Chroot Sandbox Vulnerability Lab

1 Lab Overview

The learning objective of this lab is for students to substantiate an essential security engineering principle, the *compartmentalization* principle, by studying and evaluating the `chroot` mechanism implemented in `Unix` operating systems. The basic idea of compartmentalization is to minimize the amount of damage that can be done to a system by breaking up the system into as few units as possible while still isolating code that has security privileges. This same principle explains why submarines are built with many different chambers, each separately sealed. This principle is also illustrated by the *Sandbox* mechanism in computer systems.

Sandbox can provide a restricted environment for us to run programs that are not completely trustworthy. For example, if the program is downloaded from an untrusted source, running the program in an unrestricted environment can expose the system to potential risks. If these programs can be executed in a restricted environment, even if the programs behave maliciously (the programs might contain malicious contents or they might be compromised by attackers during the execution), their damage is confined within the restricted environment. Almost all the `Unix` systems have a simple built-in sandbox mechanism, called `chroot`.

In this lab, students need to figure out how `chroot` works, why it works, and why it should only be used by root. Moreover, students will see the vulnerabilities of this type of sandbox.

2 Lab Tasks

The `chroot` command in `Unix` redefines the meaning of the *root* directory. We can use this command to change the root directory of the current process to any directory. For example, if we `chroot` to `/tmp` in a process, the root ("/") in the current process becomes `/tmp`. If the process tries to access a file named `/etc/xyz`, it will in fact access the file `/tmp/etc/xyz`. The meaning of root is inheritable; namely, all the children of the current process will have the same root as the parent process. Using `chroot`, we can confine a program to a specific directory, so any damage a process can cause is confined to that directory. In other words, `chroot` creates an environment in which the actions of an untrusted process are restricted, and such restriction protects the system from untrusted programs.

A process can call `chroot()` system call to set its root directory to a specified directory. For security reasons, `chroot()` can only be called by the super-user; otherwise, normal users can gain the super-user privilege if they can call `chroot()`. A command called `chroot` is also implemented in most `Unix` systems. If we run `"chroot newroot prog"`, the system will run the `prog` using `newroot` as its root directory. For the same reason, the `chroot` command can only be executed by the super-user (i.e., the effective user id has to be super-user).

The following is what you are expected to do in this lab:

1. **Understanding how `chroot` works:** Assume that we use `/tmp` as the root of a jail. Please develop experiment to answer the following questions:

(a) *Symbolic link:* 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?

(b) *Hard link:* what if the link is a hard link, rather that a symbolic link?

(c) *File descriptors*: 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?

(d) *Comparing the* `chroot` *command and the* `chroot()` *system call*: there are two ways to run a program 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?

Note: Once you are inside a jail, you cannot see any file outside of the jail. Therefore, you need to copy a number of commands and libraries into the jail first; otherwise, there is not much you can do.

```
% mkdir /tmp/bin
% cp /bin/ls /tmp/bin
% cp /bin/bash /tmp/bin
% cp -r /lib /tmp
```

2. **Abusing unconstrained** `chroot`**:** Assume that the root wants to allow normal users to use the `chroot` command. The root can do this by turning the `chroot` command into a `Set-UID` program. Please implement an attack to demonstrate how a normal can gain the root privilege using this unconstrained `chroot`.

   (a) Can you run a set-root-uid program inside a jail? To run a set-root-uid program, we need to make the program available within the prison (i.e., the `/tmp` directory). Unfortunately, copying a root-owned `Set-UID` program by a normal user does not preserve the root privilege of the program. For these programs, you need to use hard links. For example, we can run "`ln /bin/su /tmp/su`" to make a hard link to the `/bin/su` program from `/tmp`. Hard links preserve the ownership and the `Set-UID` property of the file.

