浙江大學
ZHEJIANG UNIVERSITY

# Introduction: From Another Perspective

Yajin Zhou (http://yajin.org)

Zhejiang University

## What happens when a program runs?

- Execute instructions (obviously)
    - fetch, decode, and execute
- Others things are happening in the backend
    - make the program to run
    - allow many programs to use/share memory
    - allow may programs to interact with devices

## All about Virtualization

- Virtualization
  - OS transforms the physical resources into easy-to-use virtual form
  - Interaction: system calls - interfaces between program and OS
- Managing: resources manager

## Virtualizing The CPU

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <assert.h>
#include "common.h"

int
main(int argc, char *argv[])
{
    if (argc != 2)
    {
        fprintf(stderr, "usage: cpu <string>\n");
        exit(1);
    }
    char *str = argv[1];
    while (1)
    {
        Spin(1);
        printf("%s\n", str);
    }
    return 0;
}
```

# Virtualizing Memory

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int
main(int argc, char *argv[])
{
    int *p = malloc(sizeof(int));
    assert(p != NULL);
    printf("(%d) memory address of p: %08x\n", getpid(), (unsigned) p);

    *p = 0;
    while(1) {
        Spin(1);
        *p = *p + 1;
        printf("(%d) p: %d\n", getpid(), *p);
    }

    return 0;
}
```

# Concurrency

```c
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

volatile int counter = 0;
int loops;

void *worker(void *arg) {
    int i;
    for(i = 0; i < loops; i++) {
        counter++;
    }
    return NULL;
}

int
main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: threads <value>\n");
        exit(1);
    }

    loops = atoi(argv[1]);
    pthread_t p1, p2;
    printf("Initial value : %d\n", counter);

    Pthread_create(&p1, NULL, worker, NULL);
    Pthread_create(&p2, NULL, worker, NULL);
    Pthread_join(p1, NULL);
    Pthread_join(p2, NULL);
    printf("Final value : %d\n", counter);
    return 0;
}
```

## I/O

```c
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <fcntl.h>
#include <sys/types.h>

int
main(int argc, char *argv[])
{
    int fd = open("/tmp/file", O_WRONLY | O_CREAT | O_TRUNC, S_IRWXU);
    assert(fd > -1);
    int rc = write(fd, "hello world\n", 13);
    assert(rc == 13);
    close(fd);
    return 0;
}
```

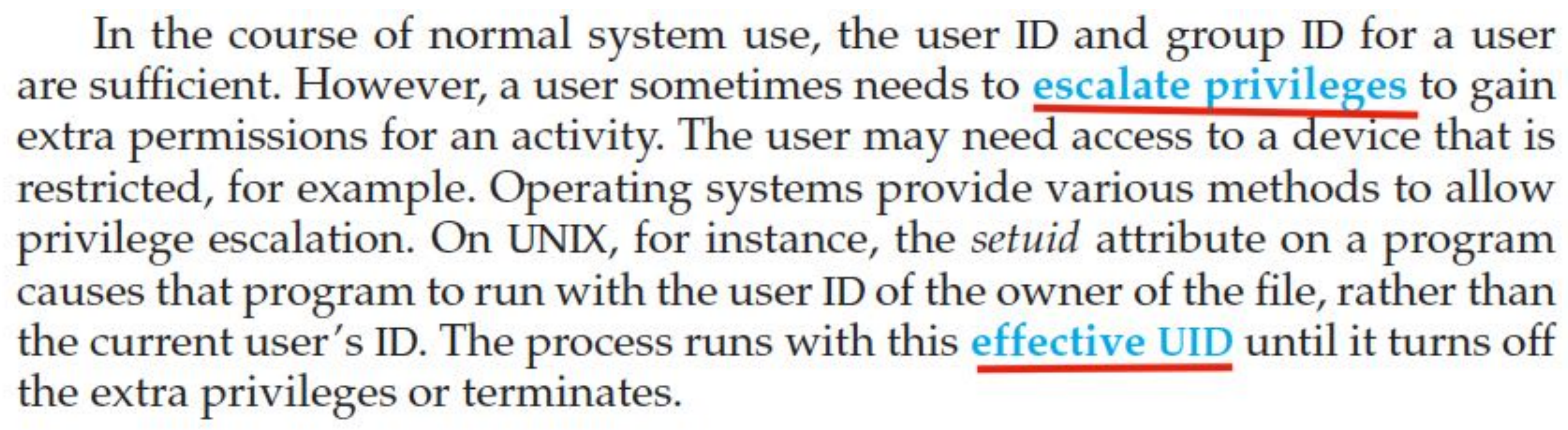
## Details

- File System: Where the data will reside on the disk
  - /tmp/file: directory path, file path
- Device driver: Issue I/O requests to underlying physical devices

## Design Goals

- OS
    - It virtualizes resources: CPU, memory, or disk
    - It handles related issues, e.g., concurrency
    - It stores files persistently
- Goals:
    - High performance
    - Protection
    - Reliability
    - Emergency-efficiency
    - Security

## A Little More About Security



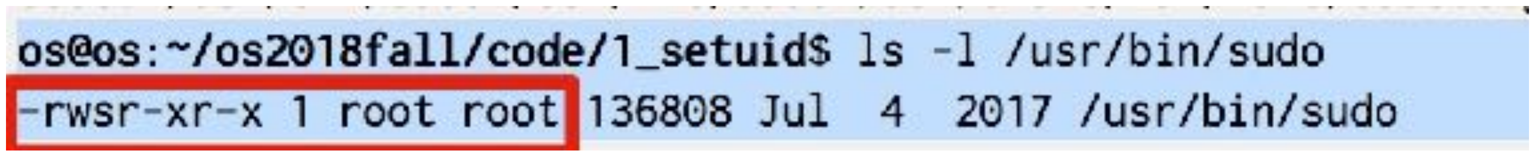
In the course of normal system use, the user ID and group ID for a user are sufficient. However, a user sometimes needs to escalate privileges to gain extra permissions for an activity. The user may need access to a device that is restricted, for example. Operating systems provide various methods to allow privilege escalation. On UNIX, for instance, the *setuid* attribute on a program causes that program to run with the user ID of the owner of the file, rather than the current user's ID. The process runs with this effective UID until it turns off the extra privileges or terminates.

## Three User IDs

- Real UID: real owner of the process
- Effective UID: ID used in access control
- Saved: Used to help disable/enable privileges
- Normally, real UID = effective UID

## Set-UID Mechanism

- Allow normal users to temporarily to gain higher privilege [Wiki]
- Why do we need this?



```
os@os:~/os2018fall/code/1_setuid$ ls -l /usr/bin/sudo
-rwsr-xr-x 1 root root 136808 Jul  4  2017 /usr/bin/sudo
```

# A Vulnerable Set-UID program

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

void main()
{
        int fd;
        char *v[2];

        fd = open("/root/flag", O_RDWR | O_APPEND);
        if (fd == -1) {
                printf("cannot open /root/flag \n");
                exit(0);
        }

        printf("fd is %d \n", fd);

        //change the uid to the normal one
        setuid(getuid());

        //Execute /bin/sh
        v[0] = "/bin/sh"; v[1] = NULL;

        execve(v[0], v, NULL);

}
```

Demo

Opened fd is not closed before executing the shell!

## RageAgainstTheCage

- CVE-2010-EASY
- https://thesnkchrmr.wordpress.com/2011/03/24/rageagainstthecage/

## Q&A