

Instantly share code, notes, and snippets.

udzura / **seckun-01-haconiwa.md** Secret

Last active 18 days ago

☆ Star

<> Code ↻ Revisions 15

URL共有はお控えください。

 seckun-01-haconiwa.md

SECKUN 2020 コンテナ演習（後半）手順書

- 進捗報告スプレッドシートはこちら:
 - （後半）に記入をお願いします

参考文献

- mruby について
 - Webで使えるmrubyシステムプログラミング入門 (近藤宇智朗, <https://www.c-r.com/book/detail/1364>)
- 関連するLinuxの機能（特に、コンテナ関連、プロセス周辺）について
 - Web記事
 - <https://eh-career.com/engineerhub/entry/2019/02/05/103000>
 - <https://container-security.dev/>
 - https://gihyo.jp/admin/serial/01/linux_containers/0001 からの連載記事
 - なるほどUNIXプロセス (Jesse Storimer, 島田浩二ほか, <https://tatsu-zine.com/books/naruhounix>)
 - コンテナ型仮想化概論 (川口 直也, <http://www.cutt.co.jp/book/978-4-87783-478-4.html>)
 - 詳解UNIXプログラミング (<https://www.seshop.com/product/detail/16746>)

- Linuxプログラミングインタフェース
(<https://www.oreilly.co.jp/books/9784873115856/>)

上述の書籍・記事に書いてある内容は、深いところまではフォローして解説しません。

Haconiwaのビルド

前回 (<https://gist.github.com/udzura/fa82a8cb49a611c5f6400e561cf38a1d>) 作ったVMをそのまま使いましょう。

必要なパッケージのインストール

```
$ sudo apt install build-essential util-linux rake psmisc bison git flex make cri
gperf cgroup-tools libpam0g-dev \
autotools-dev libcap-dev libargtable2-dev parallel systemtap-sdt-dev \
libtool
```

ビルド実行

- mrubyは2.1.2を用います。mruby 3で入る新機能はシステムプログラミングに限って言えばそこまでの意義を持たないので、トラブルを避ける意味で2系を用います。

```
$ git clone https://github.com/haconiwa/haconiwa.git
$ cd haconiwa
$ git fetch origin && git checkout seckun
$ rake
```

- 注意
 - チェックアウトするネットワーク環境によってはgit cloneなどがタイムアウトします、そういうエラーメッセージの場合は何度か rake を実行してください。
 - どうしてもビルドできない場合、講師がこっそりバイナリを配ります...

動作確認

```
$ ./mruby/bin/mruby -e 'p Haconiwa::VERSION'
"0.11.5"
```

```
$ ./mruby/bin/mirb
mirb - Embeddable Interactive Ruby Shell
```

```
> p Haconiwa::VERSION
"0.11.5"
=> "0.11.5"
> exit
```

以下では、mrubyスクリプトによるプログラミングを行います。

プロセスの作成

- 基本となります。CRubyとあまり変わりありません。

```
pid = Process.fork do
  Exec.execve ENV.to_hash, "/bin/sh"
end

_, s = Process.waitpid2(pid)
p s
```

- 新しくシェルを起動していることを確認。

```
# 起動
$ ./mruby/bin/mruby mycontainer.rb
$ ps auxf
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root             2  0.0  0.0      0     0 ?        S      Dec13    0:00 [kthreadd]
root             3  0.0  0.0      0     0 ?        I<     Dec13    0:00 \_ [rcu_gp]
...
root           881  0.0  0.0  12184  7316 ?        Ss     Dec13    0:00 sshd: /usr/sbi
root        175131  0.0  0.1  13800  8852 ?        Ss     Dec17    0:00 \_ sshd: vagr
vagrant    175246  0.0  0.0  13960  6208 ?        S      Dec17    0:01 | \_ sshd:
vagrant    175247  0.0  0.0  12216  6680 pts/1    Ss     Dec17    0:00 | \_ -t
vagrant    248825  0.4  0.0   8704  5460 pts/1    S      13:02    0:00 | \
vagrant    248826  0.0  0.0   2616   612 pts/1    S      13:02    0:00 |
vagrant    248827  0.0  0.0  11832  3772 pts/1    R+     13:02    0:00 |
...
$ exit
#<Process::Status: pid=248826,exited(0)>
```

