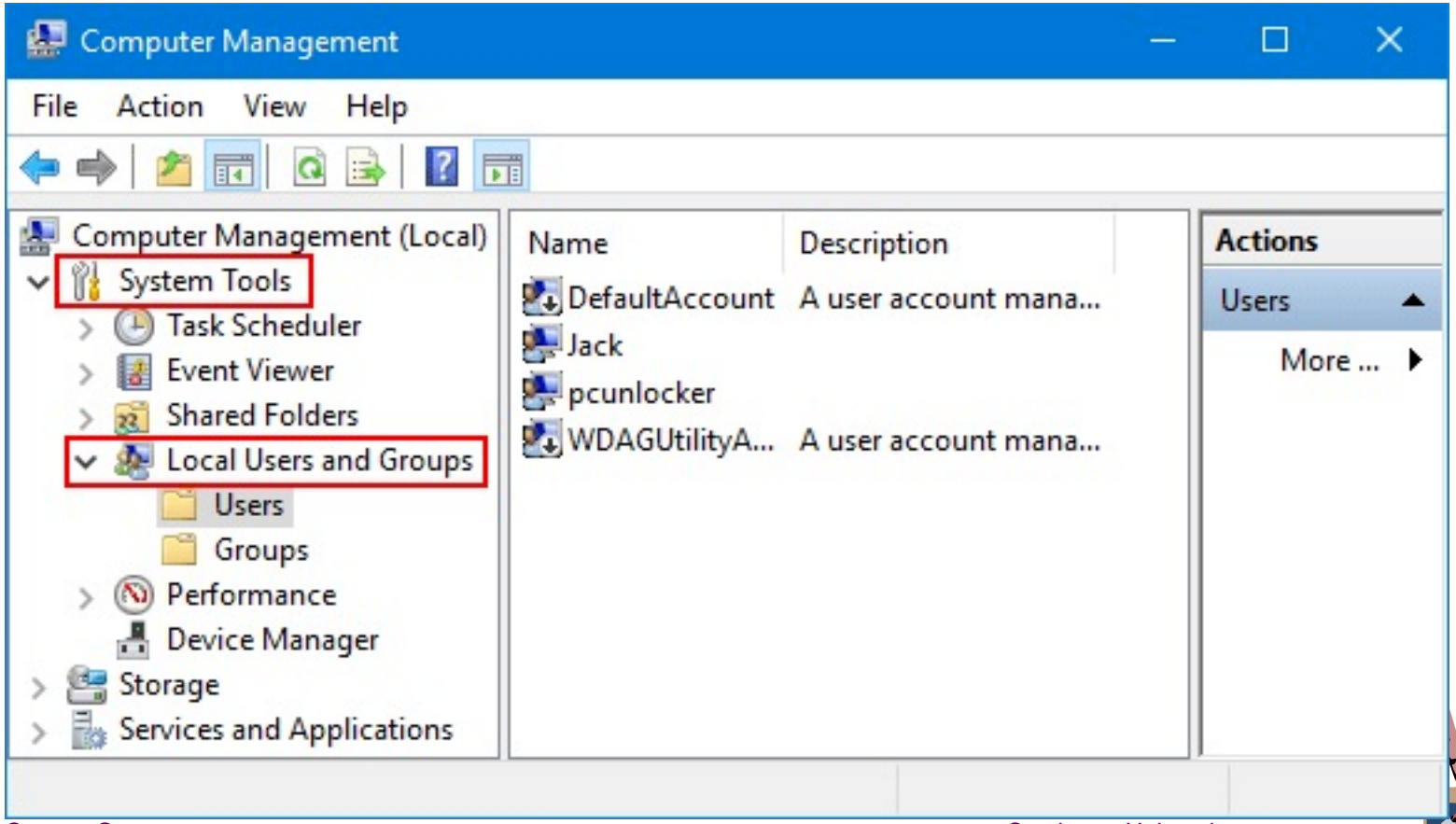
SET-UID Program and Attacks via Environment Variables

肖 卿 俊
办公室：九龙湖校区计算机楼212室
电邮：csqjxiao@seu.edu.cn
主页： https://csqjxiao.github.io/PersonalPage
电话：025-52091022

