Mohammad Arshad

U00304676

NCS 531 -11

CHROOT SANDBOX VULNERABILITY LAB

TASK:

1. **Understanding how chroot works:**

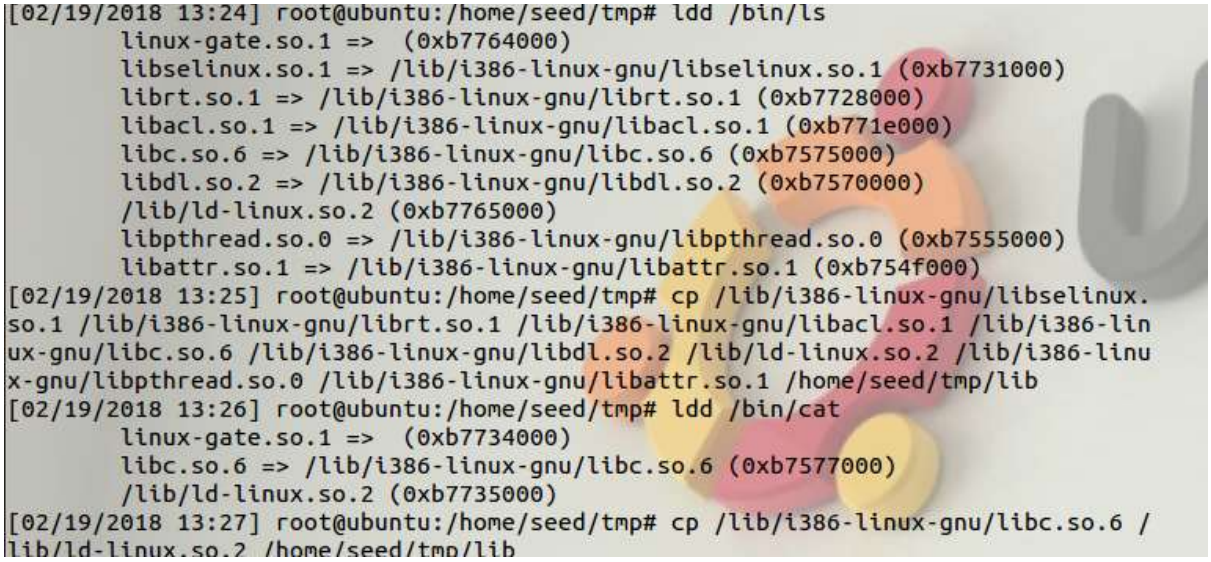
- First I created a folder **/tmp** and inside that created a **/bin** and **/lib** folder.
- Then copied the **/bin/bash** to the **/bin** folder in the newly created /**tmp** folder.
- After that copied the required library files for **/bin/bash** to the **/lib** folder in **/tmp**.

```
[02/19/2018 10:17] root@ubuntu:/home/seed# mkdir tmp
[02/19/2018 10:18] root@ubuntu:/home/seed# mkdir -p /home/seed/tmp/bin
[02/19/2018 10:18] root@ubuntu:/home/seed# cp -v /bin/bash /home/seed/tmp/bin
`/bin/bash' -> `/home/seed/tmp/bin/bash'
[02/19/2018 10:19] root@ubuntu:/home/seed# ldd /bin/bash
        linux-gate.so.1 =>  (0xb77a7000)
        libtinfo.so.5 => /lib/i386-linux-gnu/libtinfo.so.5 (0xb7774000)
        libdl.so.2 => /lib/i386-linux-gnu/libdl.so.2 (0xb776f000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb75c5000)
        /lib/ld-linux.so.2 (0xb77a8000)
[02/19/2018 10:20] root@ubuntu:/home/seed# cp ^C
[02/19/2018 10:23] root@ubuntu:/home/seed# cp /lib/i386-linux-gnu/libtinfo.so.5
/lib/i386-linux-gnu/libdl.so.2 /lib/i386-linux-gnu/libc.so.6 /lib/ld-linux.so.2
/home/seed/tmp/bin
```

- Now made the **/tmp** folder a root jail directory by using **chroot** command.
- Tried different commands inside the jail.

```
[02/19/2018 12:40] root@ubuntu:/home/seed/tmp# chroot /home/seed/tmp /bin/bash
bash-4.2# ls
bash: ls: command not found
bash-4.2# mkdir tst
bash: mkdir: command not found
bash-4.2# pwd
/
bash-4.2# bash --help
GNU bash, version 4.2.25(1)-release-(i686-pc-linux-gnu)
Usage:  bash [GNU long option] [option] ...
        bash [GNU long option] [option] script-file ...
GNU long options:
        --debug
        --debugger
        --dump-po-strings
        --dump-strings
        --help
        --init-file
        --login
        --noediting
        --noprofile
```

- Inorder to perform the upcoming tasks, I copied **/bin/ls and /bin/cat** to **/tmp** and linked the required libraries.



