls: not found
# echo sandbox >/Escape
# ^D
# ls -l /usr/sandbox
total 4
drwxr-xr-x  2 root  wheel  512 Nov  1 21:50 bin
-rw-r--r--  1 root  wheel    7 Nov  1 22:31 Escape
drwxr-xr-x  2 root  wheel  512 Nov  1 22:31 lib
drwxr-xr-x  2 root  wheel  512 Nov  1 22:30 libexec
```

After Chroot



Summary of Chroot

- It's a good, but imperfect means of restricting file access
- It's fairly useless against root
- It doesn't provide other sorts of isolation
- Setting up a usable environment is more work than you might think

FreeBSD “Jail”

- Like `chroot()` on steroids
- Assign a separate network identity to a jail partition
- Restrict root’s abilities within a jail
- Intended for nearly-complete system emulation
- Network interactions with main system’s daemons
- But we can do better. . .

Sandboxes

- Very restricted environment, especially for network daemons
- *Assume* that the daemon will do *anything*
- Example: Janus traps each system call and validates it against policy
- Can limit I/O to certain paths

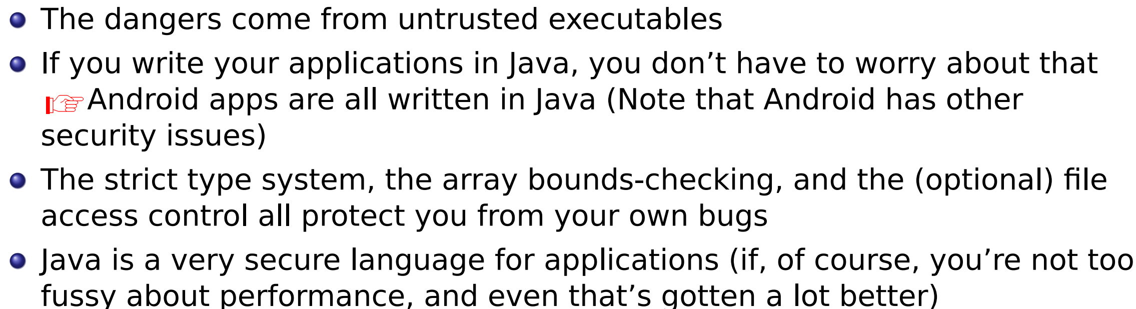
The Java Virtual Machine

- Java executables contain *byte code*, not machine language
- Java interpreter can enforce certain restrictions
- Java *language* prevents certain dangerous constructs and operations (unlike, for example, C)
- In theory, it's safe enough that web browsers can download byte code from arbitrary web sites
- But that's in theory. . .

Is the JVM Secure?

- Heavy dependency on the semantics of the Java language
- The *byte code verifier* ensures that the code corresponds only to valid Java
- The *class loader* ensures that arguments to methods match properly
- Very complex process—not high assurance
- Bugs have been found, but they're fairly subtle
- But—there have been buffer overflows in the C support library
- The support library, in fact, is large, written in C, and quite buggy
- Currently, the JVM is considered to be very insecure

Using the JVM For Servers



- The dangers come from untrusted executables
- If you write your applications in Java, you don't have to worry about that
  ☞Android apps are all written in Java (Note that Android has other security issues)
- The strict type system, the array bounds-checking, and the (optional) file access control all protect you from your own bugs