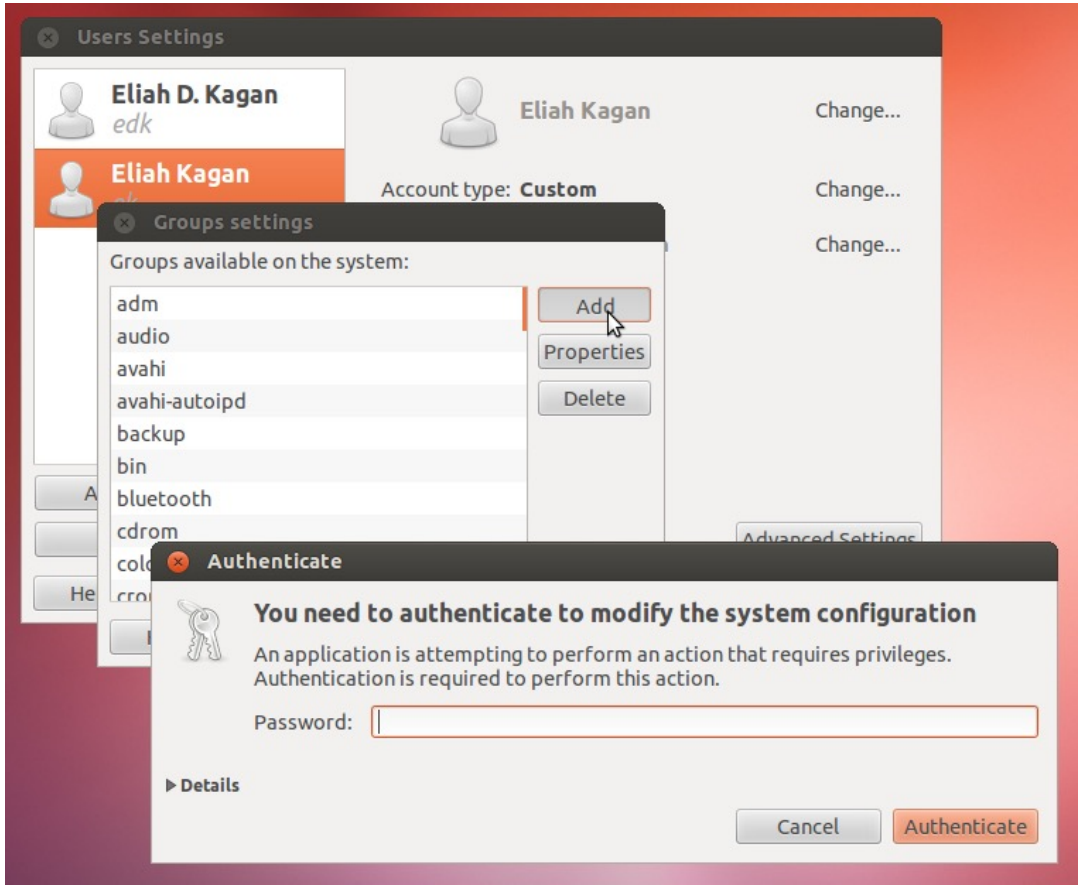
User and Group on Windows

- Each user has a user ID (uid).
- A user belongs to one or multiple groups



User and Group on Linux

- GUI is also available to manage users and groups on Ubuntu



User File on Linux System

- The user file is located in /etc/passwd
- There is a special privileged root user (uid=0)

Terminal

```
seed@VM:$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
seed:x:1000:1000:seed,,,:/home/seed:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
telnetd:x:121:129::/nonexistent:/bin/false
sshd:x:122:65534::/var/run/sshd:/usr/sbin/nologin
ftp:x:123:130:ftp daemon,,,:/srv/ftp:/bin/false
bind:x:124:131::/var/cache/bind:/bin/false
mysql:x:125:132:MySQL Server,,,:/nonexistent:/bin/false
bob:x:1001:1001:,,,:/home/bob:/bin/bash
```

root user is here

These are not real users

"seed" user for our security experiment

the first command after user loginv

Passwd is stored elsewhere

uid. gid Home Dir Shell

Group File on Linux System

- The group file is located in /etc/group

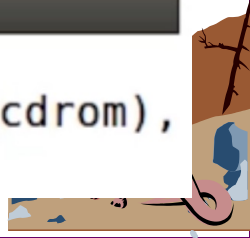
```
Terminal
seed@VM:$ cat /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:syslog,seed
tty:x:5:
disk:x:6:
lp:x:7:
mail:x:8:
news:x:9:
```

- The Linux command to add a user to a group

```
sudo usermod -a -G groupname username
```

- The command to show the IDs of current user

```
Terminal
seed@VM:$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),
27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
```



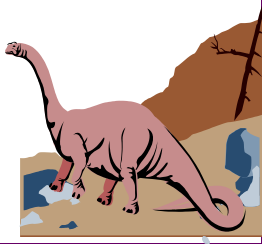
Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users RWX
  - a) **owner access** 7 ⇒ 1 1 1
  - b) **group access** 6 ⇒ 1 1 0
  - c) **public access** 1 ⇒ 0 0 1
- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.

owner group public

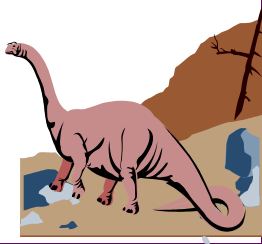
Attach a group to a file chmod 761 game

chgrp G game



A Sample UNIX Directory Listing

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4 pbg	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/



User File and Shadow File on Linux

- The shadow file stores the encrypted passwords of all users. It can be modified only by root. Normal users cannot read it.

```
Terminal
seed@VM$ ls -l /etc/passwd /etc/shadow
-rw-r--r-- 1 root root   2521 Nov  1 22:37 /etc/passwd
-rw-r----- 1 root shadow 1496 Nov  1 22:37 /etc/shadow
```

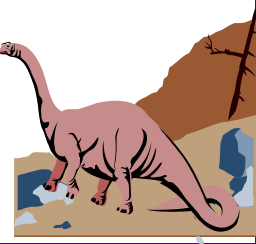
```
Terminal
seed@VM$ head -5 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
seed@VM$
seed@VM$ sudo head -5 /etc/shadow
root:$6$NrF46O1p$.vDnKEtVFC2bXslxkRuT4FcBqPpxLqW05IoECr0XKzEEO
5wj8aU3GRHW2BaodUn4K3vgyEjwPspr/kqzAqtcu.:17400:0:99999:7:::
daemon:*:17212:0:99999:7:::
bin:*:17212:0:99999:7:::
sys:*:17212:0:99999:7:::
sync:*:17212:0:99999:7:::
```

Need root user to read the shadow file

The encrypted password

A Short Recap

- /etc : Stores system administrative files and programs
- /etc/passwd : Stores all user information
- /etc/shadow : Stores all user passwords
- /etc/group : Stores all group information



Background Knowledge: Need for Privileged Programs

- Password Dilemma
  - Permissions of /etc/shadow File:

```
-rw-r----- 1 root shadow 1443 May 23 12:33 /etc/shadow
```

