DOCKER SECURITY FOR DEV
Whoami

➔ M.Sc Computer Security

➔ M.Sc Software Development

➔ Worked previously as an embedded software developer

➔ Actually working at ImmunIT

➔ Pentesting
➔ Secure coding training
➔ Security awareness training
➔ Social Engineering
➔ Project Management
➔ R&D developer
DOCKER WORLD
What is Docker?

Containerization
Operating-system-level virtualization
Execution environment virtualization
What is docker?
What is docker?
What is docker?
Why using docker?

➔ Isolate services

➔ Simplify micro-services enhancement and maintenance

➔ Avoid dependency issues

➔ Allow to execute untrusted code safely

➔ Reduce risks involved by a compromise

➔ etc
How it works?
Docker basics

# Build a container
```bash
docker build -t <name>.
```

# List running containers
```bash
docker ps
```

# List stopped containers
```bash
docker ps -a
```

# List docker images
```bash
docker images
```

# Run a container
```bash
docker run <name>
```

# Start a container
```bash
docker start <name>
```

# Stop a container
```bash
docker stop <name>
```

# Delete a container
```bash
docker rm <name>
```

# Delete an image
```bash
docker rmi <name>
```
Dockerfile & docker-compose

Dockerfile

→ Defines a docker image
Dockerfile & docker-compose

➔ Defines a containers stack

#FACT

➔ Overwrite Dockerfile behaviors
Orchestration

- Automates image buildings
- Automates deployment
- Resilient
- Macro management
- Live metrics
Orchestration

Rancher overview
Orchestration

Rancher overview

Hosts

Add Host

**rancheragent**

- **52.87.162.8**
- **1.10.3**
- Ubuntu 15.10 (4.2.0-18-generic)
- 2.5 GHz
- 991 MiB
- 15.6 GiB
- amazonec2

Stack: wordpress

- ![db_1](10.42.61.90)
- ![wordpress_1](10.42.183.10)

**rancheragent02**

- **54.210.194.161**
- **1.10.3**
- Ubuntu 15.10 (4.2.0-18-generic)
- 2.5 GHz
- 991 MiB
- 15.6 GiB
- amazonec2

Stack: wordpress

- ![wordpress_2](10.42.36.90)
Orchestration

Rancher overview

- **CPU**: Total: 4x3.6 GHz, Limit: 4000 mCPU
- **Memory**: Total: 15.6 GB, Limit: 15.6 GB
- **Storage**: 218 GB, Local Limit: 198 TiB
- **Provider**: Custom
- **Kernel**: 4.10.0
- **Docker**: 17.03.1-ce
- **OS**: Ubuntu Zesty Zapus (development branch)
Containers VS Virtual Machine
The millennial war
Containers VS Virtual Machine

Virtual Machine!

Containers!
Containers VS Virtual Machine

Uh oh. Retard alert! Retard alert!
Containers VS Virtual Machine
Security concerns
Kernel namespace

- Containers process are running in their own kernel namespace
  - Provides segregation
  - Decreases risk exposure
- Containers get their own network stack

