Operating Systems: Internals and Design Principles

Chapter 15 Operating System Security

Ninth Edition
By William Stallings

System Access Threats

System access threats fall into two general categories: = Intruders + Malicious software

Intruders

Masquerader

An individual who is not authorized to use the computer and who penetrates a system's access controls to exploit a legitimate user's account

(Likely an outsider)

Misfeasor

A legitimate user who accesses data, programs, or resources for which such access is not authorized, or who is authorized for such access but misuses his or her privileges

(Likely an insider)

Clandestine user

An individual who seizes supervisory control of the system and uses this control to evade auditing and access controls or to suppress audit collection

(Can be either insider or outsider)

Malicious Software

- Programs that exploit vulnerabilities in computing systems
- Also referred to as malware
- Can be divided into two categories:
  - Parasitic
    - Fragments of programs that cannot exist independently of some actual application program, utility, or system program
    - Viruses, logic bombs, and backdoors are examples
  - Independent
    - Self-contained programs that can be scheduled and run by the operating system
    - Worms and bot programs are examples

Countermeasures

- Intrusion Detection
- Authentication
- Access Control

Intrusion Detection Systems

- RFC 4949 (*Internet Security Glossary)* defines intrusion detection as a security service that monitors and analyzes system events for the purpose of finding, and providing real-time or near real-time warning of, attempts to access system resources in an unauthorized manner

- Intrusion detection systems (IDSs) can be classified as:
  - Host-based IDS
    - Monitors the characteristics of a single host and the events occurring within that host for suspicious activity
    - OSSEC, Tripwire
  - Network-based IDS
    - Monitors network traffic for particular network segments or devices and analyzes network, transport, and application protocols to identify suspicious activity
    - Snort, Zeek

IDS Components

Sensors

- Responsible for collecting data
- The input for a sensor may be any part of a system that could contain evidence of an intrusion
- Types of input to a sensor include network packets, log files, and system call traces

Analyzers

- Receive input from one or more sensors or from other analyzer
- Responsible for determining if an intrusion has occurred
- May provide guidance about what actions to take as a result of the intrusion

User interface

- Enables a user to view output from the system or control the behavior of the system
- May equate to a manager, director, or console component

Authentication

- In most computer security contexts, user authentication is the fundamental building block and the primary line of defense
- RFC 4949 defines user authentication as the process of verifying an identity claimed by or for a system entity
- An authentication process consists of two steps:
  - Identification step
    - Presenting an identifier to the security system (means by which a user provides a claimed identity to the system)
  - Verification step (means of establishing the validity of the claim.)
    - Presenting or generating authentication information that corroborates the binding between the entity and the identifier

Means of Authentication

- Something the individual knows
  - Examples include a password, a personal identification number (PIN), or answers to a prearranged set of questions
- Something the individual possesses
  - Examples include electronic keycards, smart cards, and physical keys
  - Referred to as a ***token***
- Something the individual is (static biometrics)
  - Examples include recognition by fingerprint, retina, and face
- Something the individual does (dynamic biometrics)
  - Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm

Access Control

- Implements a security policy that specifies who or what may have access to each specific system resource and the type of access that is permitted in each instance
- Mediates between a user and system resources, such as applications, operating systems, firewalls, routers, files, and databases
- A security administrator maintains an authorization database that specifies what type of access to which resources is allowed for this user
    - The access control function consults this database to determine whether to grant access
- An auditing function monitors and keeps a record of user accesses to system resources

Firewalls

Traditionally, a firewall is a dedicated computer that interfaces with computers outside a network and has special security precautions built into it to protect sensitive files on computers within the network.

Design goals:

1) All traffic from inside to outside, and vice versa, must pass through the firewall. This is achieved by physically blocking all access to the local network except via the firewall
2) Only authorized traffic, as defined by the local security policy, will be allowed to pass. Various types of firewalls are used, which implement various types of security policies
3) The firewall itself is immune to penetration. This implies the use of a hardened system with a secured operating system. Trusted computer systems are suitable for hosting a firewall and often required in government applications

Buffer Overflow Attacks

- Also known as a ***buffer overrun***
- Defined in the NIST (National Institute of Standards and Technology) *Glossary of Key Information Security Terms* as:

  "A condition at an interface under which more input can be placed into a buffer or data-holding area than the capacity allocated, overwriting other information. Attackers exploit such a condition to crash a system or to insert specially crafted code that allows them to gain control of the system"