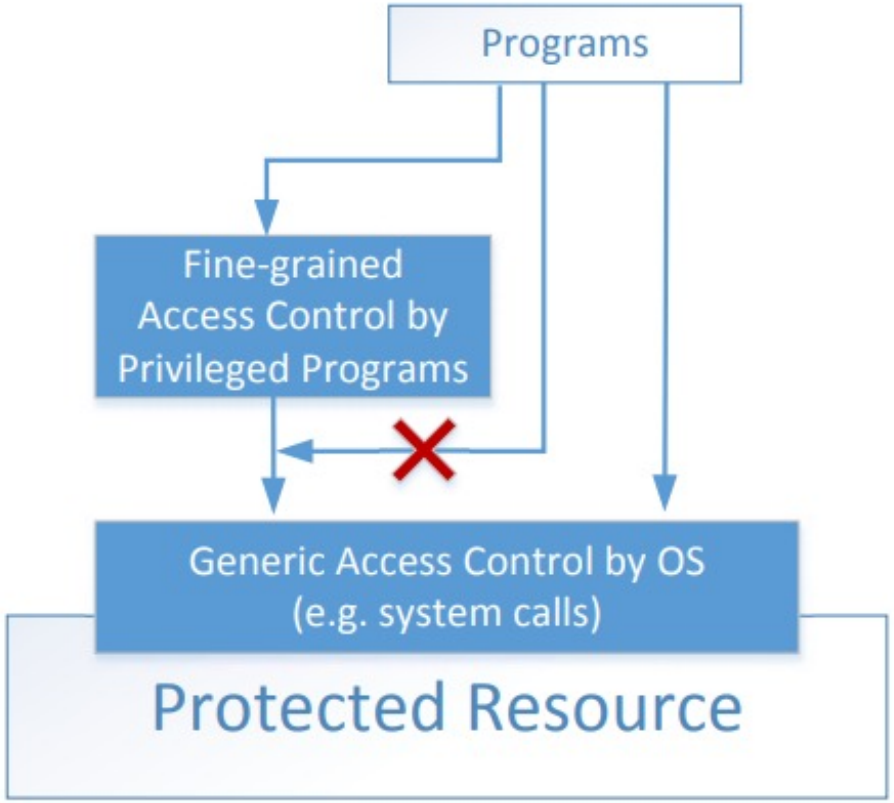
Only writable to the owner

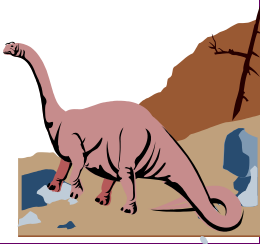
  - How would normal users change their password?

```
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWbQI1cFjn0R25yqtqrSrFeWfCgybQWWnwR4ks/.rjqyM7Xw
h/pDyc5U1BWOzkWh7T9ZGu.:15933:0:99999:7:::
daemon:*:15749:0:99999:7:::
bin:*:15749:0:99999:7:::
sys:*:15749:0:99999:7:::
sync:*:15749:0:99999:7:::
games:*:15749:0:99999:7:::
man:*:15749:0:99999:7:::
lp:*:15749:0:99999:7:::
```

Two-Tier Approach

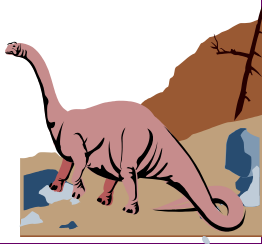
- Implementing fine-grained access control in operating systems make OS over complicated.
- OS relies on extension to enforce fine-grained access control
- Privileged programs are such extensions





Types of Privileged Programs

- Daemons
  - Computer program that runs in the background
  - Needs to run as root or other privileged users
- Set-UID Programs
  - Widely used in UNIX systems
  - Program marked with a special bit



Set-UID Concept

- **Allow user to run a program with the program owner's privilege.**
- Allow users to run programs with temporary elevated privileges
- Example: the `passwd` program

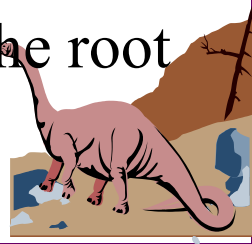
```
$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 41284 Sep 12  2012
/usr/bin/passwd
```

https://dev.to/0xbf/you-know-rwx-but-what-is-rws-when-run-ls-l-linux-tips-549c

- The "s" in "rws" means "set-uid" when run this program: **the program doesn't run with the privilege of user who ran it, instead, with the privilege of file owner.**

Set-UID Concept

- Every process has two User IDs.
- **Real UID (RUID)**: Identifies real owner of process
- **Effective UID (EUID)**: Identifies privilege of a process
  - Access control is based on EUID
- When a normal program is executed, RUID = EUID, they both equal to the ID of the user who runs the program
- When a Set-UID is executed, RUID ≠ EUID. RUID still equal to the user's ID, but EUID equals to the program **owner**'s ID.
  - If the program is owned by root, the program runs with the root privilege.



Turn a Program into Set-UID

- Change the owner of a file to root :

```
seed@VM:~$ cp /bin/cat ./mycat
seed@VM:~$ sudo chown root mycat
seed@VM:~$ ls -l mycat
-rwxr-xr-x 1 root seed 46764 Nov  1 13:09 mycat
seed@VM:~$
```

- Before Enabling Set-UID bit:

```
seed@VM:~$ mycat /etc/shadow
mycat: /etc/shadow: Permission denied
seed@VM:~$
```

- After Enabling the Set-UID bit:

```
seed@VM:~$ sudo chmod 4755 mycat
seed@VM:~$ mycat /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWbQI1cFjn(
h/pDyc5U1BWOzkWh7T9ZGu.:15933:0:99999:7:::
daemon:*:15749:0:99999:7:::
bin:*:15749:0:99999:7:::
sys:*:15749:0:99999:7:::
```

How it Works

- A Set-UID program is just like any other program, except that it has a special marking, which a single bit called Set-UID bit
  - /usr/bin/id command prints a message containing the system identifications (ID) for the current user

```
$ cp /bin/id ./myid
$ sudo chown root myid
$ ./myid
uid=1000(seed) gid=1000(seed) groups=1000(seed), ...
```

  - If myid is a rooted-owned set-uid program

```
$ sudo chmod 4755 myid
$ ./myid
uid=1000(seed) gid=1000(seed) euid=0(root) ...
```

Another Example of Set-UID Program

```
$ cp /bin/cat ./mycat
$ sudo chown root mycat
$ ls -l mycat
-rwxr-xr-x 1 root seed 46764 Feb 22 10:04 mycat
$ ./mycat /etc/shadow
./mycat: /etc/shadow: Permission denied
```

