CE4062/CZ4062

Computer Security

Lecture 5: Operating System Security

Tianwei Zhang

Outline

- **Operating System Security Basis**
- **Security Protection Stages employed by OS**
- **UNIX Security Model**
- **Security Vulnerabilities in OS**

Outline

- **Operating System Security Basis**
- **Security Protection Stages employed by OS**
- **UNIX Security Model**
- **Security Vulnerabilities in OS**

OS Becomes More Complex

From single-user to multi-user

- DOS is truly single user
- MacOS, Linux, NT-based Windows are multi-user, but typically only one user in PCs.
- Cloud computing allows multiple users all over the world to run on the same OS, and they do not know each other.
- Tradeoff: efficiency versus security

From trusted apps to untrusted apps

- Simple real-time systems: only run one specific app
- Runs verified apps from trusted parties
- Modern PCs and smartphones: run apps from third-party developers
- Tradeoff: functionality versus security

Complex OS Brings More Challenges

Protecting a single computer with one user is easy

- Prevent everybody else from having access
- Encrypt all data with a key only one person knows

Sharing resources safely is hard

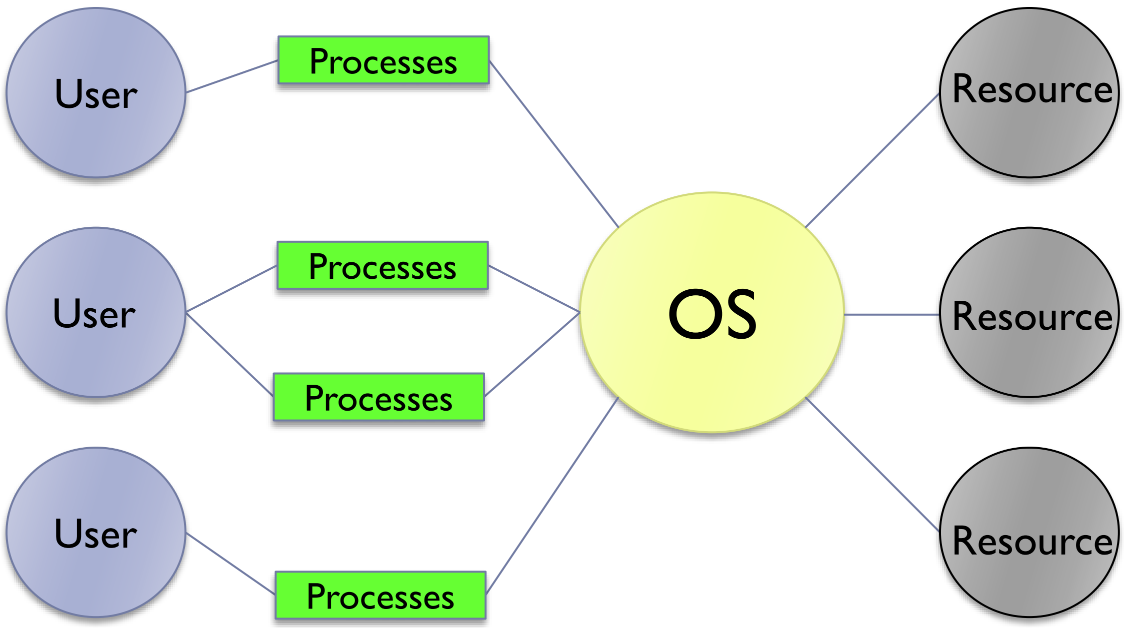
- Preventing some people from reading private data (e.g., medical record, financial data, employee information)
- Prevent some people from using too many resources (e.g., disk space, CPU core)
- Prevent some people from interfering with other programs (e.g., inserting keystrokes / modifying displays)

OS Responsibility

Functionalities		Security goals
❑ Support multiple users concurrently	→	❑ Protect users from each other
❑ Manage multiple apps concurrently	→	❑ Protect apps from each other
❑ Connect to the network	→	❑ Protect the system from the untrusted network
❑ Sharing data with different domains	→	❑ Secure the data sharing

