Sandbox and SFI

Chengyu Song

Slides modified from
Dawn Song

Sandbox

- A controlled environment for untrusted applications, by limiting system resources they can access
  - File system
  - IPC
  - CPU and memory
  - Disk and I/O rates
  - Network access
  - Sensors (camera/microphone/GPS/...)

Mechanisms: chroot

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.

Only a privileged process (Linux: one with the CAP_SYS_CHROOT capability) may call chroot().

In the past, chroot() has been used by daemons to restrict themselves prior to passing paths supplied by untrusted users to system calls such as open(2).

chroot limitations

However, if a folder is moved out of the chroot directory, an attacker can exploit that to get out of the chroot directory as well. The easiest way to do that is to chdir(2) to the to-be-moved directory, wait for it to be moved out, then open a path like ../../../etc/passwd.

It is not intended to be used for any kind of security purpose, neither to fully sandbox a process nor to restrict filesystem system calls.

Mechanisms: Jail

- FreeBSD jail, an OS-level virtualization mechanism
  - Each jail is a virtual environment running on the host machine with its own files, processes, user and superuser accounts.
  - Each jail is sealed from the others
  - The limited scope of a jail allows system administrators to delegate several tasks which require superuser access without handing out complete control over the system
- https://www.freebsd.org/cgi/man.cgi?query=jail&format=html

Mechanisms: namespaces

- Linux namespaces (similar to jail): virtualization and isolation
  - Cgroup: Cgroup root directory (resources quota)
  - IPC: System V IPC, POSIX message queues
  - Network: Network devices, stacks, ports, etc.
  - Mount: Mount points
  - PID: Process IDs
  - User: User and group IDs
  - UTS: Hostname and NIS domain name

Mechanisms: more OS-level virtualization mechanisms

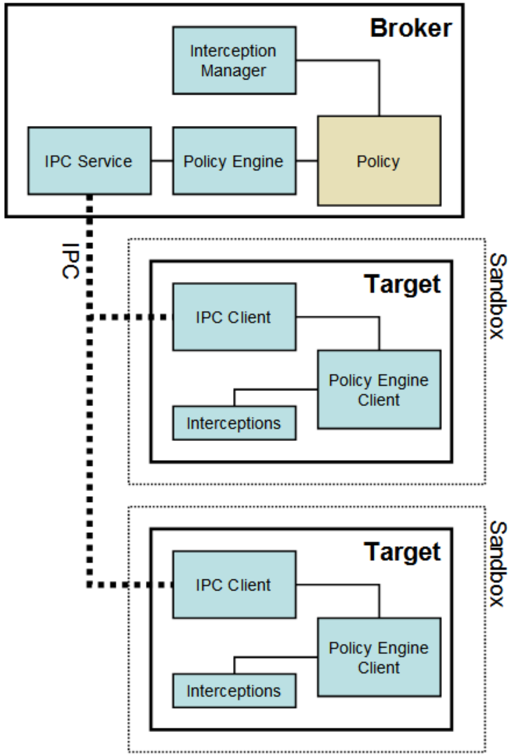
https://en.wikipedia.org/wiki/Operating-system-level_virtualization

- Docker, containers, LXC, etc

Mechanisms: reusing DAC & MAC

- Windows: security tokens, job object, desktop object, integrity level
- Linux: DAC, capabilities, SELinux

Examples: browser sandbox



Examples: iOS sandbox

- Mandatory Access Control Framework (MACF)
- Apple Mobile File Integrity (AMFI)
- Entitlements
- Permission system

Examples: Android sandbox

- UID-based isolation
- SEAndroid
- Binder
- Permission system

Vulnerabilities in reference monitors

- ~~Memory corruption~~
- Incomplete mediation
- Time-of-check-to-time-of-use (TOCTTOU)
- Confused deputy

TOCTTOU

```
/* Process A */                   | /* Process B */
/* Part of a setuid program */    |
if (access("file", W_OK) != 0) {  |
   exit(1);                       |
}                                 |
                                  | /* After the access check */
                                  | symlink("/etc/passwd", "file");
                                  | /* Before the open, "file" */
                                  | /* points to the password */
fd = open("file", O_WRONLY);      |
write(fd, buffer, sizeof          |
   (buffer));                     |
```

TOCTTOU

- In Unix, often occurs with file system calls because system calls are not atomic
- But, TOCTTOU vulnerabilities can arise anywhere there is mutable state shared between two or more entities

Confused deputy

- `(SYSX)FORT` is a fortran compiler, that:
  - Needs to write stats to `(SYSX)STAT`
  - Allows user to provide filename where debugging output is written to at run time
- Problem
  - Billing info is stored in the home directory. So user can provide billing filename to compiler and trash the directory with debugging info.

Confused deputy

- Solutions?
  - Capability delegation
  - ACL: `setuid()`
- Problems?
  - Trust

Software fault isolation (SFI)

- OS-level sandboxes work at process level,
- What about sub-process level components?
  - Browser process: HTML parser, JS engine, etc.
  - Monolithic kernel: file systems, drivers, etc.
- SFI: sandbox inside the process' address space

SFI mechanisms

- How can a compromised/malicious module attack others?

SFI mechanisms

- How can a compromised/malicious module attack others?
  - Read, write, invoke

SFI mechanisms

- How can you confine a module's capability to read, write, invoke?
  - Hardware features?
  - Software-based approach: check/mask the target address
    - inline reference monitor

Native client

https://developer.chrome.com/native-client

API Integrity

- How to do inter-module communication under SFI?
  - Can you pass a pointer?
- How to enforce fine-grained access control over objects?
  - A specification