Oh.. You got this? Attack the modern web
HELLO DENVER!
...Known for more than recreational ... stuff
WARNING

This talk may contain comments or opinions that at times may differ with those of Cisco Systems. The views expressed here do not necessarily reflect those of Cisco Systems. Audience discretion is advised.
Moses Frost

Work [at] Cisco, Teach/ [at] SANS, Blog [at]
renegade.blog

On the socialz: @mosesrenegade
$~>./WHOAMI

Moses Frost

Work [at] Cisco, Teach/ [at] SANS, Blog [at] renegade.blog
On the socialz: @mosesrenegade
WHY ARE WE HERE?
The ‘stack’ is changing
The ‘stack’ is changing
And they probably redesigned the whole sickbay, too! I know engineers, they LOVE to change things.
My Motivation?
How did we get here?
1. h’oK Define:

What makes up a modern web stack.
What is this?
Cloud Native Charter

- Container Packaged Applications
- Dynamically Managed
- Micro-services oriented

Probably a decent foundation to build from
Cloud Native Charter

- Container Packaged Applications
- Dynamically Managed
- Micro-services oriented

Probably a decent foundation to build from
Cloud Native Charter

- Container Packaged Applications
- Dynamically Managed
- Micro-services oriented

Probably a decent foundation to build from
Let's talk Service Meshes

How did we get here?
Let's talk Service Meshes

How did we get here?
2. **WordPress == ‘Hello World’; of Exploitation**

True Story
Kubernetes Deployment: Wordpress

./kubectl
*I want a new service 'mesh'
Kubernetes Deployment: Wordpress

`./kubectl`

*I want a new service 'mesh'*

GKE: Ok. Time to make the donuts.
Kubernetes Deployment: Wordpress

`.kubectl`

*I want a new service 'mesh'

GKE: Ok. Time to make the donuts.

"Compute engines"
I want a new service ‘mesh’

GKE: Ok. Time to make the donuts.
I want a new service 'mesh'

GKE: Ok. Time to make the donuts.

Let me break this down.

"Compute engines"

"mysql"

docker
docker
docker

docker
docker
docker

"wordpress"

docker
docker
docker

docker
docker
docker

"Service Orchestration"
Kubernetes Deployment: Wordpress

But what the f*!@ is rest of this?!
Kubernetes Deployment: Wordpress

But what the f*!*@ is rest of this?!
**Some Architecture Love**

**Docker:**
- Namespaces and Cgroups in linux
- Windows supports docker as well
- Not the only container runtime
- “Pause Container”
  - Namespaces cannot talk to each other without having a common PARENT
  - Pause is the parent

**Kubernetes:**
- Portable Container Orchestration Environment
- Orchestration for containers
- More than just docker compose

**Services:**
- RBAC is a new feature
  - RBAC restricts what you can see.
- “Service discovery”
  - gRPC
  - DNS
- Proxy and obfuscation
  - Hides servers
  - Move Workloads
Containers
You want to deploy your application BUT:
- How do you update it?
- What if the host crashes?
- How do you scale?
- How do you share between containers?
- ... etc

Mesh
Gives you the ability to orchestrate items
- Can give you capability to move containers
- Restart apps
- Upgrade with circuit breaking
- Share Resources between disparate containers, etc etc etc
- Can be a ‘proxy’ or a sidecar
Example of a Kubernetes engine (GCE)

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Service Type</th>
<th>Endpoints</th>
<th>Pods</th>
<th>Namespace</th>
<th>Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysql</td>
<td>🟢 Ok</td>
<td>Cluster IP</td>
<td></td>
<td>1 / 1</td>
<td>default</td>
<td>persistent-disk-tutorial</td>
</tr>
<tr>
<td>prometheus</td>
<td>🟢 Ok</td>
<td>Cluster IP</td>
<td></td>
<td>1 / 1</td>
<td>stackdriver</td>
<td>persistent-disk-tutorial</td>
</tr>
<tr>
<td>wordpress</td>
<td>🟢 Ok</td>
<td>Load balancer</td>
<td></td>
<td>1 / 1</td>
<td>default</td>
<td>persistent-disk-tutorial</td>
</tr>
</tbody>
</table>

Services are sets of pods with a network endpoint that can be used for discovery and load balancing. Ingresses are collections of rules for routing external HTTP(S) traffic to services.
How did we get here?

Let's talk Service Meshes

Wordpress Demo
A word on demos

- Starting with ‘Easy’ mode.
- Using a common framework and go from here
- The language and framework is mostly **IRRELEVANT**.
Easy Mode What Works: Let's attack Wordpress

Scenario:
- Attacker see's a Wordpress site and gains access to the site.
- Gets a php webshell to work.
- First thing to note is completely missing binaries
  - curl may be installed
  - other binaries not
- curl facilitates bringing down of more binaries

Example:
- Works:
- Does NOT work:
  - http://example/index.php?backdoor=1&cmd=which+nc
TAKEAWAY?