UPDATE
User namespace

➤ Best way to prevent privilege escalation attack

➤ Configured on the host level

➤ Prevent root usage
Tools

Docker notary

➔ Verify image signature
➔ Ensure integrity
➔ Avoid backdoors
➔ Cross platform
<table>
<thead>
<tr>
<th>NAME</th>
<th>DIGEST</th>
<th>SIZE (BYTES)</th>
<th>ROLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.6</td>
<td>9ace55161307689a12857d62c30ef8daa9a376107ec0ff03e4786cedb3399b</td>
<td>528</td>
<td>targets</td>
</tr>
<tr>
<td>2.7</td>
<td>9f080005c6f55238f0ad2f46b8e65f3f2d5641747d3691e3ea8da67b52804561a</td>
<td>1374</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>2f70d6a6df02d2d73797f11f3dfb7d4ad1c526616ee7c8df5e553a72e4bf79</td>
<td>433</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>4b02d27d451aaehbf60cd59e0e65d5eb82a3b65a6b3b77bc9511cf80b82a6</td>
<td>433</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>37f4d7bb352d858797d0f8c4e6c4e69a9ed44d4e4788a4b445188d7117d14c0</td>
<td>433</td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>c1aa0f93c32358c8b4e87391f02432dc214736c3f170e2e43320c2a2eb96aa0</td>
<td>433</td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>4d3e631cdded98a0b91477b4d1a1fb42a9cadd8139c2e78029e44e199e5843</td>
<td>433</td>
<td></td>
</tr>
<tr>
<td>3.6</td>
<td>de9780d6a3a36dca6a55d260d21be0422fd30dd2d518c1e048b3263e73280cb6</td>
<td>2029</td>
<td></td>
</tr>
<tr>
<td>3.7</td>
<td>5eaf91705874a7c0a5e009c8a18994965a242c3f7e33e178e3e03d700d</td>
<td>2029</td>
<td></td>
</tr>
<tr>
<td>3.8</td>
<td>7d239706348bf040220df6ad72037f98d8593a0918d0ed3ce30c6c93e117e430</td>
<td>2029</td>
<td></td>
</tr>
<tr>
<td>edge</td>
<td>8d9027b2f7dc945db13c2dbf7075f99085ce3c5035cal1d20d9fd0d1d6d5d</td>
<td>2029</td>
<td></td>
</tr>
<tr>
<td>integ-test-base</td>
<td>352dc48dc4135cdd37be0f7a95b36435a3855a3e6e6e3fccc68333377372</td>
<td>528</td>
<td></td>
</tr>
<tr>
<td>latest</td>
<td>7d239706348bf040220df6ad72037f98d8593a0918d0ed3ce30c6c93e117e430</td>
<td>2029</td>
<td></td>
</tr>
</tbody>
</table>

```
$ docker pull alpine
Using default tag: latest
Pull (1 of 1): alpine:latest@sha256:7043076348bf5040220df6ad703798f8593a0918d0ed3ce30c6c93e117e430
8e3ba11ec2a2: Pull complete
Digest: sha256:7ad397e6348bf5040220df6ad703798f8593a0918d0ed3ce30c6c93e117e430
Status: Downloaded newer image for alpine@sha256:7043076348bf5040220df6ad703798f8593a0918d0ed3ce30c6c93e117e430
Tagging alpine@sha256:7043076348bf5040220df6ad703798f8593a0918d0ed3ce30c6c93e117e430 as alpine:latest
```
Tools

Docker bench security

# Docker Bench for Security v1.3.3
# Docker, Inc. (c) 2015-
# Checks for dozens of common best-practices around deploying Docker containers in production.
# Inspired by the CIS Docker Community Edition Benchmark v1.1.0.
#

Initializing Fri Jul 14 09:18:42 UTC 2017

[INFO] 1 - Host Configuration
[WARN] 1.1 - Ensure a separate partition for containers has been created
[NOTE] 1.2 - Ensure the container host has been Hardened
[PASS] 1.3 - Ensure Docker is up to date
  * Using 17.06.0 which is current
[INFO] 1.4 - Check with your operating system vendor for support and security maintenance for Docker
[INFO] 1.5 - Ensure auditing is configured for the Docker daemon
[WARN] 1.6 - Ensure auditing is configured for Docker files and directories - /var/lib/docker
[WARN] 1.7 - Ensure auditing is configured for Docker files and directories - /etc/docker
[WARN] 1.8 - Ensure auditing is configured for Docker files and directories - docker.service
[INFO] 1.9 - Ensure auditing is configured for Docker files and directories - docker.socket
  * File not found
[INFO] 1.10 - Ensure auditing is configured for Docker files and directories - /etc/default/docker
  * File not found
[INFO] 1.11 - Ensure auditing is configured for Docker files and directories - /etc/docker/daemon.json
  * File not found
[WARN] 1.12 - Ensure auditing is configured for Docker files and directories - /usr/bin/docker-containerd
[WARN] 1.13 - Ensure auditing is configured for Docker files and directories - /usr/bin/docker-runc

