redhat

CONTAINERS AND SECURITY

Tips, Tricks, and Mythbusting

Thomas Cameron, RHCA, RHCDS, RHCSS, RHCVA, RHCX
Global Solutions Architect Leader


AGENDA

AGENDA

Containers and Security - All Things Open, October 19th, 2015

INTRODUCTION

Who am I and why should you care?

RED HAT AND CONTAINERS

A brief history

WHAT ARE CONTAINERS?

How do they work?

WHAT ARE CONTAINERS NOT?

Mythbusting

CONTAINER SECURITY

What makes up container security?

KERNEL NAMESPACES

What are they?

AGENDA

Containers and Security - All Things Open, October 19th, 2015

CONTROL GROUPS

What are they?

THE DOCKER DAEMON

How it works and the security it provides

LINUX KERNEL CAPABILITIES

libcap and how Docker deals with it

SELINUX

What it is, what it does, and why it matters

TIPS AND TRICKS

What do do, and what not to do.

CONCLUSIONS

Go forth and contain!

INTRODUCTION

INTRODUCTION

Who am I, and why should you care?

My name is Thomas Cameron, and I'm the global solutions architect leader at Red Hat.

- In IT since 1993
- I was a police officer before that, educational background is in law enforcement & security
- At Red Hat since 2005
- Red Hat Certified Architect, Red Hat Certified Security Specialist, and other certs
- In the past, I was an MCSE, a MCT, and a CNE
- Spent a lot of time focusing on security in organizations like banks, manufacturing companies, e-commerce, etc.
- I do NOT know everything. But I have some pretty impressive scars.


Don't mess with coders.
JUDCon2011:Boston

RED HAT AND CONTAINERS

RED HAT AND CONTAINERS

A brief history

Red Hat has been working on container technologies since before 2010

- Makara acquisition 2010 – PaaS
- Rebranded as OpenShift
- “Cartridges” using SELinux, cgroups, kernel namespaces
- Docker came to prominence in 2013(-ish)
- Docker gained community adoption and we started participating in 2013. Meritocracy rules!
- Red Hat is a top contributor to Docker (#2 behind Docker at last check)

RED HAT AND CONTAINERS

A brief history

Industry adoption of Docker is incredible

- Docker has been through multiple successful venture capital rounds
- Apcera, Cisco, EMC, Fujitsu Limited, Goldman Sachs, HP, Huawei, IBM, Intel, Joyent, Mesosphere, Pivotal, Rancher Labs, Red Hat and VMware are all on board with container standardization with Docker.

RED HAT AND CONTAINERS

A brief history

Industry adoption of Docker is incredible

- Even Microsoft announced that they will support Docker containers!


Hell

WHAT ARE CONTAINERS?

WHAT ARE CONTAINERS?

How do they work?

Containerization, specifically Docker, is a technology which allows applications (web, database, app server, etc.) to be run abstracted from, and in some isolation from, the underlying operating system. The docker service can launch containers regardless of the underlying Linux distro.

Containers can enable incredible application density, since you don't have the overhead of a full OS image for each app. Linux control groups also enable maximum utilization of the system.

The same container can run on different versions of Linux

- Ubuntu containers on Fedora
- CentOS containers on RHEL

HUMAN SACRIFICE!
DOGS AND CATS, LIVING TOGETHER!
MASS HYSTERIA!

WHAT ARE CONTAINERS?

How do they work?

OK, maybe not...

Containers make it really easy for application developers to build and deploy apps.

WHAT ARE CONTAINERS NOT?

WHAT ARE CONTAINERS NOT?

Mythbusting

Containers are not a panacea. They are not “the cure to all that ails you!”

redhat.

WHAT ARE CONTAINERS NOT?

Mythbusting

Containers are not a fit for every application.

WHAT ARE CONTAINERS NOT?

Mythbusting

They are not virtualization. You can run containers on an OS on bare metal.

CONTAINER SECURITY

CONTAINER SECURITY

What makes up container security?

Containers use several mechanisms for security:

- Linux kernel namespaces
- Linux Control Groups (cgroups)
- The Docker daemon
- Linux capabilities (libcap)
- Linux security mechanisms like AppArmor or SELinux

KERNEL NAMESPACES

LINUX KERNEL NAMESPACES

What are they?