- Java is a very secure language for applications (if, of course, you're not too fussy about performance, and even that's gotten a lot better)

Virtual Machines

- Give the application an entire “machine”, down to the (virtual) bare silicon
- Run an entire operating system on this
- Run the untrusted application on that OS
- It can be *very* safe—but not perfect

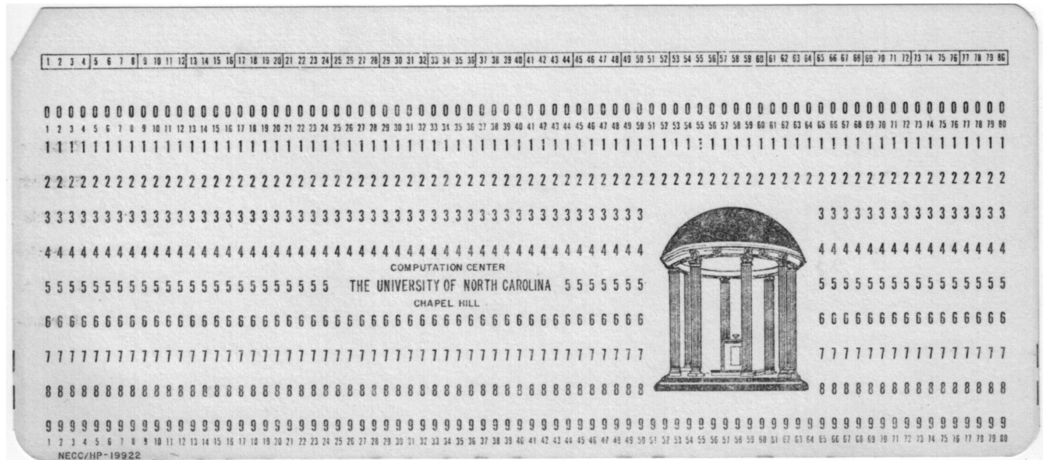
How VMs Work

- Recall the hardware access control mechanisms: privileged operations and memory protection
- Run the guest operating system unprivileged
- Any time the guest OS issues a privileged operation, it traps to the *hypervisor*
- (On some hardware architectures, kernel mode is known as “supervisor state”)
- The hypervisor emulates the operation. For example, an attempt at disk I/O is mapped to I/O to a real file that represents the virtual disk

Virtual Devices

- Virtual disks (or part or all of a real disk)
- Virtual screens, keyboards, and mice
- Virtual Ethernets
- Other virtual devices as needed

There Used to be Virtual Card Readers and Virtual Card Punches



Virtual Machine Security

- *Very* strong isolation
- *Very* high overhead. . .
- Must set up and administer an entire OS

☞ Guest copies of Microsoft Windows or Linux require just as many patches as do native copies

- Performance can be bad, though modern hardware architectures have special instructions to improve VM performance

Using Virtual Machines

- Great for testing OS changes
- Great for student use
- Internet hosting companies
- Can use them for executing suspected viruses and worms—but some viruses detect the presence of the hypervisor and hide

Interacting with a Virtual Machine

- Often don't want perfect isolation.
- Example: cut-and-paste between windows
- Performance can be dramatically enhanced if the guest OS signals the hypervisor
- Example: add a virtual "graphics" driver that calls the hypervisor, via the equivalent of a system call

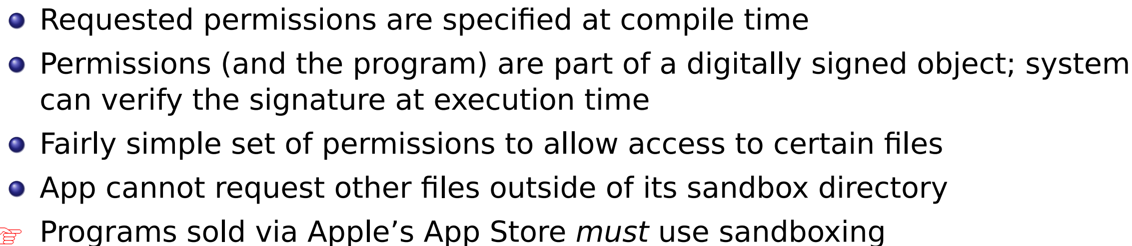
Calling the Hypervisor

- Need an analog to a system call (sometimes known as a *hypercall*)
- Use some instruction that will cause a trap—but not an instruction used by a guest OS
- Example: the first VM system (developed at IBM) relied on an instruction used only to run hardware diagnostics; never used by a real OS
- Can you run a virtual hypervisor? Sometimes...

