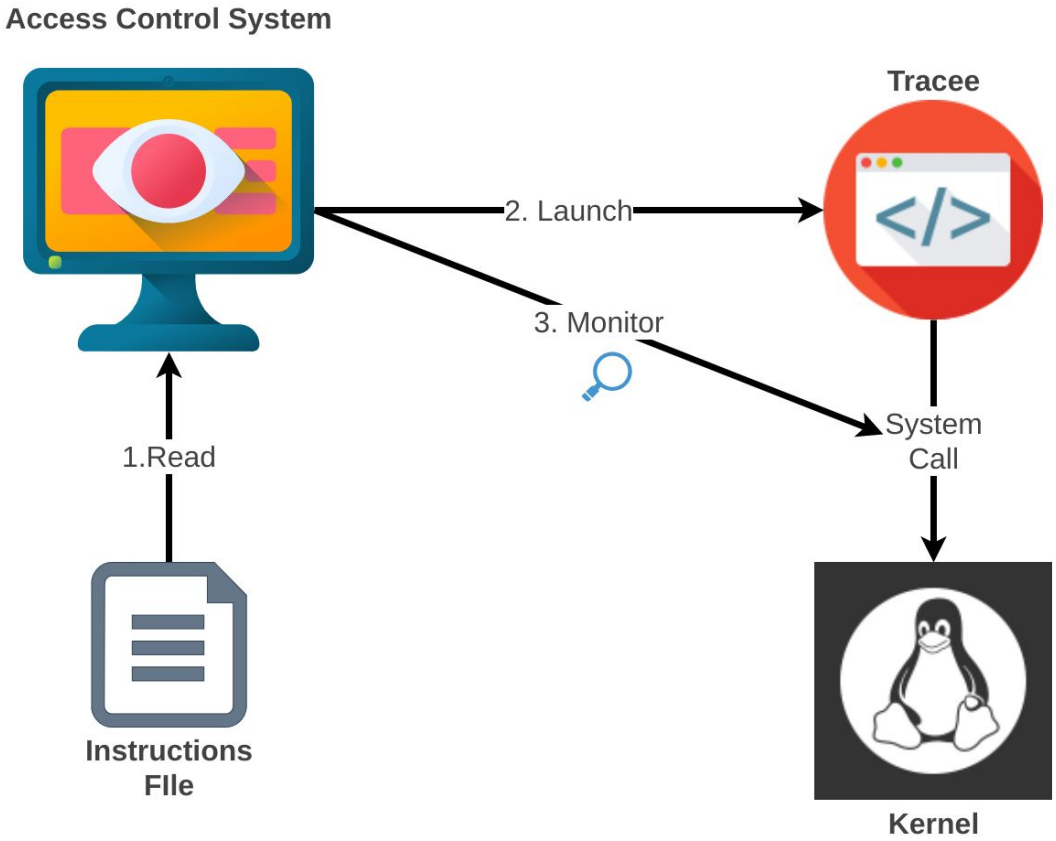
HY-457

Tutorial 2

Assignment 2 / Part 1

- Create an **access control system for system calls**
- Goal:
  - Restrict an executable from invoking a large number of system calls within a short period of time
- Access control mechanism is enabled only after a **sequence** of system calls has been identified

Overview


Access Control System
Tracee
2. Launch
3. Monitor
1.Read
System
Call
Instructions
FIle
Kernel