- Not a privileged program

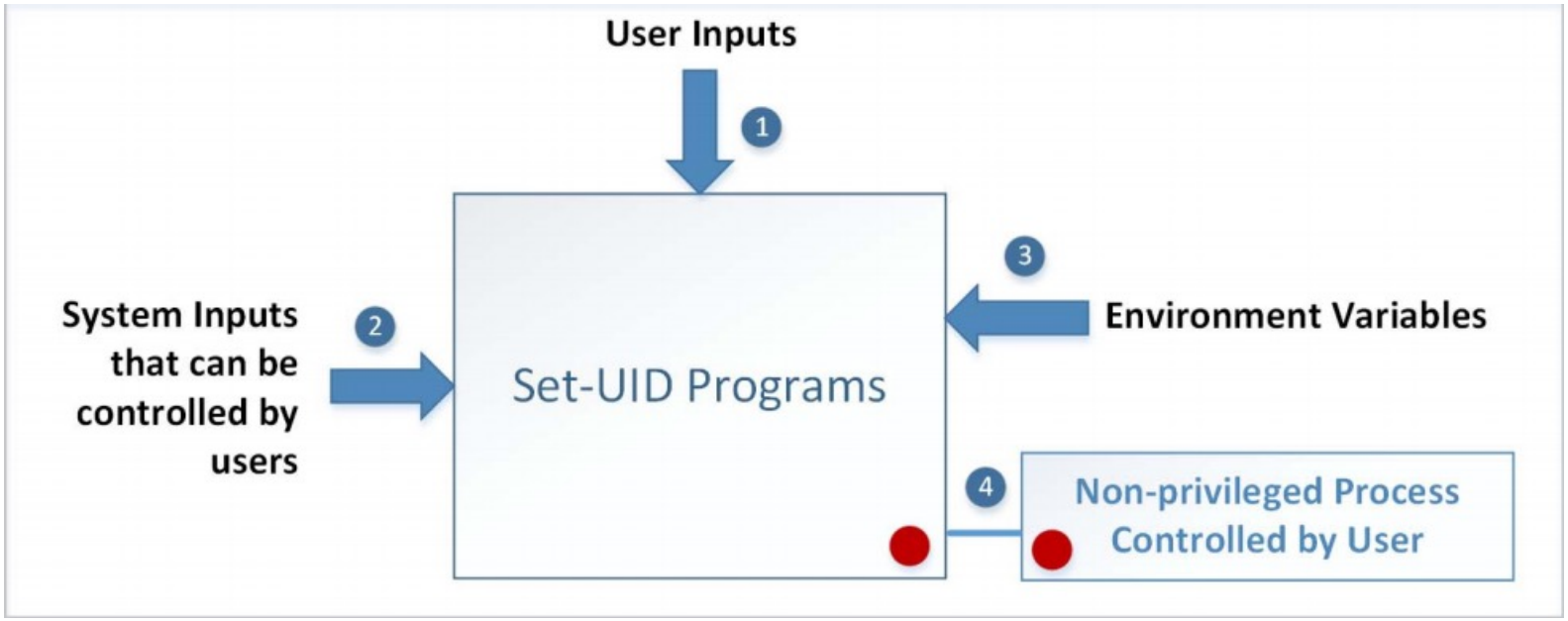
```
$ sudo chmod 4755 mycat
$ ./mycat /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8c...
daemon:*:15749:0:99999:7:::
...
```

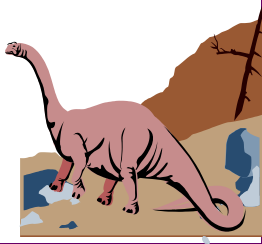
- Become a privileged program

```
$ sudo chown seed mycat
$ chmod 4755 mycat
$ ./mycat /etc/shadow
./mycat: /etc/shadow: Permission denied
```

- It is still a privileged program, but not the root

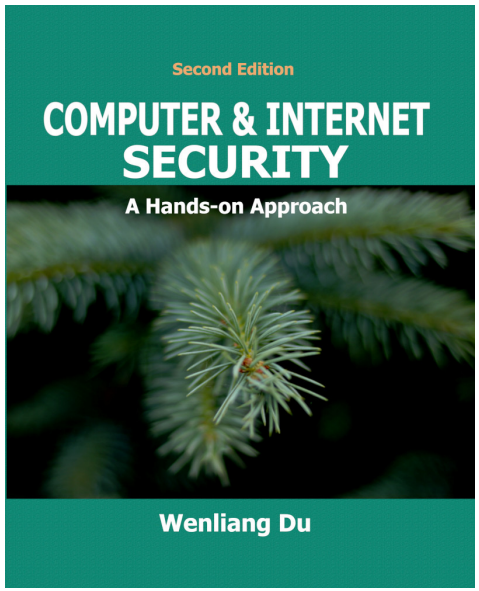
Attack Surfaces of Set-UID Programs

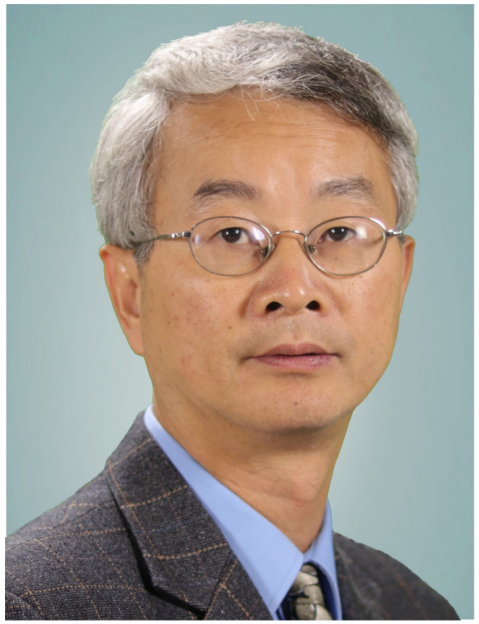




Computer & Internet Security: A Hands-on Approach

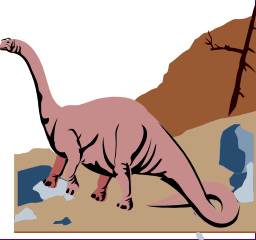
- 该书涵盖计算机和网络空间安全的基础知识，包括应用软件安全、Web 安全和网络安全。 本书旨在帮助读者了解各种各样的攻击和防御的思想与原理。





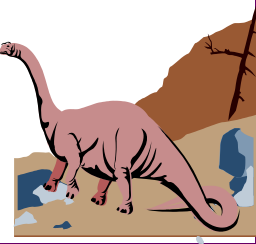
作者：杜文亮，电气工程与计算机科学系正教授

https://www.handsonsecurity.net/index.html



Attacks via Environment Variables

- Behavior can be influenced by inputs that are not visible inside a program.
- Environment Variables : These can be set by a user before running a program.