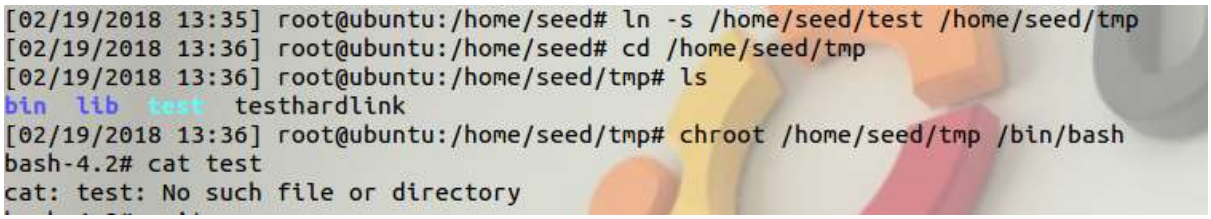
Symbolic link:

Q. If there is a symbolic link under /tmp, and this symbolic link points to a file outside of /tmp; can one follow this symbolic link to get out of the /tmp jail?

Ans: No, Symlinks are essentially just pointers to another file, you can't point to something outside the chroot because it is looking for a file with that name, which is not inside the chroot.

Inorder to prove this, I created an experiment.

- Here I created a file **test** and provided a **symbolic link** to the **/tmp** jail directory using l**n –s** command.
- Then inside the chroot **/tmp** jail I tried to open the symbolic link of test file, but it was showing as no such file or directory.



Hard link:

Q. What if the link is a hard link, rather than a symbolic link?

Ans: Yes, Hard link will work inside the chroot, this is because hardlinks are pointers to the inode.

To prove this, I created an experiment.

- Here I created a file **testhlink** and provided a **hard link** to the **/tmp** jail directory using l**n** command.
- Then inside the chroot **/tmp** jail I tried to open the hard link of testhfile, and was able to open it.

```
[02/19/2018 13:37] root@ubuntu:/home/seed# ln /home/seed/testhlink /home/seed/tmp

[02/19/2018 13:37] root@ubuntu:/home/seed# cd /home/seed/tmp
[02/19/2018 13:38] root@ubuntu:/home/seed/tmp# chroot /home/seed/tmp /bin/bash
bash-4.2# cat testhlink
this is a test file for hardlink
bash-4.2#
```

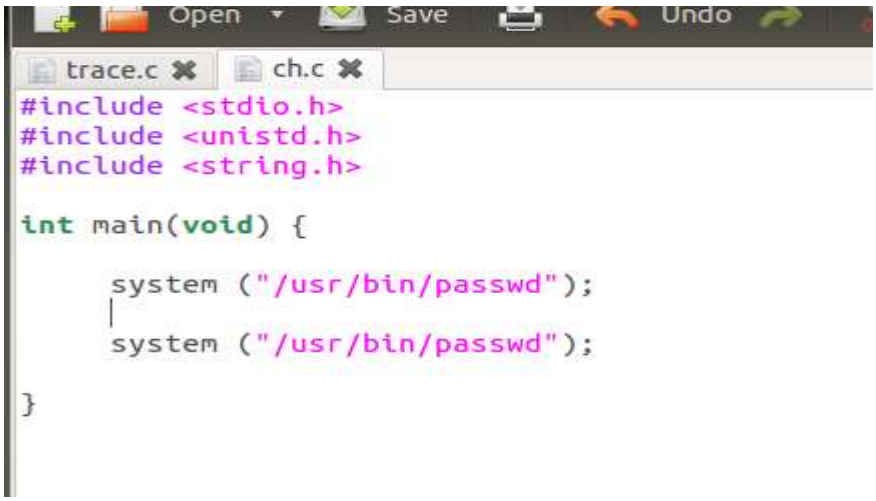
File descriptors:

Q. Before entering the /tmp jail, a super-user (or set-root-uid) process has already opened a file /etc/shadow. Can this process still be able to access this file after entering the jail?

Ans: No, The process won't be able to access this file after entering the jail.

To prove this I created a experiment.

