*Articles about OpenStack and related technologies from the RDO community*

You are here: Home ~ 2015 ~ August ~ *Hands on Linux sandbox with namespaces and cgroups*

# Hands on Linux sandbox with namespaces and cgroups

Published by Tristan Cacqueray on 13/08/2015 | 3 Responses

This article goes through Linux namespaces and cgroups usage to sandbox and isolate processes. For more detailed explanations on these Linux features, see the following articles: namespace, cgroups and resources management

Command lines should work on any modern Linux system, and you should try them out while reading to better understand the sandbox process. Most actions require admin privileges and SUDO_USER should be set to your local user.

## Introduction

The primary goal of a sandbox is to limit a process. While it’s mostly used to run non-trusted processes, it’s also very useful to isolate different contexts. This is the list of actions a process can do by default:

- A) Uses resources (cpu, memory, …)
- B) Read and write user’s files.
- C) Read and write the memory of other user’s processes.
- D) Read host configuration and access network.

Using resources limits, cgroup and namespace, this article will demonstrate how to control each of these capabilities for a sandbox.

## Resource limits

Resource limits is an old mechanism used to limit various resources like:

- nofile: maximum number of open files
- nproc: maximum number of processes
- locks: maximum number of files lock

Resource limits should always be used, new sessions can be started by using prlimit (included in util-linux-2.21):

```
prlimit --nofile=256 --nproc=512 --locks=32 /bin/bash
  # This prevent fork bomb like
  :(){ :|: &amp; };:
```

Resource limits can also be applied to current session using the ulimit tool:

```
ulimit -n 256 -u 512 -x 32
# show current limits:
ulimit -a
```

## Cgroup

Linux control groups allow fine-grained control over allocating, prioritizing, denying, managing, and monitoring system resources. Without getting too far into the details, one needs to know that cgroups are composed of different subsystems organized hierarchically. The main constraint is that two subsystems grouped together can’t be used independently. Currently, Linux offers more than ten subsystems. This article will only use:

- cpu: control cpu usage
- memory: control memory usage
- blkio: control block device usage
- devices: control device access
- freezer: control process status

For more details, see this documentation.

### Basic usage

To limit a sandbox, a new hierarchy needs to be created for each subsystem (requires libcgroup-tools):

```
# create new cgroups
cgcreate -g cpu,memory,blkio,devices,freezer:/sandbox
```

Cgroup information and configuration can be found in the virtual filesystem cgroup, usualy present in /sys/fs/cgroup:

```
# list tasks associated to the sandbox cpu group:
cat /sys/fs/cgroup/cpu/sandbox/tasks
# show the cpu share of the sandbox cpu group:
cat /sys/fs/cgroup/cpu/sandbox/cpu.shares
# kill all of sandbox's processes
kill -9 $(cat /sys/fs/cgroup/cpu/sandbox/tasks)
```

### Freezer

The freezer subsystem suspends or resumes tasks:

```
# To freeze a group
cgset -r freezer.state=FROZEN sandbox
# To unfreeze a group
cgset -r freezer.state=THAWED sandbox
# Get the status
cgget -r freezer.state sandbox
```

This has the effect of completely locking the sandbox and pausing all of the processes.

### Cpu

The cpu subsystem controls compute resource allocation. CFS periods and quotas can effectively control a process's cpu usage. The quota sets how much cpu time a process can use during each period. Note that this does not take into account the number of cpu cores.

Using a period of 100ms, here are a few example:

```
# Allows only 1ms every 100ms to simulate a slow system
cgset -r cpu.cfs_period_us=100000 -r cpu.cfs_quota_us=1000 sandbox

# Limit usage at 10% for a multi core system
cgset -r cpu.cfs_period_us=100000 \
     -r cpu.cfs_quota_us=$[ 10000 * $(getconf _NPROCESSORS_ONLN) ] \
           sandbox
```

This is specially useful to prevent a sandbox from hogging all cpu ressources.