Namespaces are just a way to make a global resource appear to be unique and isolated. The namespaces that the Linux kernel can manage are:

- Mount namespaces
- PID namespaces
- UTS namespaces
- IPC namespaces
- Network namespaces
- User namespaces

MOUNT NAMESPACES

LINUX KERNEL NAMESPACES

What are they?

Mount namespaces allow a container to “think” that a directory which is actually mounted from the host OS is exclusively the container's.

When you start a container with the -v [host-path]:[container-path]:[rw|ro] argument, you can mount a directory from the host in the container. The container “sees” the directory in its own mount namespace, not knowing that it is actually on the host. So multiple containers could, for instance use the host's /var/www/html directory without having to copy content to all the containers.

root@t540p:~

File Edit View Search Terminal Help

```
[root@t540p ~]# cat /var/www/html/index.html
this is my silly web page
[root@t540p ~]# docker run -it -v /var/www/html:/var/www/html fedora bash
bash-4.3# cat /var/www/html/index.html
this is my silly web page
bash-4.3# 
```

LINUX KERNEL NAMESPACES

Security implications - discussion

How do mount namespaces affect security?

PID NAMESPACES

LINUX KERNEL NAMESPACES

What are they?

PID namespaces let the container think it's a new instance of the OS. When you start a container on a host, it will get a new process ID. PID namespaces enable the container to “see” the PIDs inside the container as unique, as if the container were its own instance of an OS.

In the following example, I launch a Fedora container running bash, and run “ps ax”

The container only “sees” its own PID namespace, so the bash process exists within the container as PID 1. On the host, however, the docker process is PID 18557:

tcameron@t540p:~

File Edit View Search Terminal Help

```
[tcameron@t540p ~]$ docker run -it fedora bash
bash-4.3# ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:00 bash
    6 ?        R+     0:00 ps ax
bash-4.3# 
```

tcameron@t540p:~

File Edit View Search Terminal Help

```
10822 tty2     SLl+   3:08 /opt/google/chrome/chrome
10830 tty2     S+     0:00  \_ cat
10831 tty2     S+     0:00  \_ cat
10834 tty2     S+     0:00  \_ /opt/google/chrome/chrome --type=zygote
10836 tty2     S+     0:00  |   \_ /opt/google/chrome/nacl_helper
10839 tty2     S+     0:00  |   \_ /opt/google/chrome/chrome --type=zygote
10923 tty2     Sl+    0:02  |       \_ /opt/google/chrome/chrome --type=renderer
11047 tty2     Sl+    4:04  |       \_ /opt/google/chrome/chrome --type=ppapi --
11118 tty2     Sl+   17:28  |       \_ /opt/google/chrome/chrome --type=renderer
10919 tty2     Sl+    2:23  \_ /opt/google/chrome/chrome --type=gpu-process --ch
10971 tty2     S+     0:00      \_ /opt/google/chrome/chrome --type=gpu-broker
11658 tty2     Sl+    0:00 /usr/lib64/libreoffice/program/oosplash --impress fil
11673 tty2     Sl+    3:58  \_ /usr/lib64/libreoffice/program/soffice.bin --impr
14994 tty2     Sl+    0:15 evince /home/tcameron/Desktop/container_security/Dock
14999 tty2     Sl+    0:00 /usr/libexec/evinced
16451 ?        Ssl    0:01 /usr/bin/docker -d --selinux-enabled
18596 pts/1    Ss+    0:00  \_ bash
18514 tty2     Sl+    0:00 /usr/libexec/gnome-terminal-server
18517 tty2     S+     0:00  \_ gnome-pty-helper
18518 pts/0    Ss     0:00  \_ bash
18557 pts/0    Sl+    0:00  |   \_ docker run -it fedora bash
18613 pts/2    Ss     0:00  \_ bash
18658 pts/2    R+     0:00      \_ ps axf
[tcameron@t540p ~]$ 
```

LINUX KERNEL NAMESPACES

Security implications - discussion

How does this affect security within a container? How about from outside?

USER NAMESPACES

LINUX KERNEL NAMESPACES

What are they?

When you start a container, assuming you've added your user to the docker group, you start it as your user account. In the following example, I start the container as tcameron. Once the container is started, my user inside the container is root. This is an example of user namespaces.

tcameron@t540p:~

File Edit View Search Terminal Help

```
[tcameron@t540p ~]$ id
uid=1000(tcameron) gid=1000(tcameron) groups=1000(tcameron),1002(docker) context
=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[tcameron@t540p ~]$ docker run -it fedora bash
bash-4.3# id
uid=0(root) gid=0(root) groups=0(root)
bash-4.3# 
```

LINUX KERNEL NAMESPACES

Security implications - discussion

What are the security implications of user namespacing?

redhat.

