16 – Networking, OS, and virtualization

CS 2043: Unix Tools and Scripting, Spring 2019 [1]

Matthew Milano

March 1, 2019

Cornell University

Table of Contents

1. Firewalls
2. Operating systems, and what they do.
3. Containers, and how they work

Firewalls

Firewalls

- In a perfect world, we wouldn't need a firewall.
- Lives in the network, or in the kernel
- inspects traffic *before* it reaches its destination
- Two primary uses: filter legitimate services, block unwanted ones

Firewalls: the good uses

- Legit: *Filters* certain ports to prevent regions of the internet from accessing them
    - Cornell firewall drops all traffic destined to on-campus servers originating from off-campus IPs
    - `wash` firewall does the same
    - mail relay firewall would only allow known senders to connect
- prevents server from being overloaded by random external griefers
- prevents aggressive server scans from the darkweb
    - which, by the way, exists. ask me later.

Firewalls: the lazy uses.

- Block insecure / old apps
- cover up for weird/bad OS/system design
    - Example: print server on a mac at port 631
    - Example: just a lot of windows
- Block **all** uninvited remote connections
    - if your laptop isn't a server, shouldn't have exposed ports
    - if it does have exposed ports, some application is doing a bad.
- Fundamentally lazy: right answer is to secure the applications, not hide them.
- lots of legacy apps (that we're stuck with) can't be fixed, so also fundamentally necessary

Operating systems, and what they do.

Processors

- The CPU; the chip at the center of your computer
- it actually runs your code
- wired via a *bus* to everything else in your computer
- Has multiple *cores* or *hyperthreads*
    - to allow code to execute simultaneously

Processors have protection modes

- Pieces of code get associated wth a protection mode
  - there's an instruction that literally says "when you run this code, drop these privileges"
- Protection modes let you drop lots of privileges
  - device access
  - physical memory access
  - ability to change protection modes
- Operating system always runs first and keeps all its privileges
- Operating system's job is to run processes for its users

What is a process, really?

- A sequence of processor instructions
- runs from start to finish
- only thing running on CPU core
- what can a process do?
    - access its own memory
    - run arbitrary computation CPU commands
    - fire *interrupts*

What is an interrupt?

- An “unexpected event”
- A request for something else to take over
- Like a signal (in C/unix), or Exception (in java/python/etc)
- Can register *interrupt handlers*, pieces of code that run interrupts
- The operating system registers itself as an *interrupt handler*
- A *syscall* is an interrupt handled by the OS
    - is how you read files, use network, etc.
    - OS registered the handler, so can have all privileges
    - most basic C functions / linux commands just fancy syscall wrappers!

A potential process flow

- start a process
  - drop privileges
  - jump to process code
- do some computation
- read a file
  - fire an interrupt
  - interrupt handler (in OS) gets file
  - file placed in process memory
  - jump back to process code
- use file contents
- do more computation
- exit with result
  - fire an interrupt
  - interrupt handler (in OS) gets result
  - OS clears process memory

Where VMs fit into this

- Using devices (from the OS) also interrupt-based!
- special instruction that sends message along system *bus*
- When host OS launches a VM
    - drops *some* privileges
    - registers *itself* (host OS) for device interrupts
    - launches guest OS
- when guest process wants to use a resource
    - interrupt back to guest OS
    - guest OS interrupts for device
        - Host OS gets interrupt
        - Host OS interrupts for device, or
        - Host OS takes over for a bit

Containers, and how they work

chrooting

change root directory

`chroot <dir> <command>`

- Must execute as root
- hides filesystem below `<dir>`
- `dir` looks like new `/`

- Why do this?
    - all PATHs relative to new root
    - system programs and libraries used from new root
    - can use programs that need incompatible libraries
    - can avoid upgrading system when using a program
- demo

chrooting

- What's still the same in the chroot?
  - kernel
  - process space
  - RAM
  - devices
- Halfway to a container; can have a `chroot` of debian on ChromeOS
- No isolation between `chroot`ed processes and "real" ones

containers

- Special OS feature called LXC containers
- hides processes from each other
- can limit device access within a single container
    - how? checks PID after interrupt, denies request from container process
- 90% of a docker container is chroot + LXC
- Other 10%? Secretly a VM.
    - but only when needed
    - this is why “fancy” Windows 10 is required
- Docker build scripts and bundles are also nice

References

[1] Stephen McDowell, Bruno Abrahao, Hussam Abu-Libdeh, Nicolas Savva, David Slater, and others over the years. "Previous Cornell CS 2043 Course Slides".