[INFO] 2 - Docker daemon configuration
[WARN] 2.1 - Ensure network traffic is restricted between containers on the default bridge
[PASS] 2.2 - Ensure the logging level is set to 'info'
[PASS] 2.3 - Ensure Docker is allowed to make changes to iptables
[PASS] 2.4 - Ensure insecure registries are not used

[FAIL] 7 - Ensure root privileges are not used
Tools

Dockscan
Container hardening & Access Control Management
Container hardening & Access Control Management

Seccomp

```json
{
    "defaultAction": "SCMP_ACT_ALLOW",
    "syscalls": [
        {
            "name": "mkdir",
            "action": "SCMP_ACT_ERRNO"
        },
        {
            "name": "chown",
            "action": "SCMP_ACT_ERRNO"
        }
    ]
}
```
Container hardening & Access Control Management

SE Linux

```plaintext
policy_module(localpolicy, 1.0)

gen_require(' 
    type user_t;
    type var_log_t;
)

allow user_t var_log_t:dir { getattr search open read };
```
Container hardening & Access Control Management

App Armor

```c
#include <tunables/globe1>
/usr/sbin/nginx {
    #include <abstractions/apache2-common>
    #include <abstractions/base>
    #include <abstractions/nis>
    capability dac_override,
    capability dac_read_search,
    capability net_bind_service,
    capability setgid,
    capability setuid,
    /data/www/safe/* r,
    deny /data/www/unsafe/* r,
    /etc/group r,
    /etc/nginx/conf.d/ r,
    /etc/nginx/mime.types r,
    /etc/nginx/nginx.conf r,
    /etc/nsswitch.conf r,
    /etc/passwd r,
    /etc/ssl/openssl.cnf r,
    /run/nginx.pid rw,
    /usr/sbin/nginx mr,
    /var/log/nginx/access.log w,
    /var/log/nginx/error.log w,
}```
Flags

Volume vs mount vs tmpfs
Flags

Winner is volume

➔ Easier to back up

➔ Can be managed through the docker CLI

➔ Cross-platform

➔ Safe sharing

➔ Remote volume

➔ Data encryption (LVM, LUKS)
Flags

Winner is volume

➔ Avoid mounting sensitive folder

➔ Use ro flag when needed
Flags

Privileged container

➔ Privileged container run as a proper OS
➔ Can modify interfaces / iptables
➔ Access host devices
Flags

Security opt

--security-opt="label-user:USER" : Set the label user for the container
--security-opt="label-role:ROLE" : Set the label role for the container
--security-opt="label-type:TYPE" : Set the label type for the container
--security-opt="label-level:LEVEL" : Set the label level for the container
--security-opt="label:disable" : Turn off label confinement for the container
--security-opt="apparmor:PROFILE" : Set the apparmor profile to be applied to the container
--security-opt="no-new-privileges:true/false" : Disable/enable container processes from gaining new privileges
--security-opt="seccomp:unconfined" : Turn off seccomp confinement for the container
--security-opt="seccomp-profile.json": White listed syscalls seccomp Json file to be used as a seccomp filter
Use dedicated networks

Isolate containers on separated networks

Create networks for exposed containers

Segregate and segment networks as your own internal network
Ports exposure

➔ Control services exposure

➔ Do not expose unnecessary ports
Ports exposure

