

Linux security and system hardening

lab 2

Jeffrey Benteux - <https://www.benteux.fr>

September 29, 2026

Outline

objective

setup

service creation

cgroups

namespaces

more namespaces

conclusion

objective

setup

service creation

cgroups

namespaces

more namespaces

conclusion

objective

- build our own container from scratch

objective

- build our own container from scratch
- using cgroups and namespaces

objective

- build our own container from scratch
- using cgroups and namespaces
- run a secure service inside the container

Outline

objective

setup

service creation

cgroups

namespaces

more namespaces

conclusion

- virtual machine with Qemu
- Debian 13 image¹
- you are free to reuse kernel from lab1 or the default image kernel

¹<https://cloud.debian.org/images/cloud/trixie/20260914-2601/debian-13-nocloud-amd64-20260914-2601.qcow2>

boot

```
qemu-system-x86_64 -drive file=/path/to/lab2-debian-13-  
nocloud-amd64-20260914-2601.qcow2 -m 1024M -enable-kvm -  
cpu host -kernel ./arch/x86_64/boot/bzImage -append '  
root=/dev/sda1 lsm=yama,selinux,bpf l1tf=full,force  
page_poison=on pti=on slab_nomerge=yes slub_debug=FZP  
spec_store_bypass_disable=seccomp spectre_v2=on mds=full  
,nosmt mce=0 page_alloc.shuffle=1 rng_core.  
default_quality=500' -net user,hostfwd=tcp::10022-:22 -  
net nic
```

check dependencies

```
# Guest
$ ls -l /proc/self/ns
# if there is no 'user' file , user namespaces are not
  supported by the
# current kernel

# Host
# Edit configuration
make menuconfig
# Type '/' and look for USER_NS, enable , save , quit ,
  recompile
```

Outline

objective

setup

service creation

cgroups

namespaces

more namespaces

conclusion

- our process is a web server
- we will create a custom service
- with a dedicated user

create a service user

```
# Guest  
# useradd -M -r -s /usr/bin/nologin svc_websrv  
# grep svc_websvc /etc/passwd /etc/shadow
```

custom service

create a service user

```
# Guest
# useradd -M -r -s /usr/bin/nologin svc_websrv
# grep svc_websvc /etc/passwd /etc/shadow
```

run a process as service user

```
# runuser -u svc_websrv -- python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...

# ps faux | grep python
root 216  0.0  2.1 36808 21472 ? /usr/bin/python3 /usr/share/unattended-
    upgrades/unattended-upgrade-shutdown --wait-for-signal
root 417  0.0  0.3  7472  3860 pts/1 |  \- /sbin/runuser -u svc_websrv --
    python3 -m http.server 8000
svc.web+ 418  0.0  1.9 29508 18948 pts/1 |  \- python3 -m http.server 8000
```

Outline

objective

setup

service creation

cgroups

namespaces

more namespaces

conclusion

- let's isolate resource for our web server
- we want limits for cpu, memory and PIDs (for client connections)

cgroups support in kernel

```
# Change kernel config for cgroup support
$ make menuconfig

# Make changes so that your config contains:
CONFIG_CGROUPS=y
CONFIG_MEMCG=y
CONFIG_CFS_BANDWIDTH=y

# Check on guest
# cat /sys/fs/cgroup/cgroup.controllers
cpuset cpu io memory hugetlb pids rdma misc
```

create custom cgroups

```
# Allow sub-cgroups to handle needed parameters
echo '+cpu +memory +pids' > /sys/fs/cgroup/cgroup.
    subtree_control

# Creates the cgroup
# mkdir /sys/fs/cgroup/web

# Giving it half a CPU
# echo "50000 100000" | tee /sys/fs/cgroup/web/cpu.max

# Giving it half the VM memory
# echo 512M | tee /sys/fs/cgroup/web/memory.max

# Will handle 99 simultaneous connections
# echo 100 | tee /sys/fs/cgroup/web/pids.max
```

add our process to the cgroup

```
# PID=$(pgrep 'python3') && echo "$PID" > /sys/fs/cgroup/web  
/cgroup.procs  
# cat /sys/fs/cgroup/web/cgroup.procs  
<PID>  
# cat /proc/<PID>/cgroup  
0::/web
```

monitor

```
# systemd-cgtop
```

objective

setup

service creation

cgroups

namespaces

more namespaces

conclusion

- now that our process has limited resources, let's hide systems components from it with namespaces

- now that our process has limited resources, let's hide systems components from it with namespaces
- we will focus on the following namespaces: `mount`, `pid`, `user`

- now that our process has limited resources, let's hide systems components from it with namespaces
- we will focus on the following namespaces: `mount`, `pid`, `user`
- we are forced to kill our `runuser` to use both `cgroups` and namespaces