ファイルシステムの隔離、Linux Namespaceの隔離

chroot(2)

- ファイルツリーを何処か別のところに作成します。 `docker export` などを使えばいいでしょう。

```
$ sudo mkdir /tmp/myroot
$ sudo docker pull debian:10-slim
$ sudo docker run -d debian:10-slim sleep inf
50c3bc0867a1adfdc4b1b8814520a6455055869287db3e3daf216685535d26c4
$ sudo docker export 50c3bc08 | sudo tar xvf - -C /tmp/myroot
```

- コマンドでchrootする

```
$ sudo chroot /tmp/myroot
root@ubuntu2004:/# ls -l
total 76
drwxr-xr-x  2 root root 4096 Dec  9 23:22 bin
drwxr-xr-x  2 root root 4096 Nov 22 12:37 boot
drwxr-xr-x  4 root root 4096 Dec 18 13:05 dev
drwxr-xr-x 28 root root 4096 Dec 18 13:05 etc
drwxr-xr-x  2 root root 4096 Nov 22 12:37 home
drwxr-xr-x  7 root root 4096 Dec  9 23:22 lib
drwxr-xr-x  2 root root 4096 Dec  9 23:22 lib64
drwxr-xr-x  2 root root 4096 Dec  9 23:22 media
drwxr-xr-x  2 root root 4096 Dec  9 23:22 mnt
drwxr-xr-x  2 root root 4096 Dec  9 23:22 opt
drwxr-xr-x  2 root root 4096 Nov 22 12:37 proc
drwx----- 2 root root 4096 Dec  9 23:22 root
drwxr-xr-x  3 root root 4096 Dec  9 23:22 run
drwxr-xr-x  2 root root 4096 Dec  9 23:22 sbin
drwxr-xr-x  2 root root 4096 Dec  9 23:22 srv
drwxr-xr-x  2 root root 4096 Nov 22 12:37 sys
drwxrwxrwt  2 root root 4096 Dec  9 23:22 tmp
drwxr-xr-x 10 root root 4096 Dec  9 23:22 usr
drwxr-xr-x 11 root root 4096 Dec  9 23:22 var
root@ubuntu2004:/# ls /proc/
root@ubuntu2004:/#
```

- 入れはするが、必要なファイルシステムをマウントしていないためほぼ何もできない。

発展課題: 難しければスキップ推奨です

- 調べてみよう: `jail(2)`, `jail(8)` <https://www.freebsd.org/cgi/man.cgi?query=jail&apropos=0&sektion=2>

clone(2) とLinux Namespaceに関するフラグ

- Linux Namespaceとは

ref: <https://container-security.dev/namespace/>

Namespace には Linux 5.9 の段階では、次の8つがあります。

Namespace	概要
Cgroup	Namespace ごとに cgroup を作成する (Linux 4.6 から)
IPC	IPC や POSIX message queues など分離
Network	ネットワークデバイスやアドレスなどを分離
Mount	ファイルシステムを分離
PID	プロセスID を分離する (Linux 3.8 から)
Time	システムクロックの一部を分離する (Linux 5.6 から)
User	UID / GID を分離する (Linux 3.8 から)
UTS	hostname を分離する