NETWORK NAMESPACES

LINUX KERNEL NAMESPACES

What are they?

Network namespaces allow a container to have its own IP address, independent of that of the host. These addresses are not available from outside of the host, this is private networking similar to that of virtualization. The Docker service sets up an iptables masquerading rule so that the container can get to the rest of the Internet.

In the following query, I find that my Fedora instance has the address 172.17.0.7, even though the host doesn't have an address associated with the ethernet interface:

tcameron@t540p:~

File Edit View Search Terminal Help

```
[tcameron@t540p ~]$ docker inspect --format '{{ .NetworkSettings.IPAddress }}' f
d12ac784fb7
172.17.0.7
[tcameron@t540p ~]$ ip a show enp0s25
2: enp0s25: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DO
WN group default qlen 1000
    link/ether 54:ee:75:10:7b:fc brd ff:ff:ff:ff:ff:ff
[tcameron@t540p ~]$ 
```

LINUX KERNEL NAMESPACES

Security implications - discussion

What are the security implications of network namespacing?

IPC NAMESPACES

LINUX KERNEL NAMESPACES

What are they?

IPC namespaces do the same thing with interprocess communications. My container has no IPCs mapped, but my host has many:

tcameron@t540p:~

File Edit View Search Terminal Help

```
bash-4.3# ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

bash-4.3# 
```

tcameron@t540p:~

File Edit View Search Terminal Help

```
[tcameron@t540p ~]$ ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x6c6c6536 0          root       600        4096       0
0x3714112c 8814593    root       600        1000       7
0xf325112f 8847362    root       600        8          7
0x00000000 163843     gdm        600        524288     2          dest
0x00000000 262148     tcameron   600        524288     2          dest
0x00000000 294917     tcameron   600        4194304    2          dest
0x00000000 524294     tcameron   600        524288     2          dest
0x00000000 6979591    tcameron   600        393216     2          dest
0x00000000 7208968    tcameron   600        4194304    2          dest
0x00000000 7110665    tcameron   600        1048576    2          dest
0x00000000 7307274    tcameron   600        524288     2          dest
0x00000000 6389771    tcameron   600        8388608    2          dest
0x00000000 7340044    tcameron   600        393216     2          dest
0x00000000 7667725    tcameron   600        393216     2          dest
0x00000000 9109518    tcameron   600        524288     2          dest
0x00000000 7864335    tcameron   600        393216     2          dest
0x00000000 7897104    tcameron   600        67108864   2          dest
```