Instructions File

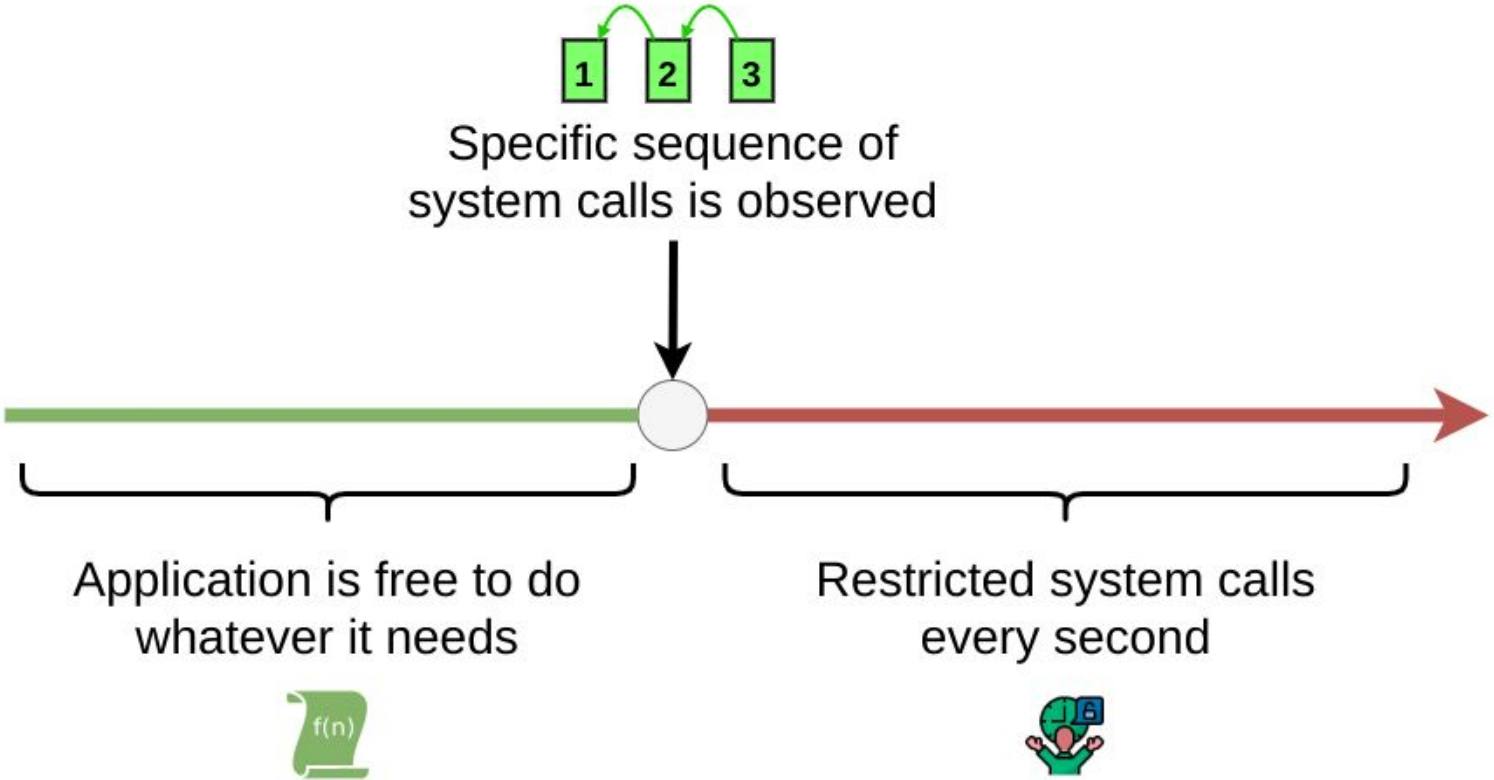
- Given as input to your system

- Contains:
  - 1st line contains sequence that enables the system
  - Rest of the lines contain restrictions on system calls

instructions.txt

```
mprotect mprotect munmap
clone 5
mmap 6
gettid 3
```

Timeline

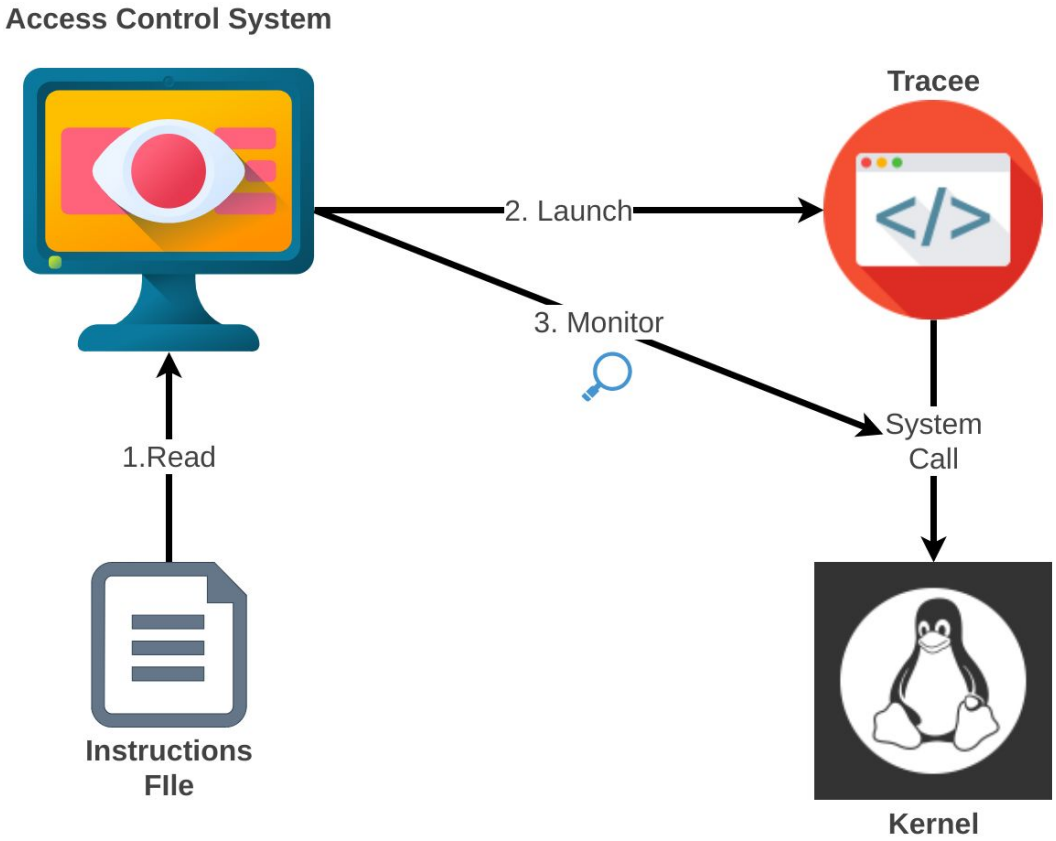

1
2
3
Specific sequence of
system calls is observed
Application is free to do
whatever it needs
f(n)
Restricted system calls
every second