- One of the most prevalent and dangerous types of security attacks

```
int main(int argc, char *argv[]) {
    int valid = FALSE;
    char str1[8];
    char str2[8];

    next_tag(str1);
    gets(str2);
    if (strncmp(str1, str2, 8) == 0)
        valid = TRUE;
    printf("buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);
}
```

(a) Basic buffer overflow C code

```
$ cc -g -o buffer1 buffer1.c
$ ./buffer1
START
buffer1: str1(START), str2(START), valid(1)
$ ./buffer1
EVILINPUTVALUE
buffer1: str1(TVALUE), str2(EVILINPUTVALUE), valid(0)
$ ./buffer1
BADINPUTBADINPUT
buffer1: str1(BADINPUT), str2(BADINPUTBADINPUT), valid(1)
```

(b) Basic buffer overflow example runs

Figure 15.1 Basic Buffer Overflow Example

Memory Address	Before gets(str2)	After gets(str2)	Contains Value of
. . . .	. . . .	. . . .	
bffffbf4	34fcffbf 4 . . .	34fcffbf 3 . . .	argv
bffffbf0	01000000	01000000	argc
bffffbec	c6bd0340 . . . @	c6bd0340 . . . @	return addr
bffffbe8	08fcffbf	08fcffbf	old base ptr
bffffbe4	00000000	01000000	valid
bffffbe0	80640140 . d . @	00640140 . d . @	
bffffbdc	54001540 T . . @	4e505554 N P U T	str1[4-7]
bffffbd8	53544152 S T A R	42414449 B A D I	str1[0-3]
bffffbd4	00850408	4e505554 N P U T	str2[4-7]
bffffbd0	30561540 0 V . @	42414449 B A D I	str2[0-3]
. . . .	. . . .	. . . .	

Figure 15.2 Basic Buffer Overflow Stack Values

Exploiting Buffer Overflow

- To exploit any type of buffer overflow the attacker needs:
  - To identify a buffer overflow vulnerability in some program that can be triggered using externally sourced data under the attackers control
  - To understand how that buffer will be stored in the processes memory, and hence the potential for corrupting adjacent memory locations and potentially altering the flow of execution of the program

Compile-Time Defenses

- Countermeasures can be broadly classified into two categories:

  1) Compile-time defenses, which aim to harden programs to resist attacks

  2) Runtime defenses, which aim to detect and abort attacks in executing programs

Compile-time Techniques

- **Choice of programming language**
  - One possibility is to write the program using a modern high-level programming language that has a strong notion of variable type and what constitutes permissible operations on them
  - The flexibility and safety provided by these languages does come at a cost in resource use, both at compile time and also in additional code that must execute at runtime

- **Language extensions and use of safe libraries**
  - There have been a number of proposals to augment compilers to automatically insert range checks on pointer references
  - Libsafe is an example that implements the standard semantics but includes additional checks to ensure that the copy operations do not extend beyond the local variable space in the stack frame

- **Safe coding techniques**
  - Programmers need to inspect the code and rewrite any unsafe coding constructs
  - An example is the OpenBSD project which produces a free, multiplatform 4.4BSD-based UNIX-like operating system
  - Among other technology changes, programmers have under-taken an extensive audit of the existing code base, including the operating system, standard libraries, and common utilities

- **Stack protection mechanisms**
  - An effective method for protecting programs against classic stack overflow attacks is to instrument the function entry and exit code to set up and then check its stack frame for any evidence of corruption
  - Stackguard, one of the best-known protection mechanisms, is a GNU Compile Collection (GCC) compiler extension that inserts additional function entry and exit code
  - Stack Canaries

Runtime Techniques

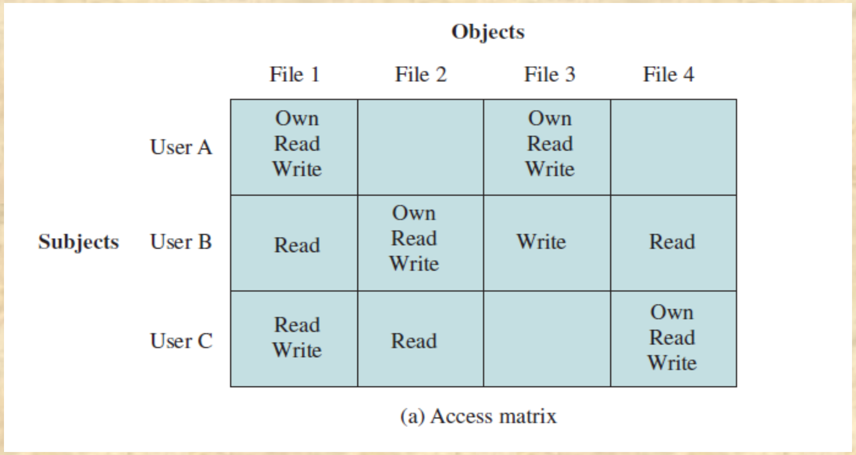
- **Executable address space protection**
  - A possible defense is to block the execution of code on the stack, on the assumption that executable code should only be found elsewhere in the processes address space
  - Extensions have been made available to Linux, BSD, and other UNIX-style systems to support the addition of the no-execute bit
  - Some hardware support from the processors MMU is needed. (tag pages that non-executable)