### Memory

The memory subsystem controls memory resource allocation:

```
# Set a limit of 2Gb
cgset -r memory.limit_in_bytes=2G sandbox

# Get memory stats used by the cgroup
cgget -r memory.stat sandbox
```

This prevents the sandbox from reserving all memory.

### Block I/O

The blkio subsystem controls and monitors access to I/O on block devices:

```
# Limit to 1MB/s for read and write on common hard drive device
for dev in 253:0 252:0 252:16 8:0 8:16 1:0; do
  cgset -r blkio.throttle.read_bps_device="${dev} 1048576" sandbox
  cgset -r blkio.throttle.write_bps_device="${dev} 1048576" sandbox
done
```

This limit the sandbox bandwith for hard drive read and write operations.

### Devices

The device subsystem allows or denies access to devices:

```
# Deny access to devices
cgset -r devices.deny=a sandbox

# Allow access to console, null, zero, random, unrandom
for d in "c 5:1" "c 1:3" "c 1:5" "c 1:8" "c 1:9"; do
  cgset -r devices.allow="$d rw" sandbox
done
```

This prevents the sandbox from accessing host devices.

### Sandbox using Linux cgroups

All together, the cgroups proposed above limit the following system resources:

- 10% of total cpu resources
- 2Gb of memory
- 1Mb/s of block i/o
- no devices access

Here is a demo of the sandbox:

```
# join the cgroups
cgexec -g cpu,memory,blkio,devices,freezer:/sandbox \
  prlimit --nofile=256 --nproc=512 --locks=32 /bin/bash

# check that sandbox groups are assigned
cat /proc/self/cgroup

# On a 4 cores systems, a single thread can't load more than 40%
python -c "while True: 42*42"
# Check with top cpu load.

# block access speeds should be around 1 MB/s
dd if=zero_file of=/dev/zero iflag=direct bs=4k count=1024

# direct access to devices are not allowed
head -c 512 /dev/sda
```

From another session, the sandbox can be terminated with:

```
# freeze the sandbox
cgset -r freezer.state=FROZEN sandbox

kill -9 $(cat /sys/fs/cgroup/cpu/sandbox/tasks)
# unfreeze
cgset -r freezer.state=THAWED sandbox
```

In brief, Linux cgroups offers good sandbox mechanisms like hardware usage limitation and process controls. It mitigates the capability A) Uses resources (cpu, memory, …) mentioned in the introduction. The next chapter presents how namespaces can be used to further restrict a sandbox.

## Namespaces

Linux namespaces are sets of system resources that can be unique for a process. Currently, Linux implements six namespaces:

- Mounts namespace: filesystem mount points
- PID namespace: process id number
- IPC namespace: inter process communication resources as shown by the "ipcs" command
- UTS namespace: hostname and domain name
- Network namespace: network resources
- User namespace: user and group id numbers

Linux namespaces can be independently assigned, the "unshare" utility is used to create processes with new namespaces.

## Filesystem isolation

Using the mounts namespace, the host filesystem can be isolated from the sandbox. This example will replace the /home directory by /home/sandbox:

```
# Create a new home directory tree
mkdir -p /home/sandbox/${SUDO_USER}
chown ${SUDO_USER} /home/sandbox/${SUDO_USER}
# Make sure / is private
mount --make-rprivate /
# Create a new root shell with the mount namespace
unshare --mount
  # New mount will be local to this session
  mount -o bind /home/sandbox /home
  # Other directories can be replaced e.g., /tmp
  mount -t tmpfs none /tmp
  # Drop privilege back to user
  su -l ${SUDO_USER}
```

This sandbox has a dedicated /home and /tmp directory. X11 access is denied because the sandbox does not have access to the ~/.Xauthority file nor the unix socket (/tmp/.X11-unix). This offers a good counter measure against capability B) and can be further extended by replacing more directories before dropping privilege.