- I created a small program that will call **/usr/bin/passwd** and called it twice. This will write the new password into the **/etc/shadow** file.



- Here we can see that the **/usr/bin/passwd** executed twice.

```
[02/20/2018 22:23] root@ubuntu:/home/seed# gcc -o ch ch.c
[02/20/2018 22:24] root@ubuntu:/home/seed# ch
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Enter new UNIX password:
Retype new UNIX password:
Sorry, passwords do not match
passwd: Authentication token manipulation error
passwd: password unchanged
[02/20/2018 22:24] root@ubuntu:/home/seed#
```

- Now I changed the program such that the second system call for **/urs/bin/passwd** will be after entering the chroot **/tmp** jail.

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main(void) {

      system ("/usr/bin/passwd");
      /* chroot */
      chdir("/home/seed/tmp");
      if (chroot("/home/seed/tmp") != 0) {
          perror("chroot /home/seed/tmp");
          return 1;
      }
      system ("/usr/bin/passwd");

}
```

- Here we can see that the second call was not executed as inside the jail /tmp we don't have the /etc/shadow.

```
[02/20/2018 22:23] root@ubuntu:/home/seed# gcc -o ch ch.c
[02/20/2018 22:23] root@ubuntu:/home/seed# ch
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
[02/20/2018 22:23] root@ubuntu:/home/seed#
```

Comparing the chroot command and the chroot() system call:

Q. there are two ways to run aprogram in a jail. One ways is to use the chroot command; the other is to modify the program to call chroot() system call directly. What are the difference between this two methods?

Ans. **chroot()** changes the root directory of the calling process to that specified in path. This directory will be used for pathnames beginning with /. The root directory is inherited by all children of the calling process.

chroot - run command or interactive shell with special root directory. Run command with root directory set to NEWROOT.

To show the working of **chroot(),** i created an experiment.

- Here the I wrote a program that will chroot in to **/tmp** and then open the **chrttest** which is inside **/tmp**.

*test ✖ testhlink ✖ chroot.c ✖ passwd ✖ chrttest ✖

```
#include <stdio.h>
#include <unistd.h>

int main(void) {
     FILE *f;

     /* chroot */
     chdir("/home/seed/tmp");
     if (chroot("/home/seed/tmp") != 0) {
          perror("chroot /home/seed/tmp");
          return 1;
     }

     /* do something after chrooting */
     f = fopen("chrttest", "r");
     if (f == NULL) {
          perror("chrttest");
          return 1;
     } else {
          char buf[100];
          while (fgets(buf, sizeof(buf), f)) {
               printf ("%s", buf);
          }
     }