Not all binaries are here.
No ... netcat? I'll bring my own
   ... curl? I'll bring my own
   ... bash? I'll bring my own
Have execve()? Yes, BYO.
./kubectl (yes)
KUBECTL IS PORTABLE

kubectl ...
curl $url; chmod a+x ./kubectl; mv ./kubectl /usr/local/bin/kubectl

The ‘secured’ way:
Error from server (Forbidden): nodes is forbidden: User "system:serviceaccount:default:default" cannot list nodes at the cluster scope: Unknown user "system:serviceaccount:default:default"
TALK JOURNEY

How did we get here?

Let's talk Service Meshes

Wordpress Demo

Microservices
The ‘app’ changes
Before Microservices

App
- Twitter App
  - Create User
  - User gets an email (email function in app)
  - User Logs in
  - User now can tweet
  - User tweets

“App”
- function user()
App 1
- Twitter App
  - Create User

App 2
- Email Worker
  - User gets an email (email function in app)

App 3
- Tweet Worker
New Problem

State/Variables/etc
Microservices

App 1
- Twitter App
  - Create User

App 2
- Email Worker
  - User gets an email (email function in app)

App 3
- Tweet Worker

```
function user()
function tweet()
function mail()
```
Thoughts on exploitation

Apps must share ‘data’ between them
‘In memory databases’ make this convenient.
Can be a conduit for exfil and other issues.
Can you ‘egress’ out of redis store?
Can you ‘ingress’ to the store?
Change values?
The redis story
On the GWPN Mailing List

Question:
Can anyone explain with detail, how code contained within a KEY in a Redis database exposed to the internet, can be executed?
On the GWPN Mailing List

Answer:
Turn outs redis can WRITE TO DISK.
On the GWPN Mailing List

Answer:
Turn outs redis can WRITE TO DISK.
And the directory can be set to
/var/spool/cron
Answer:
Turn outs redis can WRITE TO DISK.
And the directory can be set to
/var/spool/cron
And executes as root.
Serverless's Wat Server?
Let's talk Service Meshes

Wordpress Demo

Microservices

Serverless

How did we get here?
No servers needed
"Serverless"

Build an app without worrying about OS, or Server or Anything. It’s just a function.
AWS Lambda, Google Function...
Google App Engine? (sort of)
Lambda/GCE/Azure Function Example

NodeJS based for the most part.

- NodeJS Infrastructure provided ‘for you’
- Develop a single function
  - Or bring multiple functions
  - Set your own node build.
- Great for simple eventing.

Develop your own bugs also.

Can hide ‘functions’ from the public so that your ‘server’ only see’s it.

BYO–Bugs.
Let's go back to: Mesh

`./kubectl`

*I want a new service 'mesh'*

GKE: Ok. Time to make the donuts.

"Compute engines"

Let me break this down.

"Service Orchestration"

`wordpress`

`mysql`

`docker` `docker` `docker` `docker` `docker` `docker` `docker`
Imagine Chaining

Language 1:
Exploit 1
main url = ?ser=...

Language 2
Exploit 2
payload to attack
**Polyglot exploits**

Language A: PHP (ObjSerialization)

class obj_example {
    public $url = 'www.google.com';

    function __construct() {
    }
}

$obj = base64_decode(unserialize($_GET['ser']))

Language B: NodeJS (RCE/ETC)

exports.helloWhat = (req, res) => {
    let message = req.query.message || req.body.message || 'Hello World!';
    res.status(200).send(message + eval(req.query.q));
};

Chained Exploits:

PHP Obj →

Deserialized Overwrite php $url variable

Pass to some library

New base64 string will get unwound to → requests

Triggers isolated Nodejs SSRF

... More chained language exploits.
Visualize

Deserialized Obj

Overwrite URL

Call curl

Return /etc/passwd

Serverless Call

Read /etc/passwd
Ok we are all doomed and everyone is going to die.
No
Problems found. Problems fixed

Docker Escape
- Dirty Cow exploit
- Speculative Execution

These are definitely out of our direct control.
- Unless you are a kernel/hardware dev.

Kubernetes Exploit / Abuse
- v1.7 and below
  - Can read kubernetes API
  - Can use kubectl
  - Can read other systems
- RBAC has solved many of these issues...
- .. Mostly by default, so research continues to how many are still affected.

Chaining Exploits and other issues:
- Input == dangerous
- Exploitation is REALLY tricky.
- For example:
  - Nodejs
    eval(2+2);
    //String 2+2
    eval('2+2');
    //4
    Makes eval hard/safe between languages.
Post-Modern Web Attacks

An evolving talk series

You can find me at
@mosesrenegade
moses@moses.io