LINUX KERNEL NAMESPACES

Security implications - discussion

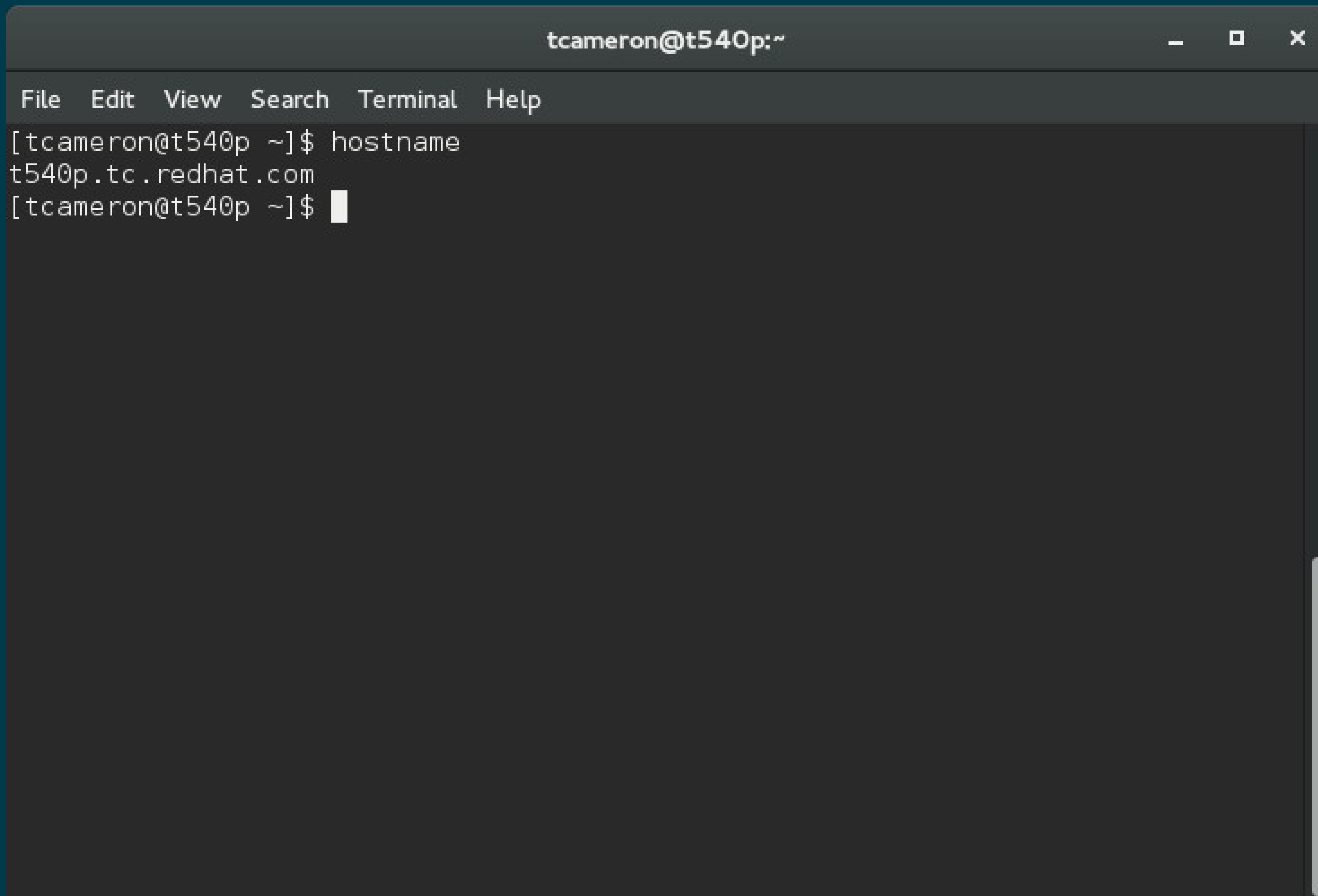
What are the security implications of IPC namespacing? Compare and contrast with chroot or other non-container isolation.