- Mount Namespaceを分離する。一緒に、必要なファイルシステムや設定をマウント。
- 分離をするのに関係するシステムコールは3つあります
 - unshare(2) 。呼び出したプロセスを新しく作った名前空間に所属させる。ただ、PID名前空間などは扱いが特殊でコツがいる
 - clone(2) 。 fork(2) や pthread_create(2) が内部で利用する、様々な条件で新規スレッド/プロセスを作成するシステムコール。フラグの中に、新規プロセスにおいて名前空間を分離するためのものが含まれる。
 - setns(2) 。呼び出したプロセスを、既存の名前空間に所属させる。
 - clone(2) にフラグを渡すか、forkとexecの間で unshare/setns(2) を呼び出すというのが基本的な実装。今回は前者。

```
pid = Namespace.clone(Namespace::CLONE_NEWNS) do
  newroot = "/tmp/myroot"
  # おまじない...
  Mount.make_rprivat "/"
  # procfsのマウント
  Mount.mount("proc", "#{newroot}/proc", type: "proc")
  # 名前解決できるように...
  Mount.bind_mount("/etc/resolv.conf", "#{newroot}/etc/resolv.conf")

  # chroot
  Dir.chroot newroot
  # カレントディレクトリを「存在するところ」にする
  Dir.chdir '/'
  # sh は使いづらいので...
  Exec.execve ENV.to_hash, "/bin/bash"
end

_, s = Process.waitpid2(pid)
p s
```

```
$ sudo ./mruby/bin/mruby mycontainer.rb
root@ubuntu2004:/#
root@ubuntu2004:/# apt update
root@ubuntu2004:/# apt install procps iproute2 iputils-ping
root@ubuntu2004:/# ps aux | head
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           2  0.0  0.0      0     0 ?        S      Dec13   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<     Dec13   0:00 \_ [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<     Dec13   0:00 \_ [rcu_par_gp]
root           6  0.0  0.0      0     0 ?        I<     Dec13   0:00 \_ [kworker/0]
root           9  0.0  0.0      0     0 ?        I<     Dec13   0:00 \_ [mm_percpu_wq]
root          10  0.0  0.0      0     0 ?        S      Dec13   0:00 \_ [ksoftirqd/0]
root          11  0.0  0.0      0     0 ?        I      Dec13   0:13 \_ [rcu_sched]
root          12  0.0  0.0      0     0 ?        S      Dec13   0:04 \_ [migration_thread]
root          13  0.0  0.0      0     0 ?        S      Dec13   0:00 \_ [idle_injectd]
...
```

```
root@ubuntu2004:/# apt install lsb-release
root@ubuntu2004:/# lsb_release -a
No LSB modules are available.
Distributor ID: Debian
Description:    Debian GNU/Linux 10 (buster)
Release:        10
Codename:       buster
```

- ようやくコンテナらしく操作ができるようになった。また、コンテナを抜けても親プロセスの `mount` の結果に影響がないことを確認する。
- いくつか分離するNamespaceを追加。

```
pid = Namespace.clone(
  Namespace::CLONE_NEWNS | Namespace::CLONE_NEWUTS | Namespace::CLONE_NEWPID
) do
  newroot = "/tmp/myroot"
  # おまじない...
  # procfsのマウント
  # 名前解決できるように...

  # chroot
  Dir.chdir newroot
  Dir.chdir '/'

  # hostname変える
  system "hostname seckun-1.local"
  Exec.execve ENV.to_hash, ["/bin/bash"]
end

_, s = Process.waitpid2(pid)
p s
```

```
$ sudo ./mruby/bin/mruby mycontainer.rb
root@seckun-1:/#
root@seckun-1:/# hostname
seckun-1.local
root@seckun-1:/# ps auxf
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0   4004  3228 ?        S    13:26   0:00 /bin/bash
root           5  0.0  0.0   7644  2824 ?        R+   13:27   0:00 ps auxf
```

- 以下のことを確認しよう:
 - ホストの、コンテナの外側のプロセスを見ることができない
 - コンテナでhostnameなど変更しても、ホストに影響を与えない。

ネットワークの分離

- ホスト分離されたネットワークを作成し、そこにアタッチしてみよう
- netns の手作りする。コマンドから:

```
$ sudo ip netns add seckun-1
$ sudo ip link add name eth1 type veth peer name guest-eth1
$ sudo ip link set guest-eth1 netns seckun-1
$ sudo ip addr add 10.100.100.1/24 dev eth1
$ sudo ip link set eth1 up
$ sudo ip netns exec seckun-1 \
    ip addr add 10.100.100.2/24 dev guest-eth1
$ sudo ip netns exec seckun-1 \
    ip link set guest-eth1 up
```