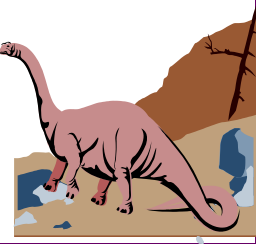
Attacks via User Inputs

User Inputs: Explicit Inputs

- Buffer Overflow – More information in Chapter 4
  - ✓ Overflowing a buffer to run malicious code
- Format String Vulnerability – More information in Chapter 6
  - ✓ Changing program behavior using user inputs as format strings

软件安全

- 第1章 Set-UID 特权程序原理及攻击
- 第2章 通过环境变量实现攻击
- 第3章 Shellshock 攻击
- 第4章 缓冲区溢出攻击
- 第5章 return-to-libc 攻击
- 第6章 格式化字符串漏洞
- 第7章 竞态条件漏洞
- 第8章 脏牛竞态条件攻击

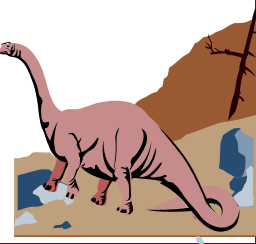


Attacks via System Inputs

System Inputs

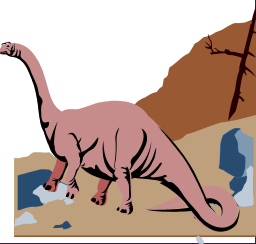
- Race Condition – More information in Chapter 7
  - ✓ Symbolic link to privileged file from a unprivileged file
  - ✓ Influence programs
  - ✓ Writing inside world writable folder
- Dirty Copy-on-Write Attack – More information in Chapter 8

软件安全
第1章 Set-UID 特权程序原理及攻击
第2章 通过环境变量实现攻击
第3章 Shellshock 攻击
第4章 缓冲区溢出攻击
第5章 return-to-libc 攻击
第6章 格式化字符串漏洞
第7章 竞态条件漏洞
第8章 脏牛竞态条件攻击



Attacks via Environment Variables

- `PATH` Environment Variable
  - Used by shell programs to locate a command if the user does not provide the full path for the command
  - system(): call /bin/sh first
  - system("ls")
    - ✓ /bin/sh uses the PATH environment variable to locate "ls"
    - ✓ Attacker can manipulate the PATH variable and control how the "ls" command is found



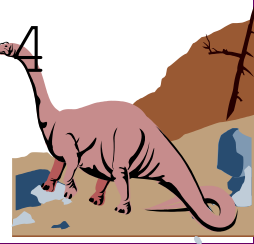
An example of modifying $PATH

- **env** is a shell command for Unix and Unix-like operating systems. It is used to print a list of environment variables.
  - Bring up a shell, and run these commands

```
cp /usr/bin/env ./myenv
sudo chown root myenv
sudo chmod 4755 myenv
```

  - By the command "`ls -l ./myenv`", we will see myenv is a root-owned set-UID program

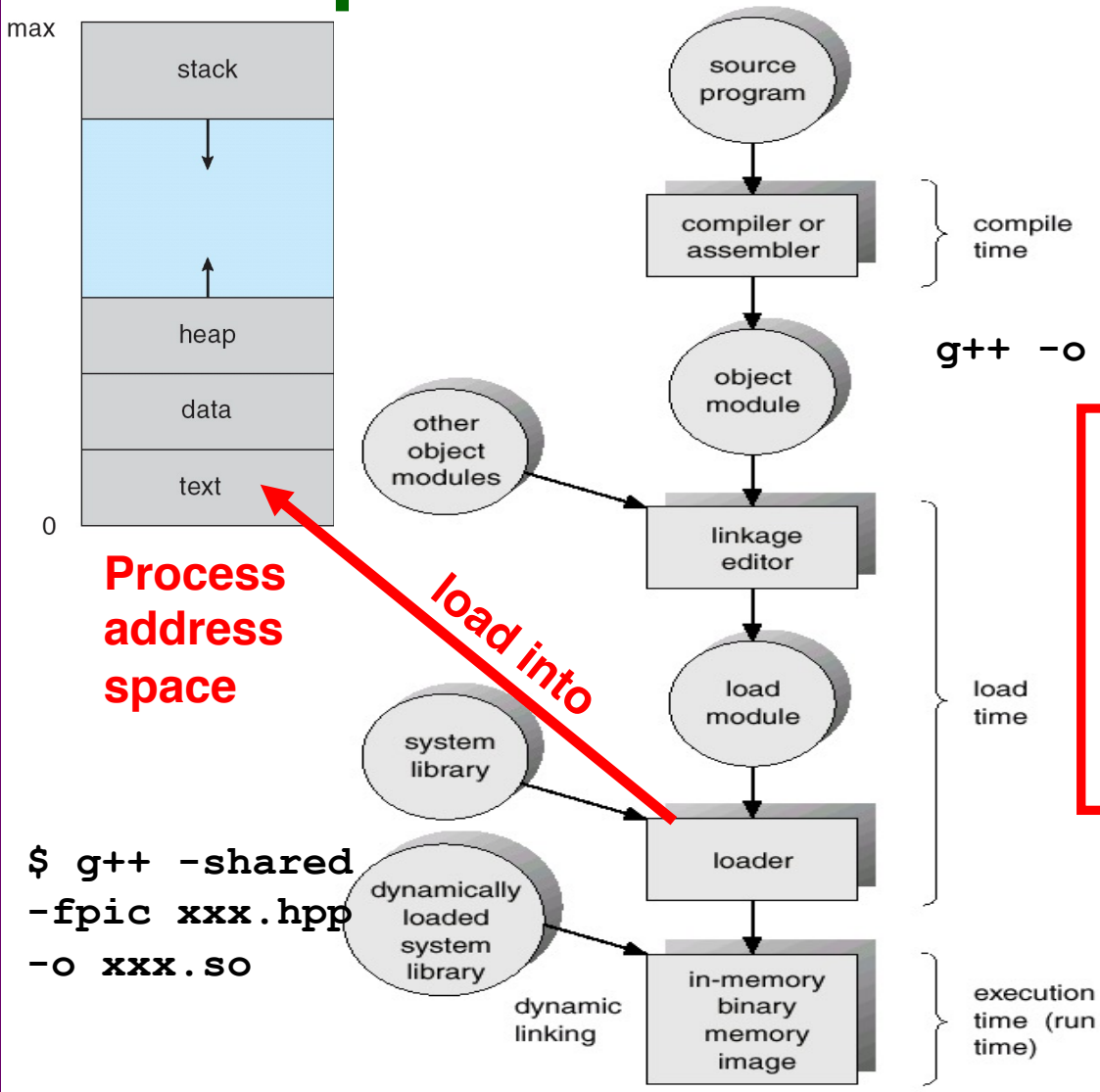
```
-rwsr-xr-x  1 root  staff  121824
Mar 27 15:49 ./myenv
```



An example of modifying $PATH

- In Linux and Unix system, **export** command can be used to create/change an environment variable
  - Bring up a shell, and run these commands
    ```
    export $PATH=/to/my/path:$PATH
    echo $PATH
    ```
  - You will see the output on screen
    ```
    /to/my/path:/usr/local/bin:/usr/bin:/bin:/usr/sbin:…
    ```
  - By the angle of myenv program, $PATH is also changed
    ```
    ./myenv | grep $PATH
    PATH=/to/my/path:/usr/local/bin:…
    ```
  - If the privileged program runs xxx executable, you can put your own version of xxx into /to/my/path

The exec() API can load a user program into the code segment of a process. How does it happen?




```
g++ -c -o foo.o foo.c
g++ -c -o main.o main.c
g++ -c -o bar.o bar.c
```

```
g++ -o fubar foo.o main.o bar.o -lm
```

If you are calling the sqrt() function from the math library, you need to include the math.h header file in your .C file and you need to explicitly link the math library into your executable

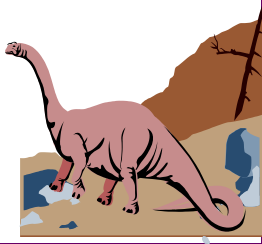
```
$ g++ -shared
-fpic xxx.hpp
-o xxx.so
```

Arguments of g++ compiler

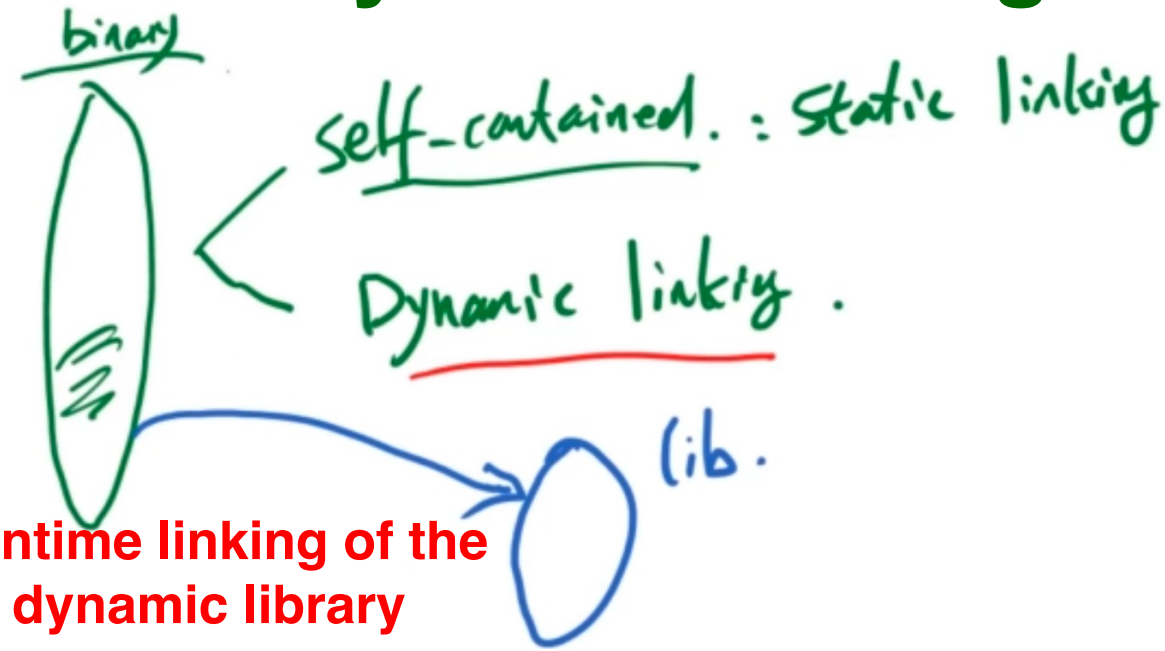
- **-o *<name>*** - name of the output file
- **-l*<library>*** - link with library. Static libraries have names like xxx.a. Shared libraries have names like xxx.so.

Dynamic Linking

- **Static linking** – system libraries and program code combined by the loader into the binary program image
- **Dynamic linking** – linking postponed until run time
- Small piece of code, **stub,** used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system checks if routine is in processes' memory address
  - If not in address space, add to address space
- Dynamic linking is particularly useful for libraries
- System also known as **shared libraries**
- Consider applicability to patching system libraries
  - Versioning may be needed



Static vs. Dynamic Linking Library



Runtime linking of the dynamic library

An Example:

```
/* hello.c */
# include <stdio.h>
int main()
{
    printf("hello world");
    return 0;
}
```

printf(...) uses libc

```
$ gcc -o hello_dynamic hello.c
$ gcc -static -o hello_static hello.c
$ ls -l
-rw-rw-r-- 1 seed seed     68 Dec 31 13:30 hello.c
-rwxrwxr-x 1 seed seed   7162 Dec 31 13:30 hello_dynamic
-rwxrwxr-x 1 seed seed 751294 Dec 31 13:31 hello_static
```

Static linking of the libc library leads to a much larger binary code

How to show the shared library of an executable?

- Use the ldd command:

```
NAME
       ldd - print shared library dependencies

SYNOPSIS
       ldd [OPTION]...  FILE...

DESCRIPTION
       ldd  prints  the  shared  libraries  required by each program or shared
       library specified on the command line.
```

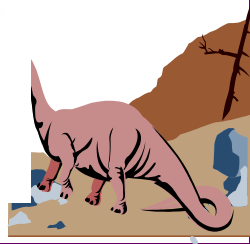
- Run ldd on an exemplified binary

```
void main()
{
   printf("Hello World\n");
}
seed@ubuntu:$ ldd a.out
        linux-gate.so.1 =>  (0xb7fff000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7e42000)
        /lib/ld-linux.so.2 (0x80000000)
```