## Process isolation

Using the mounts, PID and IPC namespaces, the sandbox can be confined to its own process tree. Process ID namespace (–pid) will make the first process start with the pid 1. In order to hide the host process, a new proc filesystem still needs to be mounted inside the new mounts namespace (–mount-proc=/proc). Finaly, to make this easier, the new process will be forked first (–fork). All put together:

```
unshare --mount --ipc --pid --mount-proc=/proc --fork \
  su -l ${SUDO_USER}
```

This sandbox has a dedicated process tree (/proc) that will deny capability C).

## Networking isolation

Using the network namespace, the sandbox can be isolated from host networks devices, routes and the firewall:

```
unshare --net su -l ${SUDO_USER}
```

This sandbox does not have any network connectivity.

In order to get network access, the namespace needs to be created using the ip utility. Network namespaces are persistent and needs to be destroyed explicitly:

```
# create a new namespace
ip netns add sandbox
ip netns exec sandbox su -l ${SUDO_USER}
```

This basic configuration should enable external network access:

```
# configure sandbox loopback
ip netns exec sandbox ip addr add 127.0.0.1/8 dev lo
ip netns exec sandbox ip link set lo up
# create a device pairs
ip link add sandbox0 type veth peer name sandbox1

# initiate the host side
ip link set sandbox0 up
# initiate the container side
ip link set sandbox1 netns sandbox up

# configure network
ip addr add 192.168.242.1/30 dev sandbox0
ip netns exec sandbox ip addr add 192.168.242.2/30 dev sandbox1
ip netns exec sandbox ip route add default via 192.168.242.1 dev sandbox1

# enable routing
echo 1 | tee /proc/sys/net/ipv4/ip_forward
ext_if=$(ip route get 8.8.8.8 | grep 'dev' | awk '{ print $5 }')
iptables -I POSTROUTING -t nat -s 192.168.242.2/32 -o ${ext_if} -j MASQUER
iptables -I FORWARD -i sandbox0 -o ${ext_if} -j ACCEPT
iptables -I FORWARD -i ${ext_if} -o sandbox0 -j ACCEPT

# configure resolv.conf
mkdir -p /etc/netns/sandbox
echo nameserver 8.8.8.8 | tee /etc/netns/sandbox/resolv.conf
```

Note that dedicated resolv.conf and hosts files can be assigned in /etc/netns.

Network namespace provides complete network isolation and using virtual networks, the traffic can be further limited to prevent capability D).

## X11 sandbox

Graphical applications need special care to be isolated. A direct access to the X11 server is not recommended since it gives the sandbox access to the keyboard events and the list of running applications. A dedicated X11 instance can be used with Xephyr and mount namespaces:

```
# Create new home directory
mkdir -p /home/sandbox/${SUDO_USER}
# Create a new cookie
echo -n | tee /home/sandbox/${SUDO_USER}/.Xauthority
xauth -f /home/sandbox/${SUDO_USER}/.Xauthority add :4 . $(mcookie)
chown -R ${SUDO_USER} /home/sandbox/${SUDO_USER}/

# Create a nest with a window manager on :4
su ${SUDO_USER} -c "Xephyr -screen 1024x768 -auth /home/sandbox/${SUDO_USE
unshare --mount /bin/sh -c "
  # Remount /home
  mount -o bind /home/sandbox /home
  # minimal startx
  export DISPLAY=:4
  export XAUTHORITY=/home/{SUDO_USER}/.Xauthority
  su -l ${SUDO_USER} -c 'twm &amp; xclock; firefox'
"
```

This sandbox does not have access to the host session :0 and is confined to a virtual screen.