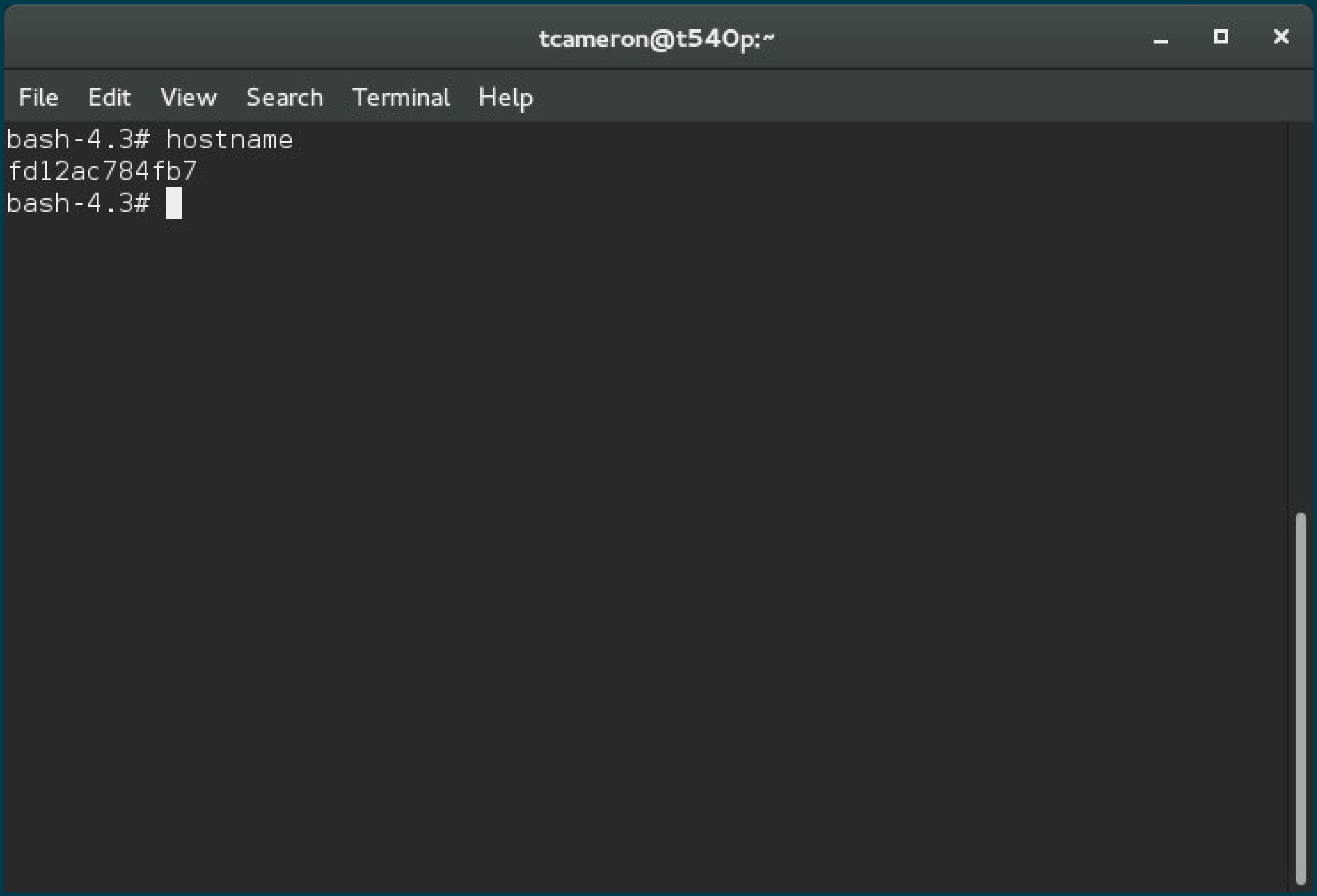
UTS NAMESPACES

LINUX KERNEL NAMESPACES

What are they?

UTS (UNIX Timesharing System) namespaces let the container "think" it's a separate OS, with its own hostname and domain name:


tcameron@t540p:~
File Edit View Search Terminal Help
[tcameron@t540p ~]$ hostname
t540p.tc.redhat.com
[tcameron@t540p ~]$


tcameron@t540p:~
File Edit View Search Terminal Help
bash-4.3# hostname
fd12ac784fb7
bash-4.3#

LINUX KERNEL NAMESPACES

Security implications - discussion

What are the security implications of UTS namespacing?

CONTROL GROUPS

CONTROL GROUPS

What are they?

From the documentation at https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt:

“Control Groups provide a mechanism for aggregating/partitioning sets of tasks, and all their future children, into hierarchical groups with specialized behavior.”

This allows us to put various system resources into a group, and apply limits to it, like how much disk IO, CPU use, memory use, network use, namespaces, and so on. In the case of containers, the resources are those assigned to that container.

CONTROL GROUPS

What are they?

This ensures that, even if a container is compromised (or just spins out of control), there are limits in place which minimizes the risk of that misbehaved container impacting the host or other containers.

CONTROL GROUPS

What are they?

Note that when I run the command systemctl status docker.service, I get the control group and slice information:

```
tcameron@t540p:~
File  Edit  View  Search  Terminal  Help
[tcameron@t540p ~]$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor prese
t: disabled)
   Active: active (running) since Mon 2015-10-19 03:23:44 EDT; 1h 58min ago
     Docs: http://docs.docker.com
 Main PID: 16451 (docker)
   Memory: 2.2M
   CGroup: /system.slice/docker.service
           └─16451 /usr/bin/docker -d --selinux-enabled

Oct 19 04:43:31 t540p.tc.redhat.com docker[16451]: time="2015-10-19T04:43:31....
Oct 19 04:44:15 t540p.tc.redhat.com docker[16451]: time="2015-10-19T04:44:15....
Oct 19 04:44:16 t540p.tc.redhat.com docker[16451]: time="2015-10-19T04:44:16....
Oct 19 04:44:16 t540p.tc.redhat.com docker[16451]: time="2015-10-19T04:44:16....
Oct 19 04:44:16 t540p.tc.redhat.com docker[16451]: time="2015-10-19T04:44:16....
Oct 19 04:46:44 t540p.tc.redhat.com docker[16451]: time="2015-10-19T04:46:44....
Oct 19 04:46:52 t540p.tc.redhat.com docker[16451]: time="2015-10-19T04:46:52....
Oct 19 04:46:57 t540p.tc.redhat.com docker[16451]: time="2015-10-19T04:46:57....
Oct 19 04:47:28 t540p.tc.redhat.com docker[16451]: time="2015-10-19T04:47:28....
Oct 19 04:50:28 t540p.tc.redhat.com docker[16451]: time="2015-10-19T04:50:28....
Hint: Some lines were ellipsized, use -l to show in full.
[tcameron@t540p ~]$
```

CONTROL GROUPS

What are they?