}
```

- Here we can see the output of the program. It entered the **/tmp** jail and opened the file.

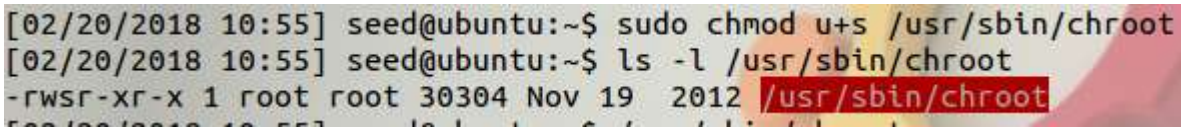
```
[02/19/2018 20:32] root@ubuntu:/home/seed# gcc -o chrt chroot.c
[02/19/2018 20:32] root@ubuntu:/home/seed# ./chrt
This is a test file for chroot().
[02/19/2018 20:32] root@ubuntu:/home/seed#
```

2. Abusing unconstrained chroot:

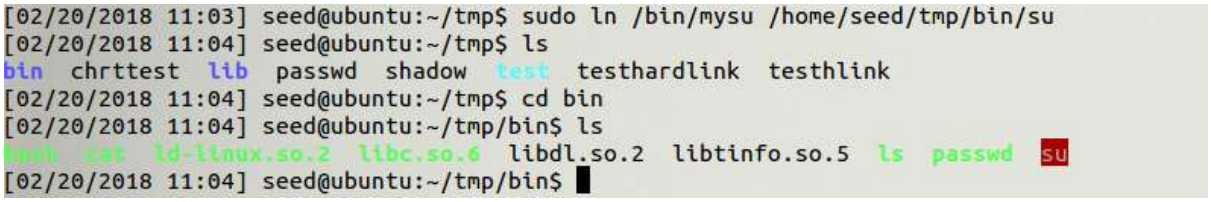
(a) Can you run a set-root-uid program inside a jail?

Ans. Yes, Given below experiment will prove it.

- Made the chroot a set uid program.

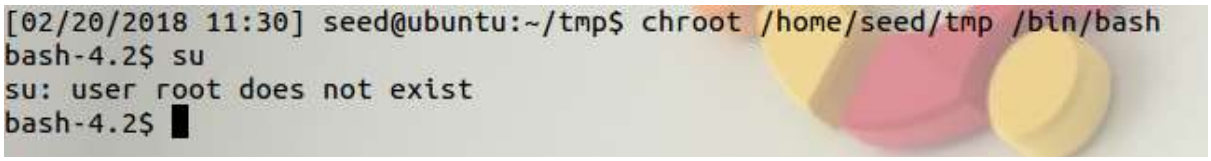


- Copied the **/bin/mysu** to **/tmp** jail folder.



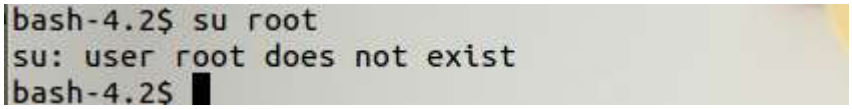
- Linked the required libraries for **/bin/mysu.**

- Entered the chroot /tmp jail and tried to run the set uid program SU. Got the output su: user root does not exist. **This shows that the set uid program was able to run inside the jail.**



(b). can you use /tmp/su to become root?

Ans. No. Below output show that user root does not exist. This means **SU needs the file /etc/passwd, from where it fetches the username.**



Some other files which **SU** needs are

/usr/bin/su

/etc/environment

/etc/group

/etc/passwd