## Use cases

### All together

Example usage of cgroups, namespaces and resource limits:

```
# Create cgroups
cgcreate -g cpu,memory,blkio,devices,freezer:/sandbox
# Allows only 1ms every 100ms to simulate a slow system
cgset -r cpu.cfs_period_us=100000 -r cpu.cfs_quota_us=1000 sandbox
# Set a limit of 2Gb
cgset -r memory.limit_in_bytes=2G sandbox
# Limit block I/O to 1MB/s
for dev in 253:0 252:0 252:16 8:0 8:16 1:0; do
  cgset -r blkio.throttle.read_bps_device="${dev} 1048576" sandbox
  cgset -r blkio.throttle.write_bps_device="${dev} 1048576" sandbox
done
# Deny access to devices
cgset -r devices.deny=a sandbox
# Allow access to console, null, zero, random, unrandom
for d in "c 5:1" "c 1:3" "c 1:5" "c 1:8" "c 1:9"; do
  cgset -r devices.allow="$d rw" sandbox
done

# Create network namespace
ip netns add sandbox

# Join cgroup, netns and activate resources limit
cgexec -g cpu,memory,blkio,devices,freezer:/sandbox   \
  prlimit --nofile=256 --nproc=512 --locks=32         \
    ip netns exec sandbox                             \
      unshare --mount --uts --ipc --pid --mount-proc=/proc --fork sh -c "
        mount -t tmpfs none /home
        mount -t tmpfs none /tmp
        mount -t tmpfs none /sys
        mount -t tmpfs none /var/log
        exec su -l ${SUDO_USER}
      "
```

While this sandbox is very minimal, it shows how to use cgroups, namespaces and resource limits together.

### Scripted usage

To automatically setup sandbox within a script, one should know that:

- Namespaces are represented by file handles. They are created at process creation with special clone(2) flags.
- To join an existing namespace, a process needs to call setns(2) with a valid namespace file handle.
- Namespaces handles can be obtain from /proc//ns/ directory.
- The tool “nsenter” can create a new process using namespaces of other processes.
- Named network namespaces created by the ip utility are kept persistent in /run/netns.
- Once destroyed, a mount namespace will undo all local mounts.
- The root filesystem needs to be mounted as private (mount –make-rprivate /) to prevent remount to propagate to the host.
- A specific cgroup can be joined by writting a process id to a /sys/fs/cgroups tasks file.

The main difficulty being that new namespaces are created with the clone syscall and to properly setup the environment, more actions are required from within the child process.

### Developpement sandbox

Openstack developpment environment can benefit from sandbox:

- a dedicated home directory allows to securely store commit keys, gitconfig and source codes.
- system and network isolation will prevent unwanted operation on the host like running Horizon on a public interfaces.

### Web browser sandbox

Web browsers can be really messy and to mitigate most common issues, a different sandbox can be used per context. For example, an openstack developer browser can be protected by only allowing access to launchpad.net and openstack.org domain using iptables.

### Vpn sandbox

The openvpn process can be ran in a complete sandbox with a specific network namespace. Network access can be restricted to the Vpn remote endpoint in order to prevent direct Internet access. Then applications can be ran in separate sandboxs but using the same network namespace.

## Clean-up

All resources created by previous examples can be removed with:

```
# destroy cgroups
cgdelete -g cpu,memory,blkio,devices,freezer:/sandbox

# destroy network namespace
ip netns delete sandbox
iptables -D POSTROUTING -t nat -s 192.168.42.2/32 -o ${ext_if} -j MASQUERA
iptables -D FORWARD -i sandbox0 -o ${ext_if} -j ACCEPT
iptables -D FORWARD -i ${ext_if} -o sandbox0 -j ACCEPT
rm -Rf /etc/netns/sandbox
```

## Conclusion

This article covered sandbox techniques using Linux. Namespaces in particular enables a process to be run in a unique environment, allowing multiple isolated instances of an application.

This is only effective for user process, as it does not prevent a privileged process from escaping the sandbox. Better isolation requires dedicated hardware and/or virtualisation.