Tracing

- Use `ptrace()` to monitor system calls

```
PTRACE(2)                        Linux Programmer's Manual                        PTRACE(2)

NAME
       ptrace - process trace

SYNOPSIS
       #include <sys/ptrace.h>

       long ptrace(enum __ptrace_request request, pid_t pid,
                   void *addr, void *data);

DESCRIPTION
       The  ptrace() system call provides a means by which one process (the "tracer") may observe
       and control the execution of another process (the "tracee"), and examine  and  change  the
       tracee's memory and registers.  It is primarily used to implement breakpoint debugging and
       system call tracing.
```

Overview


Access Control System
Tracee
2. Launch
3. Monitor
1.Read
System
Call
Instructions
FIle
Kernel

Tracing

- **Warning!**
  The tracee might be stopped at both the entry and the exit from a system call

- Need to carefully count calls

- Free to decide the action when a violation is detected
  - Print a message
  - Wait for a short period
  - ...

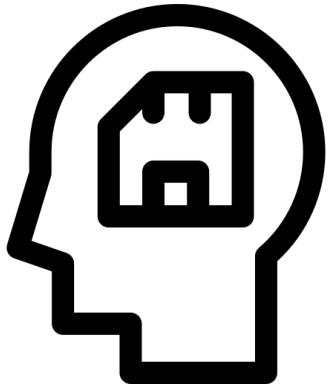
Implementation Details

- Ptrace does not know system call symbolic names
  - Works with system call numbers
- Need to map names to numbers
- **Warning!**
  System call numbers and their symbolic names may be different across architectures
- Your implementation should be compatible with the department's workstations

Implementation Details

You need to store:

1. Monitored system calls
   - Their number
   - The number of times they can be called
   - *(Optional)* Their symbolic name
2. Initialization sequence
3. The last X system calls made by the tracee
4. The time when each system call was made



Assignment 2 / Part 2

- In this assignment you will have to modify the *write* and *close* system calls in order to implement a file monitoring system
- The system keeps track of the last user who changed a file
- This information will be kept by generating a new file with the following format <filename>.<last_user>
- If it is the first time that this user has changed the file, the system will raise an alert

Implementing the modifications

1. Narrow down the monitoring to a specific directory
2. Log the file name and UID
3. If it is the first time that this user has changed the file, raise an alert
4. Generate a file with the following format <filename>.<last_user>

UID to username

- The current UID can be identified, as in user space applications
- The /etc/passwd file contains information about the UIDs and usernames
- csd9999:x:1000:985::/home/csd9999:/usr/bin/oksh

Raising an alert

- An alert can be raised using printk()

- The message format could be:

  <login> ALERT file: <filename> user: <username>

  csd9999 ALERT file: foo.txt user: user1

- Locate the message using

  $ dmesg | grep <login>

  $ dmesg | grep csd9999

Simple demo program

- Choose the folder location that you wish to monitor and change its permissions so that all users have access
- The demo program issues *write* and *close* system calls on files in this directory
- <filename>.<last_user> files can be stored in the same folder or wherever you wish
- Alerts can be located using $ dmesg

Getting the Linux Kernel src code

$ cd /spare
$ mkdir <username>
$ chmod 700 <username>
$ cd <username>
$ cp ~hy457/qemu-linux/linux-2.6.38.1.tar.bz2 .
$ tar -jxvf linux-2.6.38.1.tar.bz2
$ cd linux-2.6.38.1
$ cp ~hy457/qemu-linux/.config .
Edit .config, find CONFIG_LOCALVERSION="-hy457" and modify it

Compile the Linux Kernel

- Edit the kernel source code to implement the system call modifications
- Compile the new kernel using
  $ make ARCH=i386 bzImage
- The new bzImage can be found under linux-2.6.38.1/arch/x86/boot/

Using Qemu

- Copy the virtual disk image and start the guest OS

```
$ cp ~hy457/qemu-linux/hy457-linux.img .
$ qemu-system-i386 -hda hy457-linux.img -curses
```

- Load the image and start the guest OS with the new kernel

```
$ qemu-system-i386 -hda hy457-linux.img
  -append " root=/dev/hda" -kernel
  linux-2.6.38.1/arch/x86/boot/bzImage -curses
```