```bash
FROM alpine:latest
RUN apk update && apk add httpd
EXPOSE 80

Docker run --rm --it --p 0.0.0.0:1337:80 alpine
Docker run --rm --it --p 127.0.0.1:1337:80 alpine
```

**EXPOSE** keyword is overwritten by **–p** flag at runtime
How to avoid Denial of Service attack
By default, a container has no resource constraints and can use as much of a given resource as the host’s kernel scheduler will allow.
# How to avoid Denial of Service attacks

Memory usage

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-m</code> or <code>--memory</code></td>
<td>The maximum amount of memory the container can use. If you set this option, the minimum allowed value is 4Mi (4 megabyte).</td>
</tr>
<tr>
<td><code>--memory-swap</code></td>
<td>The amount of memory this container is allowed to swap to disk. See <code>--memory-swap</code> details.</td>
</tr>
<tr>
<td><code>--memory-swappiness</code></td>
<td>By default, the host kernel can swap out a percentage of anonymous pages used by a container. You can set <code>--memory-swappiness</code> to a value between 0 and 100 to tune this percentage. See <code>--memory-swappiness</code> details.</td>
</tr>
<tr>
<td><code>--memory-reservation</code></td>
<td>Allows you to specify a soft limit smaller than <code>--memory</code> which is activated when Docker detects contention or low memory on the host machine. If you use <code>--memory-reservation</code>, it must be set lower than <code>--memory</code> for it to take precedence. Because it is a soft limit, it does not guarantee that the container doesn't exceed the limit.</td>
</tr>
<tr>
<td><code>--kernel-memory</code></td>
<td>The maximum amount of kernel memory the container can use. The minimum allowed value is 4Mi. Because kernel memory cannot be swapped out, a container which is starved of kernel memory may block host machine resources, which can have side effects on the host machine and on other containers. See <code>--kernel-memory</code> details.</td>
</tr>
<tr>
<td><code>--oom-kill-disable</code></td>
<td>By default, if an out-of-memory (OOM) error occurs, the kernel kills processes in a container. To change this behavior, use the <code>--oom-kill-disable</code> option. Only disable the OOM killer on containers where you have also set the <code>-m/-memory</code> option. If the <code>-m</code> flag is not set, the host can run out of memory and the kernel may need to kill the host system's processes to free memory.</td>
</tr>
</tbody>
</table>
How to avoid Denial of Service attacks

**CPU usage**

| Option            | Description                                                                                                                                                                                                。
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--cpus=value</code></td>
<td>Specify how much of the available CPU resources a container can use. For instance, if the host machine has two CPUs and you set <code>--cpus=&quot;1.5&quot;</code>, the container is guaranteed at most one and a half of the CPUs. This is the equivalent of setting <code>--cpu-period=&quot;1000000&quot;</code> and <code>--cpu-quot=&quot;1500000&quot;</code>. Available in Docker 1.13 and higher.</td>
</tr>
<tr>
<td><code>--cpu-period=value</code></td>
<td>Specify the CPU CFS scheduler period, which is used alongside <code>--cpu-quot</code>. Defaults to 100 microseconds. Most users do not change this from the default. If you use Docker 1.13 or higher, use <code>--cpus</code> instead.</td>
</tr>
<tr>
<td><code>--cpu-quot=value</code></td>
<td>Impose a CPU CFS quota on the container. The number of microseconds per <code>--cpu-period</code> that the container is limited to before throttled. As such acting as the effective ceiling. If you use Docker 1.13 or higher, use <code>--cpus</code> instead.</td>
</tr>
<tr>
<td><code>--cpuset-cpus</code></td>
<td>Limit the specific CPUs or cores a container can use. A comma-separated list or hyphen-separated range of CPUs a container can use, if you have more than one CPU. The first CPU is numbered 0. A valid value might be <code>0-3</code> (to use the first, second, third, and fourth CPU) or <code>1,3</code> (to use the second and fourth CPU).</td>
</tr>
<tr>
<td><code>--cpu-shares</code></td>
<td>Set this flag to a value greater or less than the default of 1024 to increase or reduce the container's weight, and give it access to a greater or lesser proportion of the host machine's CPU cycles. This is only enforced when CPU cycles are constrained. When plenty of CPU cycles are available, all containers use as much CPU as they need. In that way, this is a soft limit. <code>--cpu-shares</code> does not prevent containers from being scheduled in swarm mode. It prioritizes container CPU resources for the available CPU cycles. It does not guarantee or reserve any specific CPU access.</td>
</tr>
</tbody>
</table>
Conclusion
Best practices

➔ Harden your containers
➔ Isolate your containers
➔ Keep up to date the underlying operating system
➔ Use security tool to monitor your containers
Consider your containers as any physical machine and ensure their compliance towards your company security policies
THANKS FOR YOUR ATTENTION
QUESTIONS ?