- ip がつくのを確認

```
$ ip a s eth1
10: eth1@if9: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state UP group default ql
    link/ether 6e:e9:07:96:1f:74 brd ff:ff:ff:ff:ff:ff link-netns seckun-1
    inet 10.100.100.1/24 scope global eth1
        valid_lft forever preferred_lft forever

$ sudo ip netns exec seckun-1 ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state UP group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
9: guest-eth1@if10: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group de
    link/ether 0a:67:12:6c:ff:df brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.100.100.2/24 scope global guest-eth1
        valid_lft forever preferred_lft forever
```

- setns(2) を用いてアタッチする。


```
pid = Namespace.clone(
  Namespace::CLONE_NEWNS|Namespace::CLONE_NEWUTS|Namespace::CLONE_NEWPID
) do
  Namespace.setns Namespace::CLONE_NEWNET, fd: File.open("/var/run/netns/seckun-1
  # 以下同じ...
end
```

- ネットワークを独立して持っていることを確認。

```
$ sudo ./mruby/bin/mruby mycontainer.rb
root@seckun-1:/# ping 10.100.100.1
PING 10.100.100.1 (10.100.100.1) 56(84) bytes of data.
64 bytes from 10.100.100.1: icmp_seq=1 ttl=64 time=0.106 ms
64 bytes from 10.100.100.1: icmp_seq=2 ttl=64 time=0.051 ms
...
root@seckun-1:/# ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
9: guest-eth1@if10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue stat
    link/ether 0a:67:12:6c:ff:df brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.100.100.2/24 scope global guest-eth1
        valid_lft forever preferred_lft forever
    inet6 fe80::867:12ff:fe6c:ffdf/64 scope link
        valid_lft forever preferred_lft forever
root@seckun-1:/# ip route
10.100.100.0/24 dev guest-eth1 proto kernel scope link src 10.100.100.2
```

発展課題: 難しければスキップ推奨です

- ブリッジを作りIPマスカレードを設定すれば外に出られます。手順を考えてみましょう

pivot_root(2)

- chroot(2) は権限によるがunjailができるので、 pivot_root(2) を使うようにする。

```
pid = Namespace.clone(
  Namespace::CLONE_NEWNS|Namespace::CLONE_NEWUTS|Namespace::CLONE_NEWPID
) do
  Namespace.setns Namespace::CLONE_NEWNET, fd: File.open("/var/run/netns/seckun-1

  Mount.make_rprivate "/"
  newroot = "/tmp/myroot"
  # chrootをこれに変える
  # pivot_root する先は何かしらのマウントポイントでないとけない。
  Mount.bind_mount newroot, newroot
```



```

Haconiwa.pivot_root_to newroot, true
Dir.chdir '/'

# procfsのマウント
Mount.mount("proc", "/proc", type: "proc")

# hostname変える
system "hostname seckun-1.local"
Exec.execve ENV.to_hash, "/bin/bash"
end

_, s = Process.waitpid2(pid)
p s

$ sudo ./mruby/bin/mruby mycontainer.rb
root@seckun-1:/# mount
/dev/sda3 on / type ext4 (rw,relatime,errors=remount-ro)
proc on /proc type proc (rw,relatime)

```

- chrootでunjailができること、pivot_rootでできないことを確認しよう。

ここまででひとこと

- ここまででコンテナで最も重要な箇所は実装できてます！ あと半分頑張りましょう。

リソース利用の制御

cgroup の概要

ref: <https://container-security.dev/cgroup/>
 cgroup はプロセスをグループ化し、そのグループに属するプロセスに対してリソースの制限を行う仕組みです。
 cgroup は cgroupfs と呼ばれるファイルシステムを通して操作します。多くは /sys/fs/cgroup にマウントされています。