What's being protected? Resources

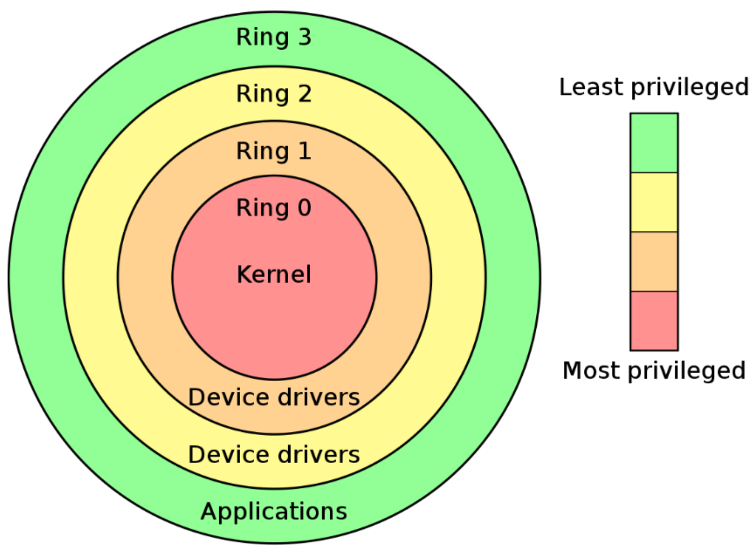
System is secure if resources are used and accessed as intended under all circumstances



Privileged Rings Inside OS

Operating modes

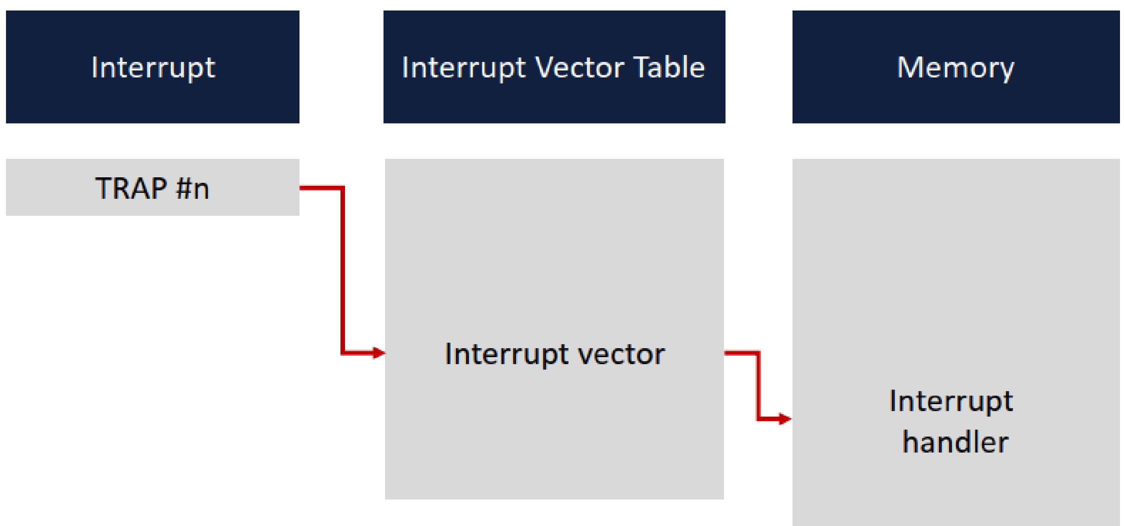
- Kernel mode has the highest privileges, running the critical functions and services
- Entities with the higher privilege levels cannot call the functions and access the objects in the lower privilege levels directly.
  - System call, interrupt, etc.
- Status flag allows system to work in different modes (context switching)



Interrupt

Context switch

- The user-level process generates a trap, which will incur an interrupt
- The CPU stores the process's states, and switches to the kernel mode by setting the status flag.
- The kernel handles the interrupt based on the trap address (interrupt vector) in an interrupt table.
- The CPU switches back to the user mode and restores the states.



Process Security

A program in execution, consisting of executable code, data, and the execution context, e.g., the contents of certain CPU registers.

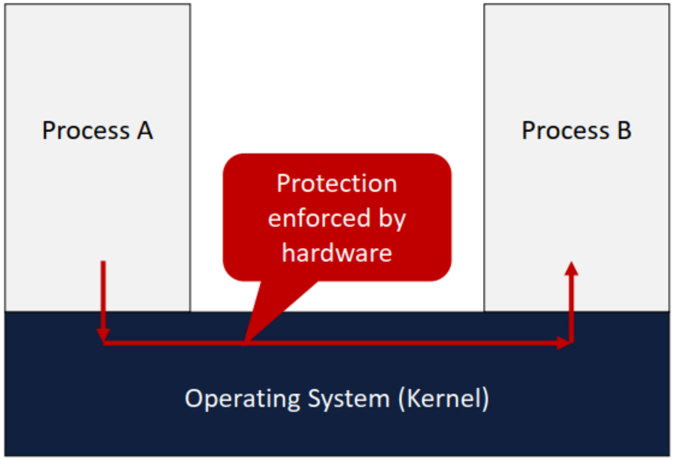
Process isolation

- A process has its own address space
- Logical separation of processes as a basis for security: an independent process cannot affect or be affected by the execution of other processes

Process communication

- A cooperating process can affect or be affected by the execution of other processes
- Such processes have to communicate with each other to share data