You can navigate the /sys/fs/cgroup/ pseudo-directory to see what resources are allocated to your containers.

There are over 8500 entries in this directory on my system, so it is not practical to talk about the details of individual cgroups, but you can get information about memory, cpu, block I/O, network I/O, and so on here.

tcameron@t540p:/sys/fs/cgroup

File Edit View Search Terminal Help

```
[tcameron@t540p ~]$ cd /sys/fs/cgroup/
[tcameron@t540p cgroup]$ find . | wc -l
8584
[tcameron@t540p cgroup]$
```

THE DOCKER DAEMON

THE DOCKER DAEMON

How it works and the security it provides

The docker daemon (/usr/bin/docker) is responsible for managing the control groups, orchestrating the namespaces, and so on so that docker images can be run and secured.

Because of the need to manage kernel functions, Docker runs with root privileges. Be aware of this!

THE DOCKER DAEMON

How it works and the security it provides

There are some considerations for running Docker:

- Only allow trusted users to run docker. The Docker documentation recommends that you add users to the docker group so they can run docker commands. With this flexibility comes risk. Make sure you only delegate this ability to trusted users. Remember that they can mount the host filesystem in their container with root privileges!
- If you are using the REST API to manage your host(s), make sure you do not have vulnerabilities exposed. Ensure you have strong authentication.
- Use SSL if you are going to expose the REST API over http. Don't expose it except to secured networks or VPN.

LINUX KERNEL CAPABILITIES
(libcap)

LINUX KERNEL CAPABILITIES

libcap and how Docker deals with it

The root user historically had the ability to do anything, once authenticated. Linux capabilities is a set of fine grained controls which allow services or even users with root equivalence to be limited in their scope.