制限できるリソースのことをサブシステムと呼び、CPU のコア数やメモリ使用量、プロセス数などを制限できます。

- コマンドラインで試す (v1です)

```

$ sudo mkdir /sys/fs/cgroup/cpu/seckun-1
$ sudo ls /sys/fs/cgroup/cpu/seckun-1
cgroup.clone_children  cpuacct.usage          cpuacct.usage_percpu_sys  cpuacct.u
cgroup.procs           cpuacct.usage_all       cpuacct.usage_percpu_user  cpu.cfs_p
cpuacct.stat           cpuacct.usage_percpu    cpuacct.usage_sys          cpu.cfs_c

```

```
$ bash # シェル新しく立ち上げる
$ echo $$ | sudo tee /sys/fs/cgroup/cpu/seckun-1/tasks
251170
$ sudo cat /proc/self/cgroup | grep cpu
5:cpu,cpuacct:/seckun-1
4:cpuset:/
```

- プロセスが所属している。
- CPU制限をかける。

```
$ cat /sys/fs/cgroup/cpu/seckun-1/cpu.cfs_period_us
100000
$ echo 20000 | sudo tee /sys/fs/cgroup/cpu/seckun-1/cpu.cfs_quota_us
20000
$ cat /sys/fs/cgroup/cpu/seckun-1/cpu.cfs_quota_us
20000
```

- CPUを使う。

```
$ ruby -e 'loop { 2 ** 100 }'
```

- 別のターミナルでtopを打つと:

```
top - 14:22:18 up 5 days, 59 min, 3 users, load average: 1.12, 1.03, 1.01
Tasks: 218 total, 2 running, 216 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.8 us, 0.0 sy, 0.0 ni, 98.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 7959.5 total, 1000.9 free, 678.2 used, 6280.4 buff/cache
MiB Swap: 1953.0 total, 1951.7 free, 1.3 used. 6992.4 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
 251210 vagrant   20   0   80504  22080  5344 R   19.9   0.3   0:06.72 ruby
 173216 root       20   0    7828   5212  3400 S    1.0   0.1  19:13.22 runmh
 162077 root       20   0 1858780 48028 26996 S    0.3   0.6   8:35.97
containerd
    1 root       20   0 169796 13820  8300 S    0.0   0.2   0:09.93 systemd
...
```

発展課題: 難しければスキップ推奨です

- メモリも同じように制限をかけられます。調べよう

mruby製コンテナでのcgroupの利用

- 1 - libcgroupのラッパを使う。
- 2 - ファイル操作のAPI (Dir.mkdir/File#read/File#write) を使ったり、コマンドを打っちゃう。

後者の作戦で簡単に。

```
system "mkdir -p /sys/fs/cgroup/cpu/seckun-1"

pid = Namespace.clone(
  Namespace::CLONE_NEWNS|Namespace::CLONE_NEWUTS|Namespace::CLONE_NEWPID
) do
  File.open("/sys/fs/cgroup/cpu/seckun-1/tasks", "w") do |f|
    f.write Process.pid.to_s
  end
  File.open("/sys/fs/cgroup/cpu/seckun-1/cpu.cfs_quota_us", "w") do |f|
    f.write "20000"
  end

  Namespace.setns Namespace::CLONE_NEWNET, fd: File.open("/var/run/netns/seckun-1
  Mount.make_rprivate "/"
  newroot = "/tmp/myroot"
  # chrootをこれに変える
  # pivot_root する先は何かしらのマウントポイントでないといけない。
  Mount.bind_mount newroot, newroot
  Haconiwa.pivot_root_to newroot, true
  Dir.chdir '/'

  # procfsのマウント
  Mount.mount("proc", "/proc", type: "proc")
  # /dev/null を生やす
  Mount.mount("dev", "/dev", type: "devtmpfs")

  # hostname変える
  system "hostname seckun-1.local"
  Exec.execve ENV.to_hash, "/bin/bash"
end
...
```