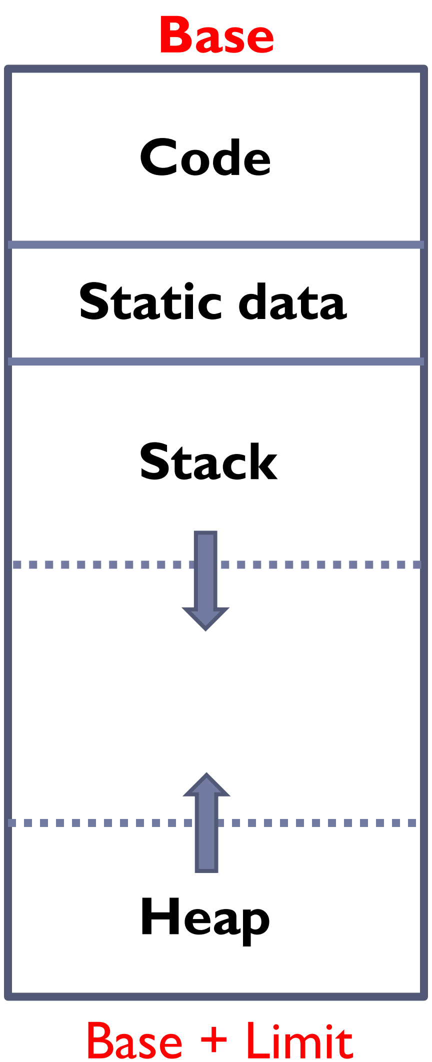
  - Inter-Process Communication
- A context switch between processes can be an expensive and insecure operation



Memory Management

Memory layout

- Each process is allocated a segment of memory for storing data and computation results
- Memory scope is restricted by Base and Limit
- Divided into memory pages of equal lengths.
- A process is not allowed to access memory pages not belonging to it
  - The OS will check if each memory access is allowed.
  - A page fault will be generated if a memory access is illegal.



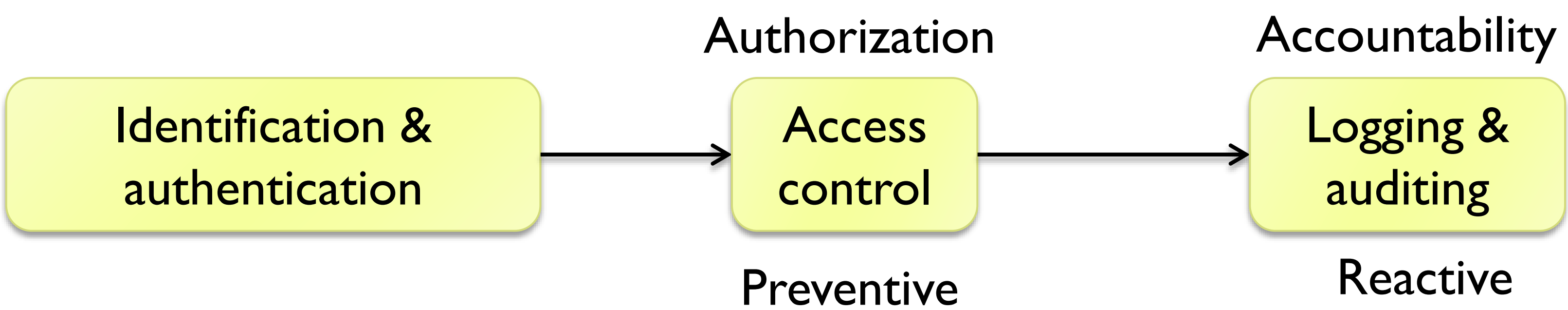
Outline

- Operating System Security Basis
- **Security Protection Stages employed by OS**
- UNIX Security Model
- Security Vulnerabilities in OS

Security Protection from OS

OS is responsible for protecting the apps running on it

- OS controls what users/processes can do



Identification & authentication → Access control → Logging & auditing

How does a computer know if I am a correct user?

- What you know? password, PIN, public/private keys…
- What you have? smartcard, hardware tokens…
- Who you are? biometrics, face recognition, voice recognition…

How does the system conduct authentication?

- Compare the input credential with the stored one
- Allow entry when the credential matches

Identification & authentication → **Access control** → Logging & auditing

Principle of least privilege

- Users should only have access to the resources needed to perform the desired tasks.
- Too much privilege allows a malicious user to conduct unintended activities

Privilege separation

- Spit a system into different components, and each component is assigned with the least privilege for its tasks
- Limiting the privilege can prevent any attacker from taking over the entire system.