- now that our process has limited resources, let's hide systems components from it with namespaces
- we will focus on the following namespaces: `mount`, `pid`, `user`
- we are forced to kill our `runuser` to use both `cgroups` and namespaces
- for the sake of this lab, we will start our process in a bash shell

start our process in namespaces

```
# sudo -u svc_websrv unshare --fork --mount --mount-proc --pid --user --map-root
--user -- /bin/bash

# ps fau
root \_ sudo -u svc_websrv unshare --fork --mount --mount-proc --pid --user --
map-root-user -- /bin/bash
svc-web+ \_ unshare --fork --mount --mount-proc --pid --user --map-root-user --
/bin/bash
svc-web+ \_ /bin/bash
```

put our process in the previous cgroup

```
# Add it to our cgroup
# PID=$(pgrep unshare); echo "$PID" > /sys/fs/cgroup/web/
    cgroup.procs
# Because we already created a process child (bash), we need
    to add it too
echo "<bash_PID>" > /sys/fs/cgroup/web/cgroup.procs
```

test in container

```
# Inside the container , our bash PID is 1
```

```
# echo $$
```

```
1
```

```
# Our UID and GID is root
```

```
# id
```

```
uid=0(root) gid=0(root) groups=0(root) context=kernel
```

```
# We cannot see host processes
```

```
# ps faux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.4	4760	3992	pts/3	S	17:02	0:00	/bin/bash
root	3	0.0	0.4	6796	4032	pts/3	R+	17:05	0:00	ps faux

start the web server

```
# Start our web server in container
# python3 -m http.server 8000 &

# Check that it has the expected rights and cgroup (host)
svc_web+ 339 0.0 1.9 29508 [...] python3 -m http.server 8000

# cat /sys/fs/cgroup/web/cgroup.procs
328 (unshare PID)
329 (bash PID)
339 (python3 -m http.server PID)
```

- our container is now isolated regarding PIDs, UID/GIDs and mount points

- our container is now isolated regarding PIDs, UID/GIDs and mount points
- however the child namespace contains the mount subtree of its parent

- our container is now isolated regarding PIDs, UID/GIDs and mount points
- however the child namespace contains the mount subtree of its parent
- let's change root

hide mountpoints

```
# Host
apt install busybox-static

# cat chroot.sh
ROOT=/tmp/myroot

sudo rm -rf "$ROOT"
sudo mkdir -p "$ROOT"/{bin,dev,proc,sys,etc,tmp}
sudo cp $(which busybox) "$ROOT/bin/busybox"

sudo ln -s busybox "$ROOT/bin/sh"
sudo ln -s busybox "$ROOT/bin/lx"
sudo ln -s busybox "$ROOT/bin/mount"
sudo ln -s busybox "$ROOT/bin/umount"
sudo ln -s busybox "$ROOT/bin/ps"

# chmod +x chroot.sh && ./chroot.sh
```

hide mountpoints

```
# Container
# mount --make-rprivate /
# mount --bind /tmp/myroot /tmp/myroot
# chroot /tmp/myroot/ /bin/sh
BusyBox v1.37.0 (Debian 1:1.37.0-6+b9) built-in shell (ash)
Enter 'help' for a list of built-in commands.

/ # ls
bin    dev    etc    proc   sys    tmp
```

- Docker or LXC make a full working filesystem when changing root

- Docker or LXC make a full working filesystem when changing root
- we will not

- Docker or LXC make a full working filesystem when changing root
- we will not
- let's exit ours in the container: `exit`

Outline

objective

setup

service creation

cgroups

namespaces

more namespaces

conclusion

- let's add more namespaces: UTS, IPC and time

add namespaces to container

```
# Container  
exit
```

```
# Host  
sudo -u svc_websrv unshare --fork --mount --mount-proc --  
    pid --user  
--map-root-user -uts -ipc -time -- /bin/bash
```

test UTS namespace

```
# Container: change hostname
# hostname container1
# hostname
container1

# Host
# hostname
localhost
```

test IPC namespace

```
# Host  
# readlink /proc/self/ns/ipc  
ipc:[4026531839]
```

```
# Guest  
# readlink /proc/self/ns/ipc  
ipc:[4026532180]
```

- if we add a network namespace (`--net`), what happens?

- if we add a network namespace (`--net`), what happens?
- our server cannot be reached by the host anymore

- if we add a network namespace (`--net`), what happens?
- our server cannot be reached by the host anymore
- network namespaces do not share anything related to the network stack

- if we add a network namespace (`--net`), what happens?
- our server cannot be reached by the host anymore
- network namespaces do not share anything related to the network stack
- to make network namespaces communicates, virtual interfaces and bridges can be used.

Outline

objective

setup

service creation

cgroups

namespaces

more namespaces

conclusion

Attacks on our web server:

Attacks on our web server:

- what if an attacker tries a DoS?
 - cgroups-allocated resources will be full but not the entire machine, it will not crash the server

Attacks on our web server:

- what if an attacker tries a DoS?
 - cgroups-allocated resources will be full but not the entire machine, it will not crash the server
- what if an attacker gets a RCE?
 - he cannot escalate privileges via FS: mount namespace + changing root

Attacks on our web server:

- what if an attacker tries a DoS?
 - cgroups-allocated resources will be full but not the entire machine, it will not crash the server
- what if an attacker gets a RCE?
 - he cannot escalate privileges via FS: mount namespace + changing root
 - he cannot abuse host processes: PID namespace

Attacks on our web server:

- what if an attacker tries a DoS?
 - cgroups-allocated resources will be full but not the entire machine, it will not crash the server
- what if an attacker gets a RCE?
 - he cannot escalate privileges via FS: mount namespace + changing root
 - he cannot abuse host processes: PID namespace
 - even if he succeeded, he is our web service user: user namespace + service user

Going further

- <https://www.redhat.com/en/blog/mount-namespaces>
- https://man7.org/linux/man-pages/man7/network_namespaces.7.html

The End. Questions?