- **Address space randomization**
  - A runtime technique that can be used to thwart attacks involves manipulation of the location of key data structures in the address space of a process
  - Moving the stack memory region around by a megabyte or so has minimal impact on most programs but makes predicting the targeted buffer's address almost impossible
  - Another technique is to use a security extension that randomizes the order of loading standard libraries by a program and their virtual memory address locations

- **Guard pages**
  - Caps are placed between the ranges of addresses used for each of the components of the address space
  - These gaps, or guard pages, are flagged in the MMU as illegal addresses and any attempt to access them results in the process being aborted.
  - Accessing them results in a segmentation fault.
  - A further extension places guard pages between stack frames or between different allocations on the heap

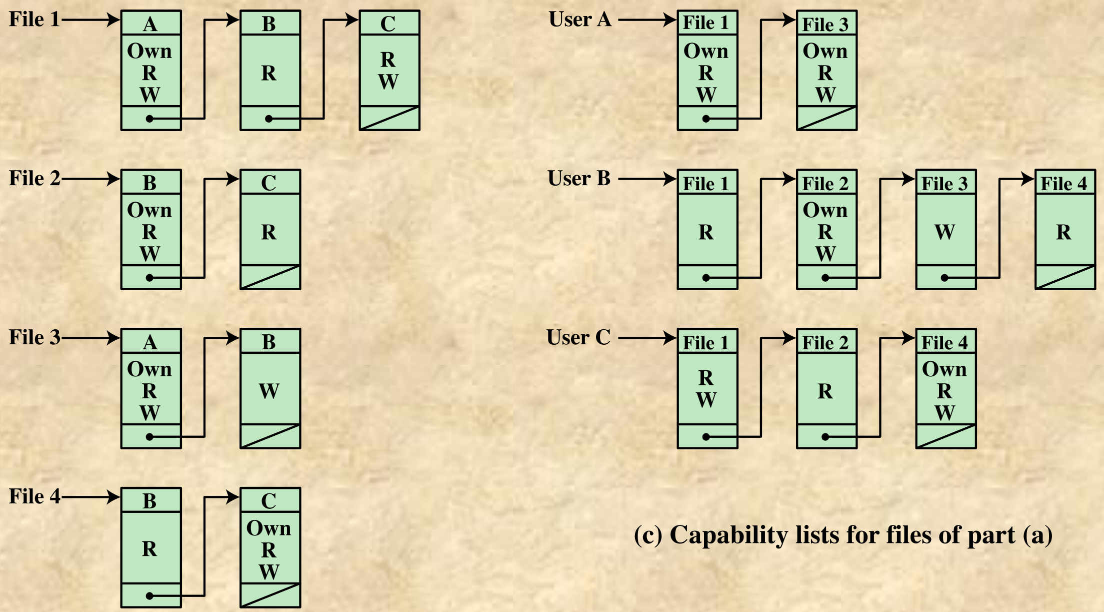
File System Access Control

- Identifies a user to the system
- Associated with each user there can be a profile that specifies permissible operations and file accesses
- The operating system can then enforce rules based on the user profile
- The database management system, however, must control access to specific records or even portions of records
- The database management system decision for access depends not only on the user's identity but also on the specific parts of the data being accessed and even on the information already divulged to the user



		Objects			
		File 1	File 2	File 3	File 4
Subjects	User A	Own Read Write		Own Read Write	
	User B	Read	Own Read Write	Write	Read
	User C	Read Write	Read		Own Read Write

(a) Access matrix



(b) Access control lists for files of part (a)

(c) Capability lists for files of part (a)