Limitations of Virtual Machines

- They can be *too* real
- Would you let your enemy put a machine inside your data center?
- VMs can spread viruses, launch DoS attacks, etc.
- VMs require just as much care, administration, and monitoring as do real machines
- In many situations, they represent an *economic* mechanism rather than a security mechanism
- (Save on power, cooling, etc.)
- But—may be less painful when wiping the disk and starting over

The MacOS App Sandbox



- Requested permissions are specified at compile time
- Permissions (and the program) are part of a digitally signed object; system can verify the signature at execution time
- Fairly simple set of permissions to allow access to certain files
- App cannot request other files outside of its sandbox directory
- ☞ Programs sold via Apple’s App Store *must* use sandboxing

App Permissions

App Sandbox

Network: Incoming Connections (Server)
Outgoing Connections (Client)

Hardware: Camera
Microphone
USB
Printing

App Data: Contacts
Location
Calendar

File Access:

Type	Permission & Access
User Selected File	None
Downloads Folder	None
Pictures Folder	None
Music Folder	None
Movies Folder	None

Steps: ✓ Add the "App Sandbox" entitlement to your entitlements file

There are other permissions related to Apple’s online services, e.g., to permit in-app purchases or to be part of their “Game Center”.

Note Carefully

- These restrictions do not map cleanly to file permissions or to a `chroot`-type environment
- `Contacts` is a set of files
- `Location` is a system service
- USB is a broad set of devices
- There are magic filenames: `Downloads`, `Pictures`, etc.
- No other file accesses are permitted

HTML5 Sandboxing

- HTML5 allows IFRAMEs to be sandboxed:
- Plugins, applets, etc., are disabled
- Cookies aren't shared with the sandbox
- No pop-ups, new browser windows, etc.

Windows Sandboxing

- Windows has multiple sandboxes
- Some apps (e.g., Internet Explorer, Adobe Acrobat Reader) are split into trusted/untrusted halves; the untrusted half is sandboxed using *AppContainer*
- All Windows Store apps must be sandboxed
- On some versions of Windows 10, users can create a lightweight virtual machine to run applications

Docker

- Multiplatform application “containers”
- Runs on top of the kernel; contains an application and the libraries it needs
- Does *not* require a full new OS, hence is lighter-weight
- Uses Linux *cgroups* and *namespaces* to protect the host

Cgroups

- High-level mechanism for resource control and accounting
- Can limit memory use, disk bandwidth, CPU consumption, etc.
- Applies to groups of processes
- Also measures resource consumption; usable for billing

Namespaces

- Programs normally refer to things by name: file names, process IDs, etc.
- Linux *namespaces* let a process refer to things by different names, or use an old name for a different resource
- Example: `/etc/passwd` normally refers to a particular system file—but with namespaces, a program opening `/etc/passwd` could actually get a different file—and it couldn't tell that that was what was happening
- Other namespaces: process ID, network addresses, usernames, and more

Docker

- Uses namespaces to remap files and other resources
- Uses cgroups to limit resource usage
- Virtualizes userspace, not the kernel—much higher performance
- But wait—there's more!

More About Docker

- Because Docker virtualizes the file system namespace, it's possible to have different versions of the same files in different containers
- A Docker application's container can contain the specific versions of the libraries and packages it relies on
- A different Docker application on the same machine can use different versions of these libraries, with no conflict
- Eliminates issues of host system dependency
- An application and its dependencies are combined into a *Docker image*
- Docker images can be shipped around!

Docker

- Docker is more for containerization than virtualization
- Security is a useful aspect, but probably not the main motivation
- It does not always simplify system administration—you may have to patch containers
- Example: suppose there's a bug in, say, the JPG rendering library—you have to patch all of your VMs *and* all Docker images that use it
- And remember that one reason for containers is so that different images can run different library versions—including buggy, insecure versions. . .

Which is More Secure, Docker or a VM?