   Important Note: The `/bin/su` command in our provided `Ubuntu` VM image does not work from the prison (we still have not figured out why). We use another version of `su`. The `su` program can be found in the package called `coreutils`. We have linked this package (`coreutils-7.6`) in the web page. Please download that, compile it, and copy it to the `/bin` directory. Please follow the following commands:

```
% tar xzvf coreutils-7.6.tar.gz
% cd coreutils-7.6
% ./configure
% make                        # compile all
% cd src                      # su is in this directory
% sudo cp ./su /bin/mysu      # we rename it to mysu
% sudo chown root /bin/mysu   # make root the owner
% sudo chmod 4755 /bin/mysu   # turn on the setuid bit
```

After running the above command, you will have a new version of `su` called `mysu` in the `/bin` directory. To be able to access this program from the prison, you need to first make a hard link from the `/tmp` directory:

```
% ln /bin/mysu /tmp/su
```

(b) Now, can you use `/tmp/su` to become root? Think about how the `su` program authenticate users (hint: what files does `su` use when authenticating users?)

(c) Having a root shell inside a jail can only do limited damage. It is difficult, if possible, to apply the root privileges on objects that are outside of the jail. To achieve a greater damage, you would like to maintain the root privilege after you get out of that jail. Unfortunately, to get out, the process running within the jail has to exit first, and the root privileges of that process will be lost. Can you regain the root privileges after you get out of the jail? You might have to do something within the jail before you let go the root privileges.

3. **Breaking out of a `chroot` jail:** Some server programs are usually executed with root privileges. To contain the damage in case the server programs are compromised, these programs are put in a sandbox, such as the `chroot` jail. Assume that an attacker has already compromised a server program, and can cause the server program to run (with root privilege) any arbitrary code. Can the attacker damage anything outside of the sandbox? Please demonstrate your attacks. You do not need to demonstrate how you compromise a server program. Just emulate that by writing a program with embedded malicious code, and then run this program as a root in the `chroot` jail. Then demonstrate the damage that you can achieve with this malicious code. You can put anything you want in the malicious code. You should try your attacks on `Linux`. If you have an access to `Minix`, please also try your attacks on `Minix` (whether attacks on `Minix` are required is at the discretion of your instructor) .

(a) Using "`cd ..`" to get out of the jail (your malicious code should still maintain the root privilege after getting out).

- Hint 1: Remember how `Linux` prevents a process from using `cd ..` to get out of prison. If a process is at the root of a prison `/tmp`, directly using `cd ..` will not go to the real root, because the system knows that the current directory is the root of the prison, so it will not go beyond that. In the operating system, the process data structure has an entry that records the i-node of the root directory (e.g. in `Minix`, this entry is named `fp_rootdir`, we will use `fp_rootdir` to refer to this root entry in the rest of the discussion). Each process has its own root directory. If a process is running in a prison `/tmp`, the root entry will be the i-node of the directory `/tmp`.
- Hint 2: Remember that if your current directory is not the same as `fp_rootdir`, you can always conduct `cd ..`. However, you do want to do `cd ..` at the root (the prison's root) directory to get out of the prison. The question is whether you can create a scenario where the following three conditions are all true simultaneously: (1) your current directory is `/tmp`, (2) your prison is rooted at `/tmp`, but (3) `fp_rootdir` is not `/tmp`. If you remember how and when `fp_rootdir` is updated, you might be able to create the above scenario. Note `chdir()` and fchdir() calls might be useful.
- Hint 3: You may want to get familiar with the `chdir(char *)`, `chroot(char *)`, and `getcwd(char *buf, size_t size)` system calls and functions. The `chdir()` call is used to change directory (it is used to implement the `cd` command). The `chroot()` call changes the root directory to that specified in path. It should be noted that the `chroot()`

call does not change the current working directory, so that '.' can be outside the tree rooted at '/'. The `getcwd()` call gets the current working directory. The following code snippet shows examples of these calls.

```
char buffer[100];
if(chdir("/tmp") != 0)
    printf("change directory failure \n");
if(chroot("/tmp") != 0)
    printf("chroot failure \n");

getcwd (buffer, 50);
printf("Current Directory: %s\n", buffer);
```

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

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

4. Securing `chroot`: Discuss how you can solve the above problems with `chroot`. Implementation is not required.

Submission

You need to submit a detailed lab report to describe what you have done and what you have observed; you also need to provide explanation to the observations that are interesting or surprising.