Security Policy

Specify (Subject, Object, Operation) triples

Subject

- Acting system principals.
- User, program, process…

Object

- Protected resources.
- File, memory, devices…

Operation

- How subject operates on objects
- Read, write, execute, delete…

Access Control Policy

Who sets policy?

- Users, with some system restrictions

Access Control Matrix

- Each column represents an object
- Each row represents a subject
- The entry shows the allowed verbs.

	/etc	/homes	/usr
Alice	Read	Read	Read Write
Bob	Read Write	Read Write	Read Write
Carl	None	None	Read

How is policy enforced?

- OS exposes API to apps, with privileged operations
- Checks access control matrix when API functions are called

Update Access Control Matrix

Access Control Changes

- Grant capabilities: the owner of the object can grant rights to other users.
- Revoke capabilities: subjects can revoke the rights from others

Six Commands to Alter the Access Matrix

- create subject s: creates a new subject s.
- create object o: creates a new object o.
- enter r into Ms,o: adds right r to cell Ms,o.
- delete r from Ms,o: deletes right r from cell Ms,o.
- destroy subject s: deletes subject s. The column and row for s in M are also deleted.
- destroy object o: deletes object o. The column for o in M is also deleted.

More Representations

Access Control List (ACLs)

- For one object, which subject has accesses to it? (check the column in the Access Control Matrix)

Capability

- For one subject, which objects it has capability to access? (check rows in the Access Control Matrix)

Most systems use both

- ACLs for opening an object, e.g. fopen()
- Capabilities for performing operations, e.g. read()

	/etc	/homes	/usr
Alice	Read	Read	Read Write
Bob	Read Write	Read Write	Read Write
Carl	None	None	Read

Data Sharing

Problem: multiple users want to access the same file or data

- Give each user the corresponding permissions.
- When a new user joins, the permissions have to be granted again.
- When permissions are changed, need to alter each user.

Solution: group

- Set permissions for the group instead of the user
- A user joining the group will have the corresponding permissions.
- A user quitting the group will loss the corresponding permissions.
- Easier to manage and update.

Identification & authentication → Access control → **Logging & auditing**

Audit trail

- Recording all protection-orientated activities, important to understanding what happened, why, and catching things that shouldn't

/usr/adm/lastlog	Records the last time a user has logged in; displayed with **finger**
/var/adm/utmp	Records accounting information used by the **who** command.
/var/adm/wtmp	Records every time a user logs in or logs out; displayed with the **last** command.
/var/adm/acct	Records all executed commands; displayed with **lastcomm**
/var/log/	In modern Linux systems, log files are located in there

Outline

- Operating System Security Basis
- Security Protection Stages employed by OS
- **UNIX Security Model**
- Security Vulnerabilities in OS

UNIX

A family of multitasking, multiuser computer operating systems

- Developed by Dennis Ritchie & Ken Thompson at AT&T Bell Labs in 1969
- Originally for small multi-user computers in a friendly network environment;
- Later scaled up to commercial; improved gradually.

Several flavors of Unix: Unix-like OS

- Vendor versions differ in the way some security controls are managed & enforced.
- Solars, FreeBSD, macOX, …

Security was not a primary design goal.

- Dominant goals were modularity, portability and efficiency.
- Now it provides sufficient security mechanisms that have to be properly configured and administered.

Users

A system can have many accounts

- Service accounts: running background processes
- User accounts: tied to each person

User identifiers (UIDs)

- A 16-bit number (size of UID values varies for different systems)
- 0 reserved for root, 1-99 for other predefined accounts, 100-999 for system accounts/groups. User accounts start from 1000.
  - e.g., root (0); bin (1); daemon (2); mail (8); news (9); diego (261)

Superuser

A special privileged principal

- UID 0 and usually the username root
- All security checks are turned off for superuser
  - All access control mechanisms turned off
  - Can become an arbitrary user
  - Can change system clock
- Some restrictions remain but can be overcome:
  - Cannot write to a read-only file system but can remount it as writeable.
  - Cannot decrypt passwords but can reset them.

Responsibility

- Used by the operating system for essential tasks like login, recording the audit log, or access to I/O devices.
- A major weakness of Unix; an attacker achieving superuser status effectively takes over the entire system

Protecting Superuser

- Privilege separation; create users like uucp or daemon to deal with networking; if a special users is compromised, not all is lost.
- root should not be used as a personal account.

User Information

User account stored in /etc/passwd:

- Username: : used when user logs in, 1–32 characters long
- Password: 'x' indicates that encrypted password is stored in /etc/shadow
- UID: the user ID
- GID: the primary group ID
- Name: a comment field
- Homedir: the path the user will be in when they log in
- Shell: the absolute path of a command or shell

```
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
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
landscape:x:111:117::/var/lib/landscape:/usr/sbin/nologin
usbmux:x:112:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
tianweiz:x:1000:1000:Tianwei Zhang:/home/tianweiz:/bin/bash
```

Groups

Users belong to one or more groups.