- 制限を確認する。別のターミナルでtopを開こう。

```
$ sudo ./mruby/bin/mruby mycontainer.rb
root@seckun-1:/# yes >/dev/null
```

発展課題: 難しければスキップ推奨です

- 古典的なresource limit (ulimit(1)) の挙動も調べよう。

権限の制限

Linux Capabilities

ref: <https://container-security.dev/capability/>

Linux には Capability と呼ばれる仕組みがあり、ファイル及びプロセスに対して権限を細かく設定することができます。

- rootは「すべての」システム操作のための権限を持っているが、一部の権限で十分な場合があったり、逆に一部のみを制限したい必要がある。コンテナであれば、
 - 1024番以下のポートをリッスンしたい、など (setuidでもできますがまあ...)
 - 一方、時刻の操作やカーネルモジュールのインストール権限などは不要
- という場合など。
 - 詳細: <https://man7.org/linux/man-pages/man7/capabilities.7.html>

$$P'(\text{ambient}) = (\text{file is privileged}) ? 0 : P(\text{ambient})$$

$$P'(\text{permitted}) = (P(\text{inheritable}) \& F(\text{inheritable})) \mid (F(\text{permitted}) \& P(\text{bounding})) \mid P'(\text{ambient})$$

$$P'(\text{effective}) = F(\text{effective}) ? P'(\text{permitted}) : P'(\text{ambient})$$

$$P'(\text{inheritable}) = P(\text{inheritable}) \quad [\text{i.e., unchanged}]$$

$$P'(\text{bounding}) = P(\text{bounding}) \quad [\text{i.e., unchanged}]$$

where:

$P()$ denotes the value of a thread capability set before the `execve(2)`

$P'()$ denotes the value of a thread capability set after the `execve(2)`

$F()$ denotes a file capability set

- rootから特定の特権を奪う、という観点では
 - `fork(2)` する
 - Bounding setから `cap_xxx` を落とす
 - `exec(2)` する
 - という手順を踏むと、新しいプログラムでは `cap_xxx` が奪われる。
 - 詳細な解説 (これでもざっくりなので...、しかも難解なので、飛ばしてもOK)
 - 上記の表において、 $P'(\text{permitted}) = (P(\text{inheritable}) \& F(\text{inheritable})) \mid (F(\text{permitted}) \& P(\text{bounding})) \mid P'(\text{ambient})$
 - rootでの実行の場合 = (空集合 & 全集合) \mid (全集合 & $P(\text{bounding})$) \mid 空集合

- よって多くの場合、execveの後のプロセスのpermitted setはその前のbounding setに一致する
 - cf. ping バイナリの file capability
- 例として、「時刻操作ができないroot user」のシェル:

```
$ sudo capsh --drop=cap_sys_time -- -l
WARNING: libcap needs an update (cap=39 should have a name).
root@ubuntu2004:/home/vagrant/haconiwa# date
Sun 20 Dec 2020 07:17:34 AM UTC
root@ubuntu2004:/home/vagrant/haconiwa# date -s '1970-01-01'
date: cannot set date: Operation not permitted
Thu 01 Jan 1970 12:00:00 AM UTC
root@ubuntu2004:/home/vagrant/haconiwa# date
Sun 20 Dec 2020 07:17:44 AM UTC
root@ubuntu2004:/home/vagrant/haconiwa# id
uid=0(root) gid=0(root) groups=0(root)
```

- mrubyではmruby-capabilityが定義する Capability.drop_bound が使える。

```
pid = Namespace.clone(
  Namespace::CLONE_NEWNS|Namespace::CLONE_NEWUTS|Namespace::CLONE_NEWPID
) do
  # ...

  # Capability
  Capability.drop_bound(Capability::CAP_SYS_TIME)
  # Add your own config...

  Exec.execve ENV.to_hash, "/bin/bash"
end
...
```

```
$ sudo ./mruby/bin/mruby mycontainer.rb
root@seckun-1:/# date -s '1970-01-01'
date: cannot set date: Operation not permitted
Thu Jan 1 00:00:00 UTC 1970
root@seckun-1:/# date
Sun Dec 20 07:26:01 UTC 2020
```