Figure 15.3 Example of Access Control Structures

Access Control Policies

- An access control policy dictates what types of access are permitted, under what circumstances, and by whom
- Access control policies are generally grouped into the following categories:
  - Discretionary access control (DAC)
    - Controls access based on the identity of the requestor and on access rules stating what requestors are allowed to do
  - Mandatory access control (MAC)
    - Controls access based on comparing security labels with security clearances
  - Role-based access control (RBAC)
    - Controls access based on the roles that users have within the system, and on rules stating what accesses are allowed to users in given roles
  - Attribute-based access control (ABAC)
    - Controls access based on attributes of the user, the resource to be accessed, and current environmental conditions

OBJECTS

		subjects			files		processes		disk drives	
		S_1	S_2	S_3	F_1	F_2	P_1	P_2	D_1	D_2
	S_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
SUBJECTS	S_2		control		write *	execute			owner	seek *
	S_3			control		write	stop			

* - copy flag set

Figure 15.4 Extended Access Control Matrix

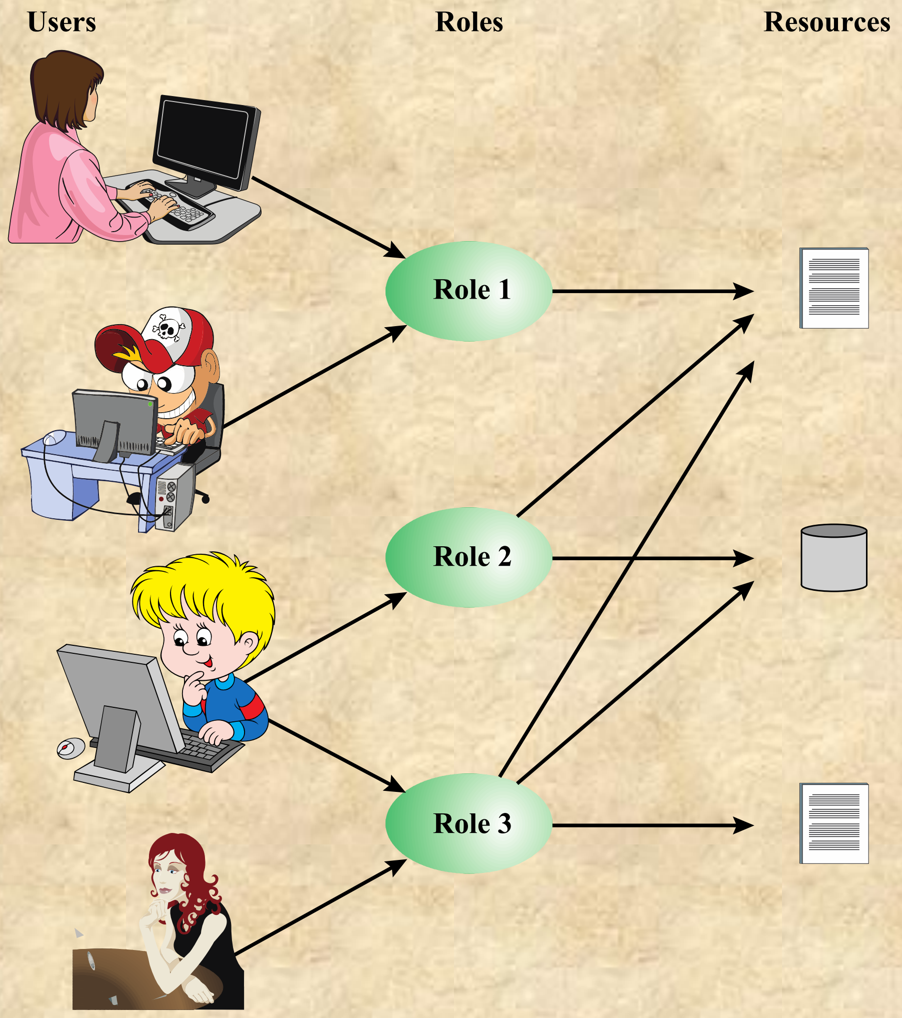


Figure 15.6 Users, Roles, and Resources

	R_1	R_2	• • •	R_n
U_1	✖			
U_2	✖			
U_3		✖		✖
U_4				✖
U_5				✖
U_6				✖
⋮				
U_m	✖			

		OBJECTS								
		R_1	R_2	R_n	F_1	F_1	P_1	P_2	D_1	D_2
ROLES	R_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
	R_2		control		write *	execute			owner	seek *
	⋮									
	R_n			control		write	stop			

Figure 15.7 Access Control Matrix Representation of RBAC