- Group info stored in /etc/group
  - group name
  - password
  - GID
  - list of users
- Each user belongs to a primary group. The ID can be found in /etc/passwd.

Group is convenient for access control decisions.

- It is easier for users to share the same access control permission or resources.
- e.g., put all users allowed to access email in a group called mail

```
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:syslog,tianweiz
tty:x:5:
disk:x:6:
lp:x:7:
mail:x:8:
news:x:9:
uucp:x:10:
man:x:12:
proxy:x:13:
kmem:x:15:
dialout:x:20:
fax:x:21:
voice:x:22:
cdrom:x:24:tianweiz
floppy:x:25:
tape:x:26:
sudo:x:27:tianweiz
audio:x:29:
dip:x:30:tianweiz
www-data:x:33:
backup:x:34:
operator:x:37:
list:x:38:
```

Processes

A process has a process ID (PID)

- New processes generated with exec or fork.

A process has two types of User ID (UID) and Group ID (GID)

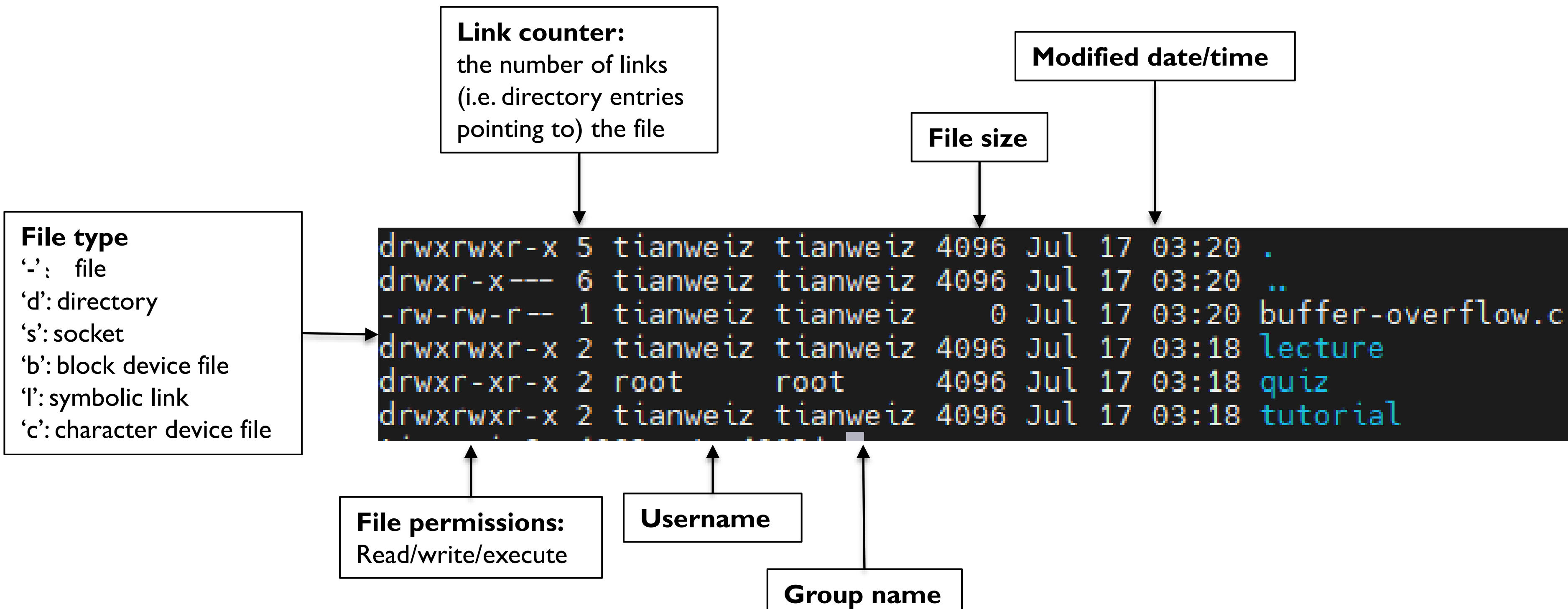
- Real UID/GID: typically the UID/GID of the user that logs into the system
- Effective UID/GID: inherited from the parent process or from the file being executed. This determines the permissions for this process

	UID		GID	
Process	**Real**	**Effective**	**Real**	**Effective**
/bin/login	root	root	system	system
User dieter logs on; the login process verifies the password and changes its UID and GID:				
/bin/login	dieter	dieter	staff	staff
The login process executes the user's login shell:				
/bin/bash	dieter	dieter	staff	staff
From the shell, the user executes a command, e.g. ls				
/bin/ls	dieter	dieter	staff	staff
The User executes command su to start a new shell as root:				
/bin/bash	dieter	root	staff	system

File, Directory and Devices

Files, directories, memory devices, I/O devices are uniformly treated as resources

- These resources are the objects of access control.
- Each resource has a single user owner and group owner



File Permission

Three permissions with three subjects

- Read, Write, Execute
- Owner, Group, Other
- Examples:
  - rw-r--r--: read and write access for the owner, read access for group and other.
  - rwx------: read, write, and execute access for the owner, no rights to group and other.