- man を見ながら他のCapabilityを調べ、変更して、操作してみよう
- なお、Capability自体を操作する権限が存在する (CAP_SETFCAP/CAP_SETPCAP) ので、安全を考える場合当然まずそれらのCapabilityを落としてからexecveをする。

発展課題: 難しければスキップ推奨です

- FreeBSDのCapsicumについて調べて比較しよう

- <https://www.freebsd.org/cgi/man.cgi?capsicum>

seccomp

ref: <https://mmi.hatenablog.com/entry/2016/08/01/044000>

seccomp (Secure Computingの略らしい)は、Linuxにおいてサンドボックスを実現するためにプロセスのシステムコールの発行を制限する機能です。

- 内部的にはeBPFを用いる
 - システムコール呼び出しをフィルタし、特定の条件で特定の挙動をさせる。
- できることは？
 - allowlist、denylist、errnoの強制指定、特定syscallのトレース、audit logへの記録、ユーザランドプログラムへの判断の委譲[new!]など。
- mruby-seccompで簡単に試すことができる: <https://github.com/haconiwa/mruby-seccomp>

```
$ git clone https://github.com/haconiwa/mruby-seccomp.git
$ cd mruby-seccomp && rake
$ ./mruby/bin/mruby examples/bash_witout_mkdir.rb
==== It will be jailed. Please try to mkdir
$ mkdir /tmp/xxx
Bad system call (core dumped)
$ exit
[2451, #<Process::Status: pid=2451,exited(159)>]
==== It will not be jailed
$ mkdir /tmp/xxx
$ ls -l /tmp/xxx
total 0
$ exit
```

- ここまでの自作コンテナに組み込むなら:

```
...
# Capabilityの直前がいいかも
context = Seccomp.new(default: :allow) do |rule|
  rule.kill(:mkdir, Seccomp::ARG(:>=, 0), Seccomp::ARG(:>=, 0))
end
context.load
# execve後に有効になる。

# Capability
Capability.drop_bound(Capability::CAP_SYS_TIME)

Exec.execve ENV.to_hash, "/bin/bash"
end
...
```

```
$ sudo ./mruby/bin/mruby mycontainer.rb
root@seckun-1:/# mkdir ./foo
Bad system call (core dumped)
```

- UNIXユーザ権限やCapabilityとはまた別の軸であることに注目すること。

cf. Linux Security Module (LSM)

項目自体が発展課題: 難しければスキップ推奨です。

- AppArmor

ref: <https://container-security.dev/lsm/apparmor.html>

AppArmor は Linux Security Module (LSM) の一つで、Mandatory access control (MAC) を実現しています。

アプリケーションごとにプロファイルを適用することができ、特定のファイルへのアクセスやシステムコールの呼び出しの制限を行うことができます。

- AppArmorは 1)バイナリのpath単位で 2)プロセス単位で プロファイルを設定することができる。バイナリ単位はContainer Security Bookの例を参照。
- プロファイルを適用する例を示す。以下で作成。

```
$ cat <<PROF | sudo tee /etc/apparmor.d/nohosts
#include <tunables/global>

profile nohosts {
    #include <abstractions/base>

    /etc/hosts r,
    /usr/bin/cat ix,
}
PROF
$ sudo apparmor_parser -r -W /etc/apparmor.d/nohosts
```

- aa-exec でプロファイル有効のbashを起動すると、何もできないけど...
/etc/hosts だけはcatコマンド経由でのみ閲覧できる。

```
$ sudo aa-exec -p nohosts bash -l
bash: /etc/profile: Permission denied
bash: /root/.profile: Permission denied
bash-5.0#
bash-5.0# cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      ubuntu2004.localdomain

# The following lines are desirable for IPv6 capable hosts
::1           localhost ip6-localhost ip6-loopback
```

```
ff02::1 ip6-allnodes  
ff02::2 ip6-allrouters
```

```
127.0.0.1 ubuntu2004.localdomain
```

```
bash-5.0# cat /etc/passwd  
cat: /etc/passwd: Permission denied  
bash-5.0# id  
bash: /usr/bin/id: Permission denied
```