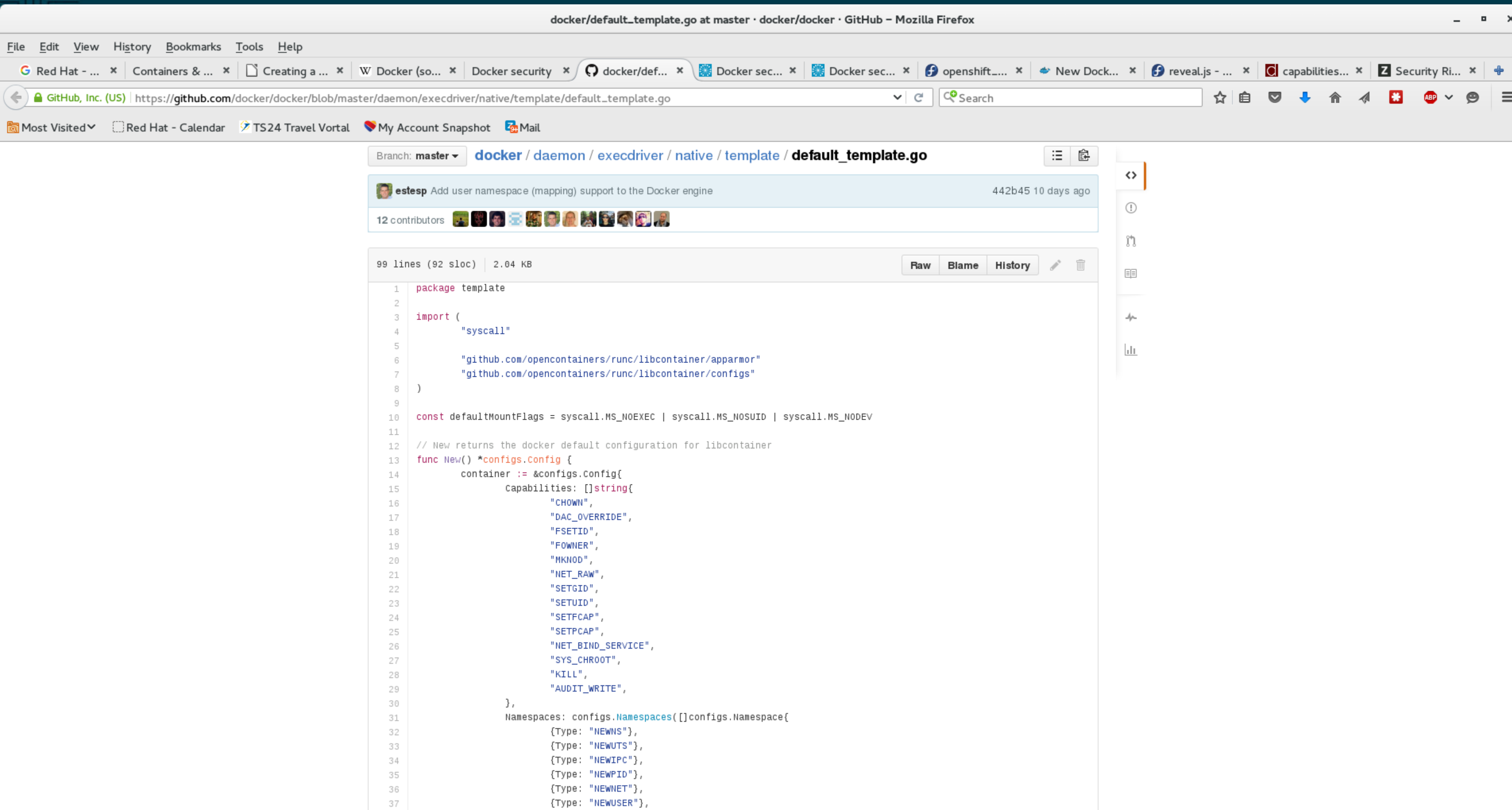
It also allows non-root users to be granted extra privileges. A regular user, for instance, could be granted the net_bind_service capability and they could bind a service to a privileged port (below 1024).

LINUX KERNEL CAPABILITIES

libcap and how Docker deals with it

In containers, many of the capabilities to manage network and other services are not actually needed. SSH services, cron, services, filesystem mounts and unmounts are not needed, network management is not needed, etc.

By default, Docker disallows many root capabilities, including the ability to modify logs, change networking, modify kernel memory, and the catch-all CAP_SYS_ADMIN.




```go
package template

import (
	"syscall"

	"github.com/opencontainers/runc/libcontainer/apparmor"
	"github.com/opencontainers/runc/libcontainer/configs"
)

const defaultMountFlags = syscall.MS_NOEXEC | syscall.MS_NOSUID | syscall.MS_NODEV

// New returns the docker default configuration for libcontainer
func New() *configs.Config {
	container := &configs.Config{
		Capabilities: []string{
			"CHOWN",
			"DAC_OVERRIDE",
			"FSETID",
			"FOWNER",
			"MKNOD",
			"NET_RAW",
			"SETGID",
			"SETUID",
			"SETFCAP",
			"SETPCAP",
			"NET_BIND_SERVICE",
			"SYS_CHROOT",
			"KILL",
			"AUDIT_WRITE",
		},
		Namespaces: configs.Namespaces([]configs.Namespace{
			{Type: "NEWNS"},
			{Type: "NEWUTS"},
			{Type: "NEWIPC"},
			{Type: "NEWPID"},
			{Type: "NEWNET"},
			{Type: "NEWUSER"},
```

SELINUX

SELINUX

What it is, what it does, and why it matters

Security Enhanced Linux (SELinux) is a mandatory access control system. Processes, files, memory, network interfaces, and so on are labeled, and there is a policy which is administratively set and fixed.

That policy will determine how processes can interact with files, each other, network ports, and the like.

redhat.

SELINUX

What it is, what it does, and why it matters

SELinux is primarily concerned with labeling and type enforcement. For a mythical service "foo," the executable file on disk might have the label foo_exec_t. The startup scripts for foo might have the label foo_config_t. The log files for foo might have the label foo_log_t. The data for foo might have the label foo_data_t. When foo is running, the process in memory might have the label foo_t.

Type enforcement is the rule set that says that when a process running in the foo_t context tries to access a file on the filesystem with the label foo_config_t or foo_data_t, that access is allowed. When the process with the label foo_t tries to write to a log file with the foo_log_t, that would be allowed, as well. Any other access, unless explicitly allowed by policy, is denied.