Octal Representation

- rw-r--r--: 110 100 100: 644
- rwxrwxrwx: 111 111 111: 777

Adjust permission:

- Users can change the permissions:
  - chmod 754 filename
  - chmod u+wrx,g+rx,g-w,o+r,o-wx filename
- root can change the ownerships:
  - chown user:group filename

Controlled Invocation

Superuser privilege is required to execute certain OS functions

- Example: password changing
  - User passwords are usually stored in the file /etc/shadow
  - This file is owned by the root superuser. So a regular user does not have R/W access to it.
  - When a user wants to change his password with the program passwd, this program needs to give him additional permissions to write to /etc/shadow

SUID: a special flag for a program

- If SUID is enabled, then user who executes this progam will inherit the permissions of that program's owner.
- passwd has such SUID enabled. When a user executes this program to change his password, he gets additional permissions to write the new password to /etc/shadow

```
root@cx4062:~# ls -al /usr/bin/passwd
-rwsr-xr-x 1 root root 59976 Mar 14 08:59 /usr/bin/passwd
```

The execute permission of the owner is given as s instead of x

Security of Controlled Invocation

Many other SUID programs with the owner of root

- /bin/passwd: change password
- /bin/login: login program
- /bin/at: batch job submission
- /bin/su: change UID program

Potential dangers

- As the user has the program owner's privileges when running a SUID program, the program should only do what the owner intended
- By tricking a SUID program owned by root to do unintended things, an attacker can act as the root