Food for thought:

- Uname (kernel name, version and build time) can’t be isolated and requires a fake uname syscall to be preloaded
- Grsecurity can harden /proc and /sys filesystem to deny access to more system information
- Sandbox provides an easy way to audit suspicious activies
- Mandatory access controls can be used to further limit capabilities

**Share this:**

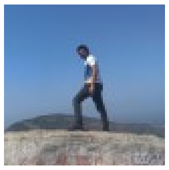
More

Posted in Infrastructures
Tagged cgroups, linux, namespaces, sandbox

### Article written by Tristan Cacqueray

Tristan is a security Engineer at eNovance and member of the Vulnerability Management Team of OpenStack. He has been actively working in open source security for the past six years. He's also known to troll about SELinux or systemd.

### 3 Responses



***Phani Kumar Yadavilli***
31/10/2016 at 06:03 | Permalink

The blog post was very informative and helpful. Could you please let me know where can i download the sandbox.

***Onur Sehitoglu***
06/12/2017 at 20:49 | Permalink

Thank you. This is a really useful and clear tutorial. I had problem in prlimit nproc though. Since I try the sandbox with my desktop user, my current processes already reached the nproc limit so I had difficulties. Than I found a newer (than this article) controller called pids limitting the number of processes per cgroup. pids.max does the job.

Thanks again.

***How to run Docker containers using common Linux tools (without Docker) – I Learned How To…***
13/12/2017 at 17:26 | Permalink

[…] Full post at RDO that explains basics of cgroups, unshares, etc. and puts all together. […]

Comments are closed.

← Previous Next →

## RDO on Twitter

My Tweets

## Top Posts & Pages

- Writing a SELinux policy from the ground up
- Hands on Linux sandbox with namespaces and cgroups
- Dive into Zuul – Gated commit system
- Build your RESTful API web-service in 5 minutes
- ZooKeeper Part 1: the Swiss army knife of the distributed system engineer

## Authors

- amoralej
- apevec
- atc
- Babu Shanmugam
- chandankumar
- chkumar246
- Chmouel Boudjnah
- Christian Schwede
- Cyril Roelandt
- Dan Radez
- dneary
- emalikova
- Emilien Macchi
- Erwan Velu
- Fabien Boucher
- Frederic Lepied
- Gonéri Le Bouder
- Graeme Gillies
- Haïkel Guémar
- Ignace Mouzannar
- James Kulina
- Joe H. Rahme
- jpena
- jruzicka
- Julien Danjou
- Laura ZANETTI
- ltoscano
- Martin Kopec

- Mary Thengvall
- mattdm
- Matthieu Huin
- Mehdi Abaakouk
- Nick Barcet
- Nicolas Hicher
- Nicolas Planel
- Numan Siddique
- Petr Kovar
- Pierre Mavro
- pixelbeat
- pmkovar
- pmyers
- Rain Leander
- Rich Bowen
- rkukura
- Sebastien Han
- Software Factory Team
- Sridhar Gaddam
- Sylvain Afchain
- tdecacqu
- Thomas Herve
- Tristan Cacqueray
- trown
- Victor Stinner
- whayutin
- Yanis Guenane
- Yassine Lamgarchal

## Categories

- Ceph
- Community Blog Round-up
- Conferences
- Deployment
  - CI
- Event
- Infrastructures
  - Agile
- OpenStack
  - Aodh

- Ceilometer
    - Gnocchi
    - Heat
    - Keystone
    - Neutron
    - Nova
    - Swift
    - Tempest
    - TripleO
- Puppet
- Python

## Tag Cloud

acronym asyncio authorization brussels budapest CD ceilometer centos cgroups configure cost design summit dojo edeploy erlang gerrit git glance grafana havana havana summit hongkong Interoperability manageIQ Monitoring networking nfv novajoin object storage opencontrail openstack.brussels openstack.mitaka openstacksummit organisation patches prague press sig storage swift3 sydney team test.documentation troubleshoot virtualenv

## Search

RSS - Posts

Back to Top ▲