system call

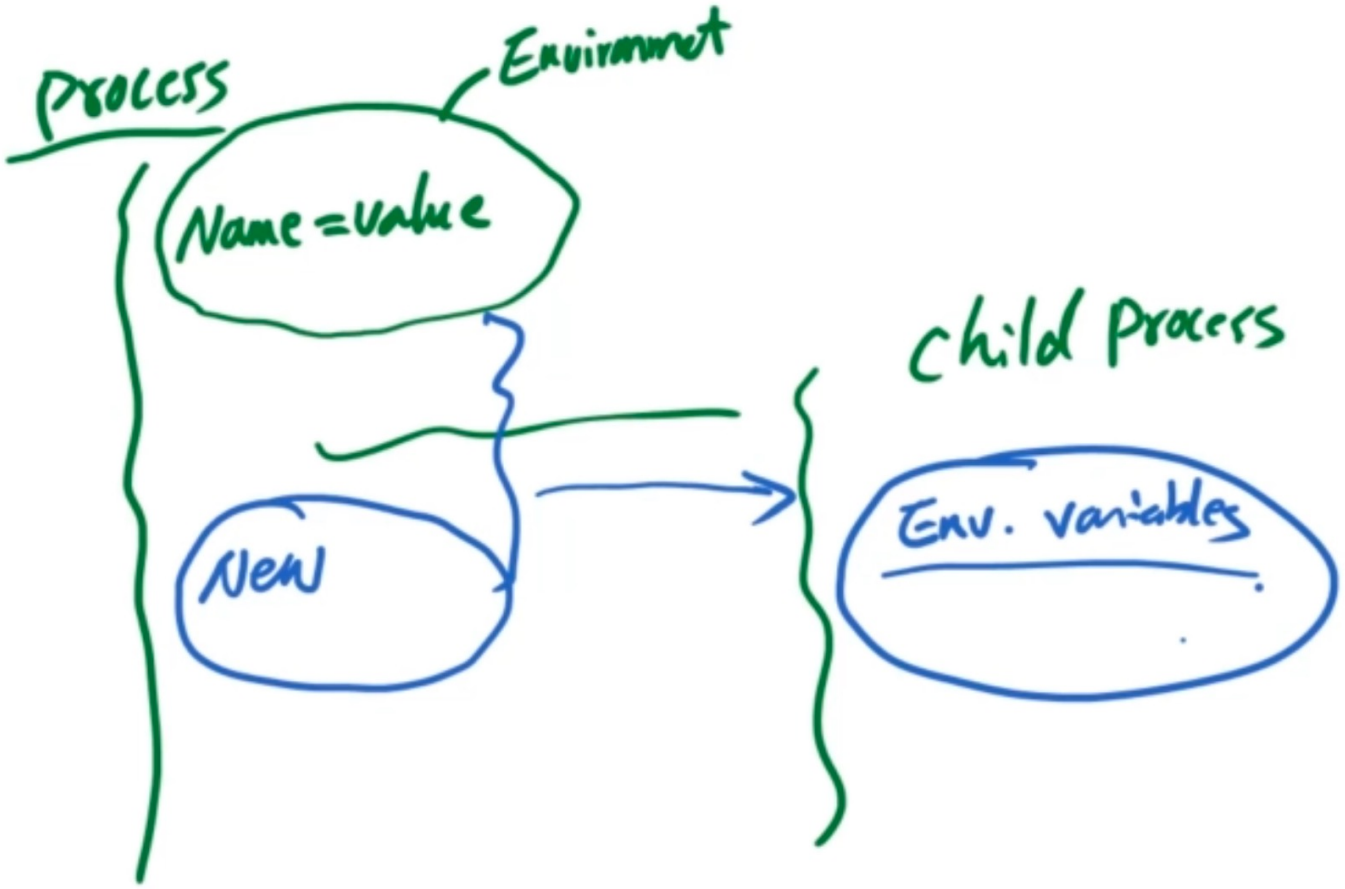
libc

linker



A child process can inherit environment variables from its parent

- Environment variables are a set of key-value pairs, which can be passed from parent to child



Linux has two environment variables to affect the dynamic linker/loader

- **LD_PRELOAD:** A list of additional, user-specified, ELF shared objects to be loaded before all others.
  - ELF (Executable Linkable Format) is a common standard file format for executable files, object code, shared libraries, and core dumps

- **LD_LIBRARY_PATH:** A list of directories in which to search for ELF libraries at execution time.

Environment variable LD_PRELOAD affects the dynamically linked library

- Use ldd command to demystify the loading process

```
void main()
{
   printf("Hello World\n");
   sleep(2);
}
```

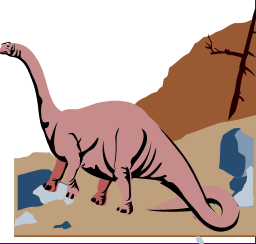
Here the sleep(…) API is called?
Where is it implemented?

```
seed@ubuntu:$ unset LD_PRELOAD
seed@ubuntu:$ ldd a.out
        linux-gate.so.1 =>  (0xb7fff000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7e42000)
        /lib/ld-linux.so.2 (0x80000000)
seed@ubuntu:$ a.out
Hello World
```

- What if we export the LD_PRELOAD variable?

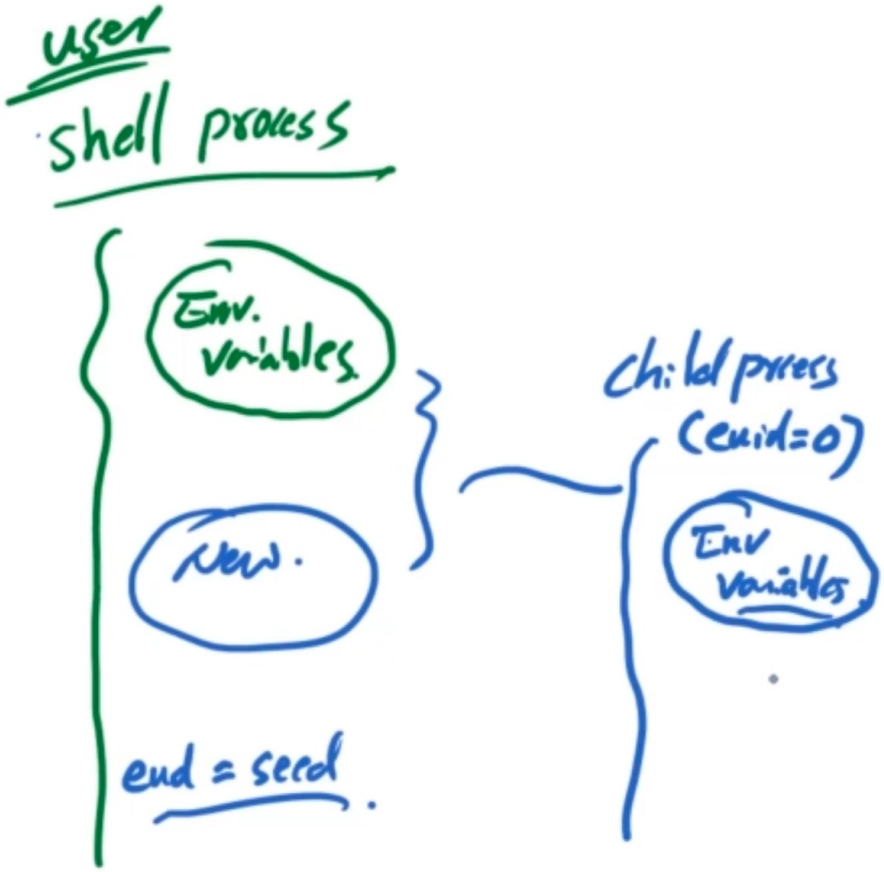
```
seed@ubuntu:$ export LD_PRELOAD=./libmylib.so.7
seed@ubuntu:$ ldd a.out
        linux-gate.so.1 =>  (0xb7fff000)
        ./libmylib.so.7 (0xb7ffa000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7e3f000)
        /lib/ld-linux.so.2 (0x80000000)
seed@ubuntu:$ a.out
Hello World
I am not sleeping!
```

```
#include <stdio.h>
void sleep(int s)
{
   printf("I am not sleeping!\n");
}
```