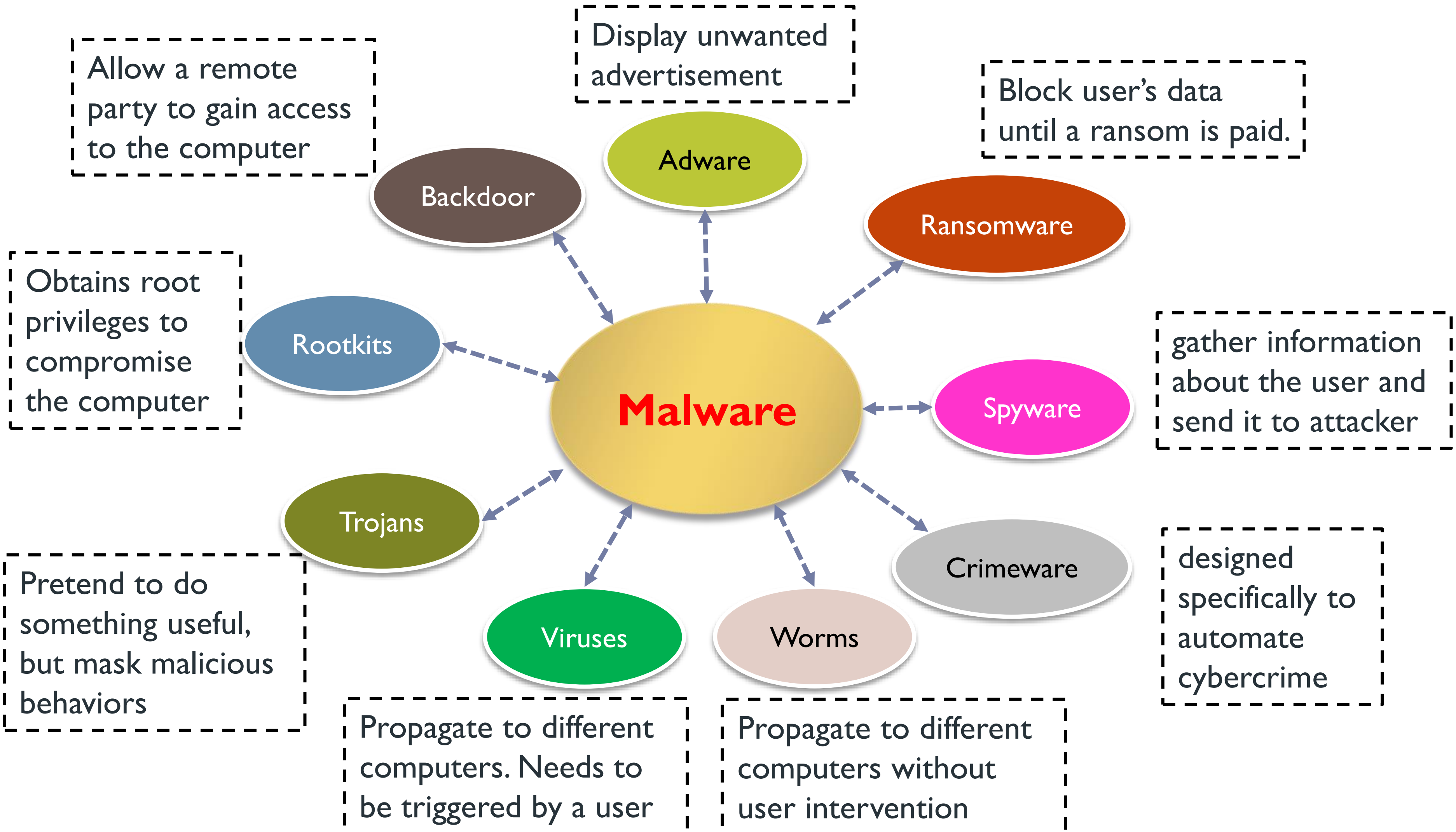
Security consideration

- All user input (including command line arguments and environment variables) must be processed with extreme care
- Programs should have SUID status only if it is really necessary.
- The integrity of SUID programs must be monitored.

Outline

- Operating System Security Basis
- Security Protection Stages employed by OS
- UNIX Security Model
- **Security Vulnerabilities in OS**

Different Kinds of Malware



Rootkit

Malware that obtains root privileges to compromise the computer

- Root user does not go though any security checks, and can perform any actions to the system
  - Insert and execute arbitrary malicious code in the system's code path
  - Hide its existence, e.g., malicious process, files, network sockets, from being detected.

How can the attacker gain the root privileges?

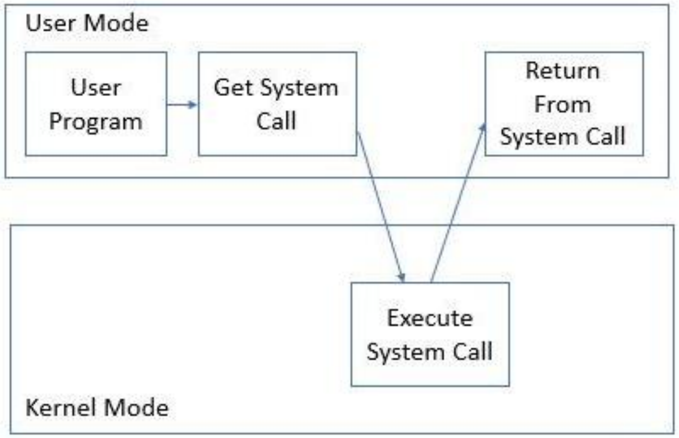
- Vulnerabilities in the software stack: buffer overflow, format string…

There are some common techniques for rootkits to compromise the systems.

System-call Table

A system call is an interface that allows a user-level process to request functions or services from the kernel level.

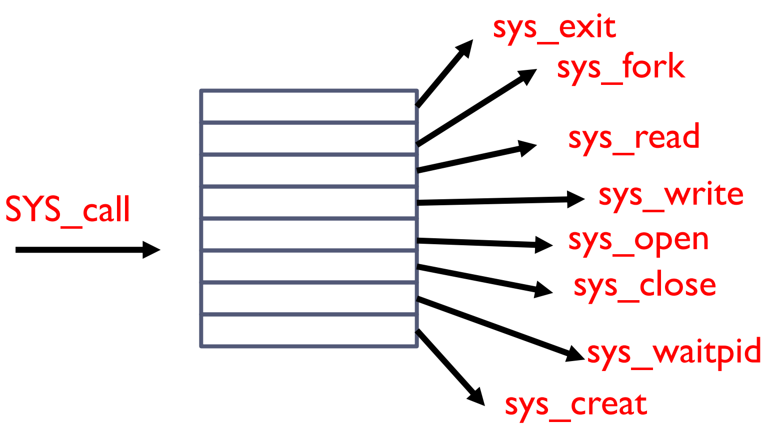
- Examples:
  - Process control
  - File management
  - Device management



How to issue a system call?

- System call table: a table of pointers in the kernel region, to different system call functions.
- A user process passes the index of the system call and corresponding parameters with the following API:

  syscall(SYS_call, arg1, arg2, …);



Highjack System-call Table

Rootkit change pointers of certain entries in the system-call table.