- mrubyではmruby-apparmorが定義する AppArmor.change_onexec で同等の操作ができる。コンテナの組み込みは、どちらかというとな当なプロファイルの定義が大変ということで省略。
 - <https://udzura.hatenablog.jp/entry/2016/12/09/155055>
 - <https://github.com/haconiwa/mruby-apparmor>
- LSMの他の選択肢として、SELinuxやTOMOYO Linuxが考えられるだろう。将来はKRSIのようなものと組み合わせるのもいいのかも。
 - <https://lwn.net/Articles/808048/>

今回のまとめ

- 通常のプロセスをいかにして独立させ、コンテナとして扱うかについて確かめた。
 - 前の講義の通り、可視範囲の制限、リソースの制限、権限の制限という3軸がある。
- あくまでもLinuxにおけるこの種のコンテナは **通常のプロセスの延長** である。
 - メリットもあり（起動の速さ、集約性）、デメリットもある（ホストの脆弱性に対し弱い、設定ミスなど）。

 seckun-02-lasttask.md

グループ課題

- 希望する課題を選び、課題の大番号ごとにグループを作成してください。
 - 事前に希望がある方はスプレッドシート「演習振り分けシート（1/10）」にお名前を入れてください
(https://docs.google.com/spreadsheets/d/1KxwDcHwoNebKUiO9FTQDFF1eAq_0leoLUPvckxabofk/edit#gid=1999206161)
- グループでディスカッションをしたり、画面共有などを駆使しながら協力して回答を考えてください。

- 今回の授業の最後に（できたところまでで構いませんので）回答例を発表していただきます。

課題一覧

- 1) Linux namespace の動作を確認する最小のコマンド例またはmrubyスクリプト実装と、その出力を考えよ。
 - 「どうやってnamespaceが分離されたプロセスを作るか」「どういう出力で分離されていることが確認できるか」を考えること。
 - 8つ全てでなくとも良いが、1.1) UTS Namespace 2.2) Net namespace の2つから取り掛かると良いかもしれない。
 - b. また、Ubuntu 20.04のディストロをベースにTime Namespaceの動作確認をするにはどうすればいいか考えよ。
- 2) cgroup による制限が働いているのを、コマンドまたはmrubyスクリプトで確認する手順を考え、示せ。
 - 「どうやってcgroupが制限されたプロセスを作るか」「どういう出力で制限されていることが確認できるか」を考えること。
 - 1.1) CPU、1.2) Memory、1.3) pidsの3つのコントローラそれぞれについて示せると良い。
 - 手順はv1/v2 どちらでも構わない。
 - b. それぞれのコントローラがどのような攻撃を防げるか例示せよ。
- 3) chrootを試せ。
 - a. chrootはどういう条件を満たせば「脱獄」ができるか検証し、PoCを実装せよ。
 - b. また、pivot_root は chroot とどう違うか。Linux Namespaceの関連と合わせて答えよ
- 4) capabilityについて調査せよ。その上で以下に答えよ。
 - a. ping のfile capabilityにより、なぜ一般ユーザがICMP Socketを作れるのか説明せよ。
 - net.ipv4.ping_group_range を考慮する必要がある（Ubuntu 20.04など）が、一旦この設定は net.ipv4.ping_group_range = 1 0 として検証せよ
 - b. デフォルト設定のDockerコンテナで、rootユーザが date -s を実行できない理由は何か、背景と技術的な点を説明せよ。
 - c. mruby を用いて 2) と同等の制限を課したシェルを起動せよ。
- 5) seccomp の機能について以下の実験を行え。C言語のほか、Haconiwaの生成するmrubyを利用しても良い。
 - a. ホワイトリスト の設定と動作確認の手順を示せ。
 - b. ブラックリスト の設定と動作確認の手順を示せ。