/etc/shadow

/etc/security/user

/etc/security/environ

/etc/security/limits

/etc/security/passwd

/var/adm/sulog

/etc/security/enc/LabelEncodings

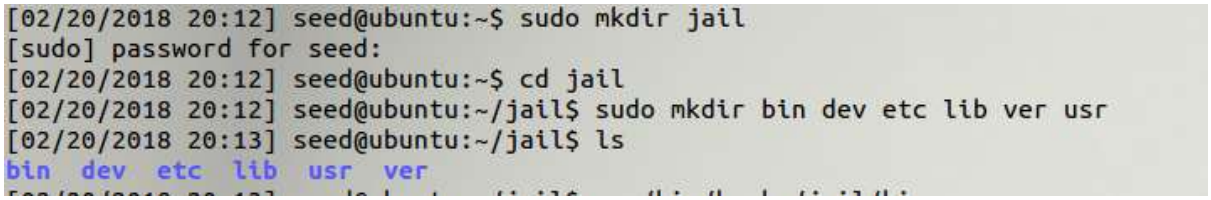
(c). Can you regain the root privileges after you get out of the jail?

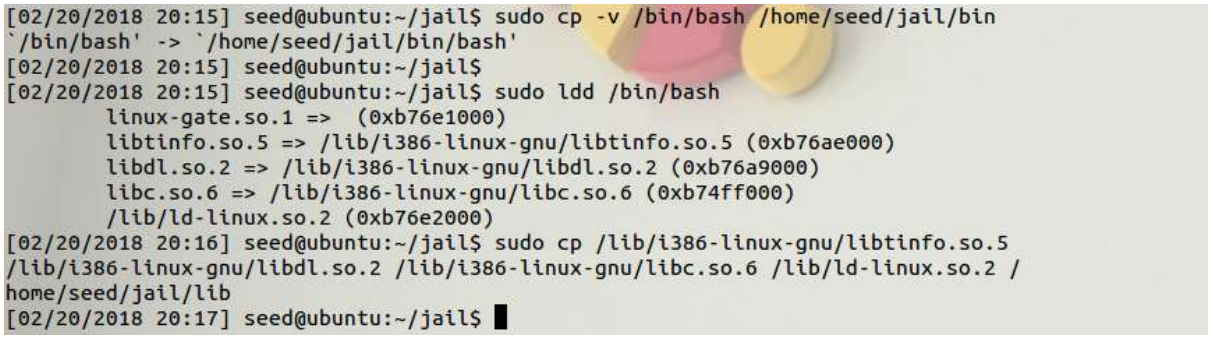
Ans. Yes, We can regain the root privileges after we get out of the jail. This can be done by changing the UID of root from 0 to any other UID and provide normal user the UID 0.

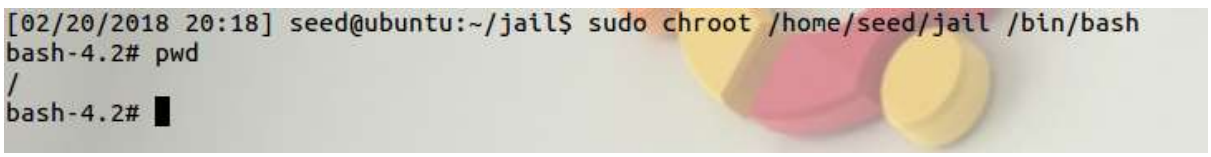
3. **Breaking out of a chroot jail:**

a.

- Created new directory jail and made it a chroot jail.

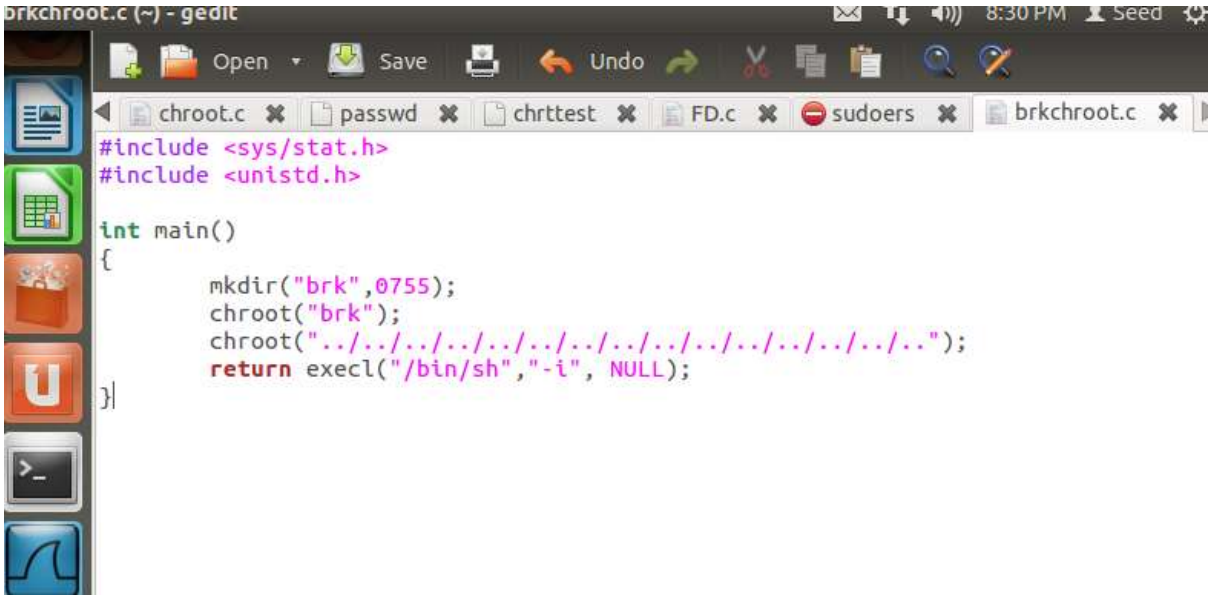






- Created a program to break the chroot jail:

1. create a temporary folder **"brk"** and we chroot to that, this way we make sure our current working directory is outside the fake root, and we can do so because we're root.
2. then we chroot to parent folders all the way up to the root using chroot(../../../../..).
3. execute file within the shell.



- Compiled the program.

- Entered the jail and tried to run ls command, which gave error command not found. Now executed the program. This gave as access to root and was able to run all the root commands.