- Other processes calling these system calls will execute the attacker's code

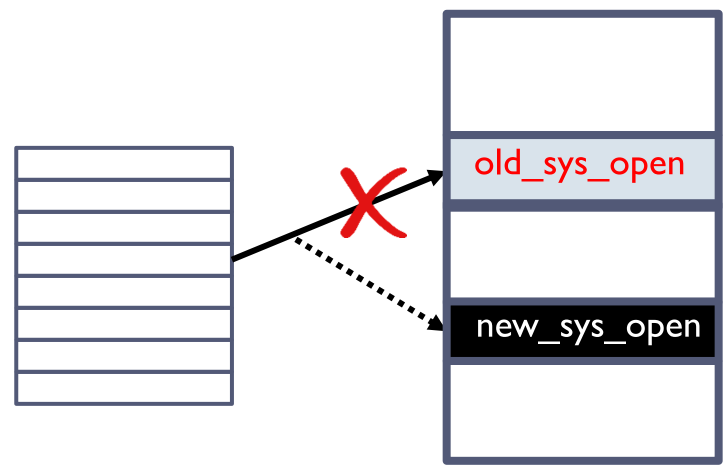
An example

- syscall_open is used to display the running process (ps command)
- Rootkit redirects this system call to new_syscall_open
  - When the object to be opened matches the malicious name, return NULL to hide it
  - Otherwise, call normal old_syscall_open

```
struct file sysmap = open("System.map-version");
long *syscall_addr = read_syscall_table(sysmap);

old_syscall_open = syscall_addr[__NR_open];
syscall_addr[__NR_open] = new_syscall_open();

malicious_object_name = {"xingyi", "bind_shell",
      "reverse_shell"...};
int new_syscall_open(char *object_name) {
    if strstr(object_name, malicious_object_name)
        return NULL;
    return old_syscall_open(object_name)
}
```



https://sw0rdm4n.wordpress.com/2014/11/03/xingyiquan-simple-linuxkernel-rootkit-for-kernel-3-x-and-kernel-2-6-x/

Compromise System Call Functions

In addition to change the pointer, rootkit can directly change the system call function.

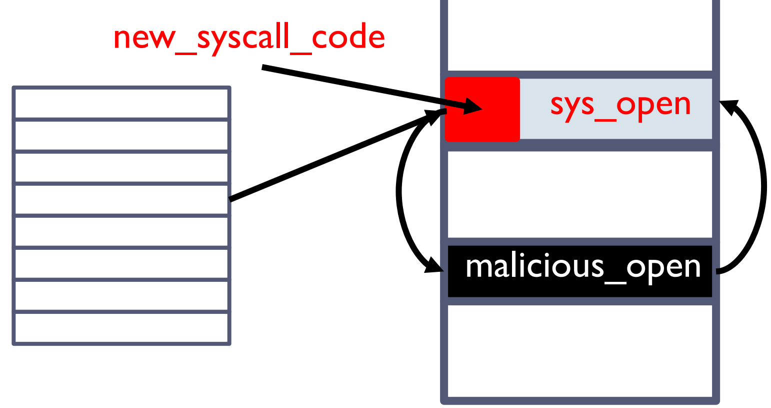
- syscall_open is used to display the running process (ps command)
- Replace the first 7 bytes of syscall_open as jump to malicious_open.
  - This faked system call will issue malicious function, restore the original system call and then call the correct one.

```
struct file sysmap = open("System.map-version");
long *syscall_addr = read_syscall_table(sysmap);
syscall_open = syscall_addr[__NR_open];

char old_syscall_code[7];
memncpy(old_syscall_code, syscall_open, 7);

char pt[4];
memncpy(pt, (long)malicious_open, 4)
char new_syscall_code[7] =
{"\xbd",pt[0],pt[1],pt[2],pt[3], // movl %pt, %ebp
"\xff","\xe5"};                  // jmp %ebp
memncpy(syscall_open, new_syscall_code, 7);

int malicious_open(char *object_name) {
    malicious_function();
    memncpy(syscall_open, old_syscall_code, 7);
    return syscall_open(object_name);
}
```



http://www.ouah.org/stealth-syscall.txt

Highjack Interrupt Descriptor Table

An interrupt is a signal from the hardware or software to notify the processor that something needs to be handled immediately.

- After receiving the signal, the processor issues the interrupt handler
- Interrupt Descriptor Table (IDT): a table of pointers to different interrupt handler functions

Rootkit can alter the pointer in the IDT to make the processor execute wrong functions.

```c
unsigned char idtr[6];
unsigned long idt_addr;
__asm__ volatile ("sidt %0" : "=m" (idtr));
idt_addr = *((unsigned long *)&idtr[2]);

old_idt_handler = idt_addr[handler_id];
idt_addr[handler_id] = new_idt_handler;

void new_idt_handler(args){
    malicious_function();
    return old_idt_hanlder(args);
}
```