SELINUX

What it is, what it does, and why it matters

If the foo process, running in the foo_t context tries to access, for instance, the directory /home/tcameron, with the label user_home_dir_t, even if the permissions are wide open, the policy will stop that access.

SELinux labels are stored as extended attributes on the filesystem, or in memory.

SELINUX

What it is, what it does, and why it matters

SELinux labels are stored in the format:

- selinux_user:selinux_role:selinux_type:mls:mcs

So for the mythical “foo” service, the full syntax for the label of the running process might be:

- user_u:object_r:foo_t:s0:c0

SELINUX

What it is, what it does, and why it matters

The default policy for SELinux is “targeted.” In the targeted policy, we don't use the SELinux user or role, so we'll ignore them for today. We will also ignore the MLS (multilevel security) label, since that is only used in the MLS policy (think top secret vs. secret in the military).

We really only care about the type (remember, type enforcement) and the MCS label. Think of MCS labels as extra identifiers. In SELinux for containers, we can be very granular about which processes can access which other processes.

These are different labels:

- user_u:object_r:foo_t:s0:c0
- user_u:object_r:foo_t:s0:c1

SELINUX

What it is, what it does, and why it matters

Type enforcement says that a process with the first label is different from the process with the second. So policy would prevent them from interacting. Also, there is no policy allowing a process running with those labels to access the filesystem unless it is labeled with foo_config_t or foo_content_t or another defined label.

Neither of those processes would be able to access /etc/shadow, which has the label shadow_t.

SELINUX

What it is, what it does, and why it matters

On a standalone system running Docker, all of the containers run in the same context by default. In Red Hat's PaaS offering, OpenShift, this is not the case. Each Openshift container runs in its own context, with labels like:

- staff_u:system_r:openshift_t:s0:c0,c1
- staff_u:system_r:openshift_t:s0:c2,c3
- staff_u:system_r:openshift_t:s0:c4,c5

So, even if someone were to gain access to the docker container process on the host, SELinux would prevent them from being able to access other containers, or the host.

redhat.

SELINUX

What it is, what it does, and why it matters

In the following example, I emulate an exploit where someone takes over a container. I use runcon (run in the context) to set my context to that of an Openshift container.

I attempt to access /etc/shadow (shadow_t label). I try to write to the filesystem. I try to read the contents of a user's home directory.

root@t540p:~

File Edit View Search Terminal Help

```
[root@t540p ~]# id
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r:unconfi
ned_t:s0-s0:c0.c1023
[root@t540p ~]# id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[root@t540p ~]# runcon -u unconfined_u -r system_r -t openshift_t -l s0:c0,c1 /b
in/bash
bash: /root/.bashrc: Permission denied
bash-4.3# cat /etc/shadow
cat: /etc/shadow: Permission denied
bash-4.3# touch /testfile
touch: cannot touch '/testfile': Permission denied
bash-4.3# ls /home/tcameron
ls: cannot access /home/tcameron: Permission denied
bash-4.3# setenforce 0
setenforce:  setenforce() failed
bash-4.3# 
```

TIPS AND TRICKS

TIPS AND TRICKS

What to do, and what not to do

Containers are, at the end of the day, just processes running on the host. Use common sense.

TIPS AND TRICKS

What to do, and what not to do

Do:

- Have a process in place to update your containers. Follow it.
- Run services in the containers with the lowest privilege possible. Drop root privileges as soon as you can.
- Mount filesystems from the host read-only wherever possible.
- Treat root inside the container just like you would on the host.
- Watch your logs.

TIPS AND TRICKS

What to do, and what not to do

Don't:

- Download any old container you find on the 'net.
- Run SSH inside the container.
- Run with root privileges.
- Disable SELinux.
- Roll your own containers once, and never maintain them.
- Run production containers on unsupported platforms.

CONCLUSION

CONCLUSION

Go forth and contain!

Containers are incredibly cool. They make application deployment really, really easy. They leverage some incredible capabilities within the Linux kernel. By design, they are relatively secure, but there are some gotchas.

As with every other piece of software out there, docker tech requires some feeding and maintenance. Well maintained, containers can make your business more agile, less complex, and safe.

redhat

THANK YOU

plus.google.com/+RedHat

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHatNews