Module: Sandboxing

chroot

Yan Shoshitaishvili
Arizona State University

Simple Sandbox Example: Filesystem Isolation

Traditional sandbox: chroot jail.

- `chroot()` first appeared in Unix in 1979, BSD shortly afterwards.
- changes the meaning of "/" for a process (and its children)
- `chroot("/tmp/jail")` will disallow processes from getting out of the jail
- used to be the de-facto sandboxing utility
- no syscall filtering or other isolation

Effects of chroot

`chroot("/tmp/jail")` has two effects:

- For this process, change the meaning of "/" to mean "/tmp/jail".
  - and everything under that: "/flag" becomes "/tmp/jail/flag"
- For this process, change "/tmp/jail/.." to go to "/tmp/jail"

`chroot("/tmp/jail")` does NOT:

- Close resources that reside outside of the jail.
- `cd` (`chdir()`) into the jail.
- Do anything else!

This has a number of implications...

chroot pitfalls: previously open resources

Neither of the effects of `chroot()` do anything to previously-open resources.

How is this useful?

Similar to open and execve, Linux has open**at** and execve**at**:

```
int open(char *pathname, int flags);
int openat(int dirfd, char *pathname, int flags);
int execve(char *pathname, char **argv, char **envp);
int execveat(int dirfd, char *pathname, char **argv, char **envp, int flags);
```

`dirfd` can be a file descriptor representing any open()ed directory, or the special value `AT_FDCWD` (note: chroot() does not change the current working directory)!

Many other system calls also have "at" variants!

chroot pitfalls: forgetfulness

The kernel has no memory of previous chroots for a process!

What happens if you chroot again?

How might that be used to escape?

Is chroot safe?

Generally, a user with an effective ID of 0 (i.e., a process run as root or SUIDed to root) can *always* break out of a chroot, unless the chroot syscall is blocked!

Also missing other forms of isolation:

- PID
- network
- IPC

While chroot used to be popular to isolate services on a machine, it has fallen out of favor. Replacements:

- cgroups
- namespaces
- seccomp

Useful Reference

Sets the kernel's concept of the root directory of your process (and does NOTHING else):

```
    chroot(“/new/root/directory”)
```

Sets the kernel's concept of the current working directory of your process:

```
    chdir(“/new/working/directory”)
```

Opens a file relative to the current working directory:

```
    open(“../../file”, O_RDONLY)
    openat(AT_FDCWD, “../../file”, O_RDONLY);
```