If a root owned SET-UID Process Inherit Environment Variable from its Parent Process, Then

- Environment Variable may become an attack surface of a root owned SET-UID process



LD_PRELOAD may be an attack surface of SET-UID program. Linux fix this problem.

- Experiment: create a root owned SET-UID myenv

```
seed@ubuntu:$ cp /usr/bin/env ./myenv
seed@ubuntu:$ sudo chown root myenv
[sudo] password for seed:
seed@ubuntu:$ sudo chmod 4755 myenv
seed@ubuntu:$ ls -l myenv
-rwsr-xr-x 1 root seed 22060 Dec 27 09:30 myenv
```

- Export some LD_xxx environment variables

```
seed@ubuntu:$ export LD_PRELOAD=./libmylib.so.1.0.1
seed@ubuntu:$ export LD_LIBRARY_PATH=.
seed@ubuntu:$ export LD_MYOWN="my own value"
```

- The difference between system env and myenv

```
seed@ubuntu:$ env | grep LD_
LD_PRELOAD=./libmylib.so.1.0.1
LD_LIBRARY_PATH=.
LD_MYOWN=my own value
seed@ubuntu:$ myenv | grep LD_
LD_MYOWN=my own value
```

SET-UID process filters the environment variables copied from its parent process, i.e., the shell

System() call is more dangerous

- The `system(cmd)` function executes the `/bin/sh` program first, and then asks this shell program to run the `cmd` command.
- A vulnerable root-owned SET-UID program

catall.c

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  char *cat="/bin/cat";

  if(argc < 2) {
    printf("Please type a file name.\n");
    return 1;
  }

  char *command = malloc(strlen(cat) + strlen(argv[1]) + 2);
  sprintf(command, "%s %s", cat, argv[1]);
  system(command);
  return 0 ;
}
```

Set by the developer

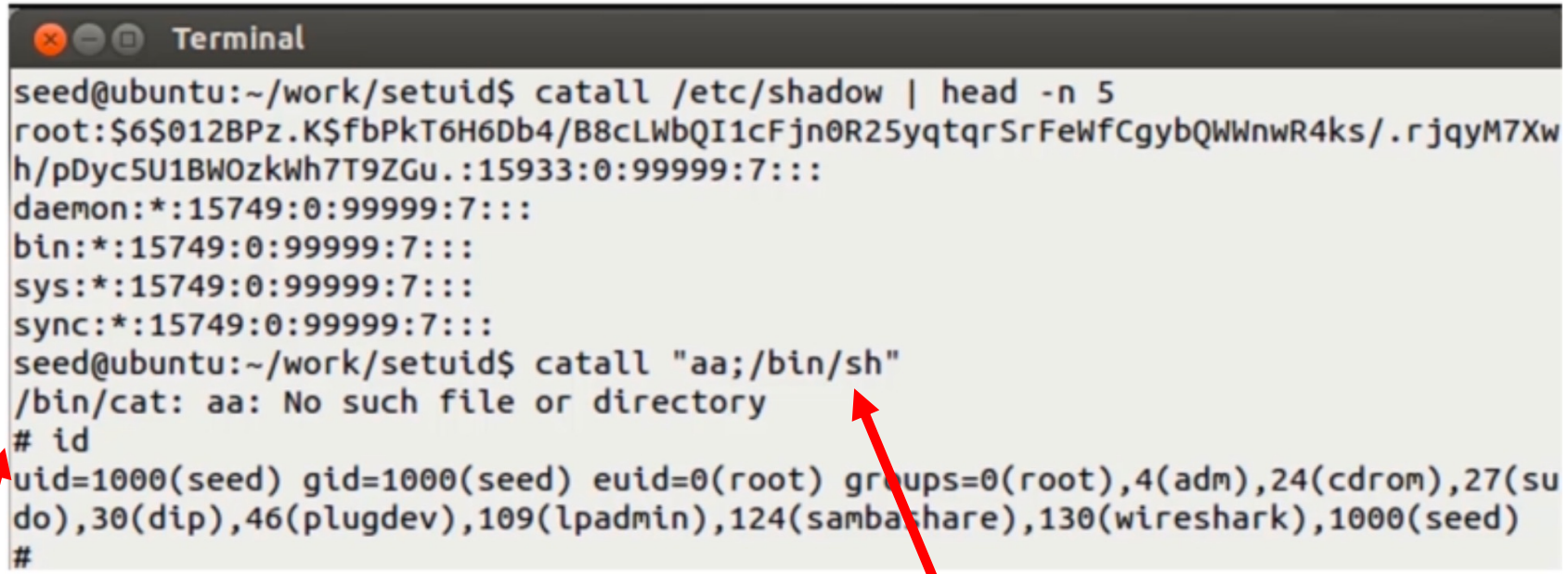
Set by user

Root privilege

Equivalent to $/bin/cat

a ; new cmd

By exploiting this vulnerability, an attacker can escalate its privilege to root



Shell with root privilege

There must be quotes here. Otherwise, the string

```
$ catall aa; /bin/sh
```

will be interpreted by shell as two commands