Which is More Secure, Docker or a VM?

- VMs provide stronger isolation
- Docker per se is a *framework*; a lot depends on how the individual images are configured

Which is More Secure, Docker or a VM?

- VMs provide stronger isolation
- Docker per se is a *framework*; a lot depends on how the individual images are configured
- Images that use common libraries are easier to update than on VMs: update one place and all instances benefit
- However: it might be harder to update libraries that are specific to certain Docker images

The Limits of Isolation

- All of the mechanisms we've described are complex (but canned scripts can help)
- Older ones typically require root privileges to set up and often to invoke
- As a consequence, they're useful for complex system designs, but not for general application isolation
- Newer ones are better, but still very complex
- And you can't hide everything

Password-Checking and Sandboxes

- An old operating system (Tenex, for the PDP-10) checked (unhashed) passwords one byte at a time.
- It returned a failure indication as soon as a byte didn't match
- Locate the password overlapping the end of virtual memory; ask the OS to check it
- If the first byte was wrong, it would return "fail".
- If the byte was right, it would try to fetch the next byte, but take a segmentation fault because it was past the edge
- Repeat as needed

Falling Off the Edge of the Earth

					s	e	c	r

				s	e	c	r	e

				s	3	c	r	e

			s	3	c	r	e	t

Walls and Doors

- Fundamentally, our security mechanisms are *walls*
- Separate applications from the kernel and from each other
- But applications need to talk to users, to the kernel, and to other applications—and that requires “doors”
- Our walls are pretty strong (though not perfect)—but doors are hard

Sandboxing a Browser

- Page-rendering is hard (and therefore error-prone): isolate it
- Protect browser tabs from each other, to prevent, e.g., cookie-stealing bugs—put each tab in a separate sandbox
- User interface—keyboard, mouse, display—is safe; doesn't need to be sandboxed

Sandboxing a Browser

- Page-rendering is hard (and therefore error-prone): isolate it
- Protect browser tabs from each other, to prevent, e.g., cookie-stealing bugs—put each tab in a separate sandbox
- User interface—keyboard, mouse, display—is safe; doesn't need to be sandboxed
- But—how do the different parts talk?

Sandboxing a Browser

- Page-rendering is hard (and therefore error-prone): isolate it
- Protect browser tabs from each other, to prevent, e.g., cookie-stealing bugs—put each tab in a separate sandbox
- User interface—keyboard, mouse, display—is safe; doesn't need to be sandboxed
- But—how do the different parts talk?
- That's why we need doors. . .

Doors

- The different pieces of the browser have communication channels to each other
- Fundamentally, this is message-passing
- How do we get the policy right?
- How do we get the implementation right?

Example: A Sandboxed Mailer and Browser

- Break up the mailer in similar fashion
- (Exercise: what should the pieces be?)
- Someone sends you an email containing a URL
- You click on it—so the mailer has to talk to the browser
- Or: there's a `mailto:` URL in a web page, so the browser needs to talk to the mailer
- There needs to be a door between the mailer and the browser

Is There a Door?

- Perhaps that communication takes place in the unprotected parts of these applications
- You can type URLs and compose email messages manually
- Is this channel safe?

Is There a Door?

- Perhaps that communication takes place in the unprotected parts of these applications
- You can type URLs and compose email messages manually
- Is this channel safe?
- No—URL-parsing is *hard*

More Limits to Isolation

- We cannot isolate things completely—we need to talk to applications
- But any communications path is a potential source of attack
- We have to understand what is risky and what isn't
- We have to design our channels carefully—and implement them correctly

Should We Sandbox?

- Yes—our applications are not secure enough
- It’s not a panacea, but it helps
- Note well: the original security model assumed that if the kernel was secure, we didn’t have to worry about applications
- We now know that that isn’t enough, which is why we uses sandboxes on top of our (nominally) secure kernels

Questions?

(Black-throated blue warbler, Central Park, October 27, 2019)