```
# cat > test
this is a testfile
^C
#
# ls
bin  brk  brkchroot  brkchroot.c  dev  etc  lib  test  usr  ver
# rm test
# ls
bin  brk  brkchroot  brkchroot.c  dev  etc  lib  usr  ver
#
```

(b) Killing processes: demonstrate how attackers can kill other processes from within a prison.

- Opened a new terminal a started top process.

```
Terminal
[02/20/2018 21:02] seed@ubuntu:~$ top

top - 21:05:25 up 18:57,  3 users,  load average: 0.23, 0.27, 0.25
Tasks: 196 total,   2 running, 192 sleeping,   2 stopped,   0 zombie
Cpu(s): 20.5%us,  5.6%sy,  0.0%ni, 73.3%id,  0.0%wa,  0.0%hi,  0.7%si,  0.0%st
Mem:   2064788k total,  1862976k used,   201812k free,   251500k buffers
Swap:  2094076k total,     5660k used,  2088416k free,   923004k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1007 root      20   0  111m  79m  12m S 17.0  4.0  23:11.37 Xorg
29024 seed      20   0  541m 111m  36m S  6.4  5.5  22:55.32 firefox
11727 seed      20   0  2852 1188  876 R  1.3  0.1   0:01.98 top
11743 seed      20   0 39448 8000 6432 S  1.0  0.4   0:00.18 gnome-screensho
23740 root      20   0 11400 5664 3988 S  1.0  0.3  12:29.67 vmtoolsd
 3453 seed      20   0  109m  25m  12m S  0.6  1.3   2:02.76 gnome-terminal
23734 seed      20   0 67572  15m  12m S  0.6  0.8  11:10.54 vmtoolsd
 2411 root      20   0 38952 9852 5488 S  0.3  0.5   0:21.26 apache2
 2808 seed      20   0  145m  13m  10m S  0.3  0.6   1:08.77 metacity
 3178 seed      20   0 73832 9780 7296 S  0.3  0.5   0:10.63 unity-applicati
10678 root      20   0     0    0    0 S  0.3  0.0   0:02.39 flush-8:0
11652 root      20   0     0    0    0 S  0.3  0.0   0:00.79 kworker/0:2
11732 root      20   0     0    0    0 S  0.3  0.0   0:00.17 kworker/0:0
17322 snort     30  10  333m  81m 4352 S  0.3  4.1  33:22.04 snort
    1 root      20   0  3664 1928 1288 S  0.0  0.1   0:07.51 init
```

- Using the previously created chroot jail break program, gained access to root.
- Now using the kill command. Stopped the top process running outside the jail.

```
[02/20/2018 21:04] seed@ubuntu:~/jail$ sudo chroot /home/seed/jail /bin/bash
bash-4.2# ./brkchroot
# pgrep top
11727
# pgrep top
11727
# kill 11727
#
```

(c) (20 bonus points) Controlling processes: demonstrate how attackers can use ptrace() to control processes that are outside of the prison?

The **ptrace()** system call provides a means by which one process (the "tracer") may observe and control the execution of another process and and examine and change the tracee's memory and registers.

- Created a program that will trace the /bin/ls process.

```
Open  Save  Undo
trace.c
#include <sys/ptrace.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>

 int main()
{   pid_t child;
    long prc;
    child = fork();
    if(child == 0) {
        ptrace(PTRACE_TRACEME, 0, NULL, NULL);
        execl("/bin/ls", "ls", NULL);
    }
    else {
        wait(NULL);
        prc = ptrace(PTRACE_PEEKUSER,
                     child, 4 * prc,
                     NULL);
        printf("The child made a "
               "system call %ld\n", prc);
        ptrace(PTRACE_CONT, child, NULL, NULL);
    }
    return 0;
}
```

- Created a test file named **testfortrace.**
- Entered the chroot jail.
- Run the program and just got the output with process number. We didn't get the entire process because we didn't had root access.
- After that we run the chroot jail break program first, using this we will get access to root. Now we will run the ./trace program.

```
[02/20/2018 21:37] seed@ubuntu:~/jail$ cat > testfortrace
this is a file for ptrace test
^C
[02/20/2018 21:37] seed@ubuntu:~/jail$ sudo chroot /home/seed/jail /bin/bash
bash-4.2# ./trace
The child made a system call -1
bash-4.2# ./brkchroot
# ./trace
The child made a system call -1
# bin  brkchroot          dev  lib           trace    trace.c~  ver
brk  brkchroot.c  etc  testfortrace  trace.c  usr

# ./trace
The child made a system call -1
# bin  brkchroot          dev  lib           trace    trace.c~  ver
brk  brkchroot.c  etc  testfortrace  trace.c  usr
rm testfortrace
#
# ls
bin  brkchroot    dev  lib    trace.c   usr
brk  brkchroot.c  etc  trace  trace.c~  ver
#
```

- From the output we can see the running process number and the process.
- Even we can change the process memories as show above, by deleting the test file within the execution the program.

4. **Securing chroot: Discuss how you can solve the above problems with chroot.**

- Ensure software within the jail is patched as well as the host OS.
- Avoid setuid executables.
- Ensure root (UID 0) does not even exist within the jail.
- Install the minimum possible within the jail --- offer an intruder nothing to work with.
- Use **chattr -i** to help prevent the creation of new files (or nodes).
- Do not run anything in the jail as root
- Set permissions as low as possible
- Set owner of all files to root when possible