Operating Systems: Internals and Design Principles

Chapter 15 Operating System Security

Eighth 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

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

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

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

- 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
  - network-based IDS
    - monitors network traffic for particular network segments or devices and analyzes network, transport, and application protocols to identify suspicious activity

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
  - verification step
    - 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

Design goals:

1) The firewall acts as a choke point, so that all incoming traffic and all outgoing traffic must pass through the firewall

2) The firewall enforces the local security policy, which defines the traffic that is authorized to pass

3) The firewall is secure against attacks

- Can be an effective means of protecting a local system or network of systems from network-based security threats while affording access to the outside world via wide area networks and the Internet

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

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

Defenses

Compile-time

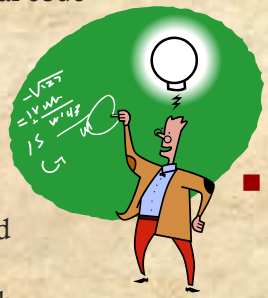
- Aim to prevent or detect buffer overflows by instrumenting programs when they are compiled
- Possibilities:
  - choose a high-level language that does not permit buffer overflows
  - encourage safe coding standards
  - use safe standard libraries
  - include additional code to detect corruption of the stack frame

Runtime

- Can be deployed in operating systems and updates and can provide some protection for existing vulnerable programs
- These defenses involve changes to the memory management of the virtual address space of processes
  - these changes act either to:
    - alter the properties of regions of memory
    - or to make predicting the location of targeted buffers sufficiently difficult to thwart many types of attacks

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



- **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

- **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

- **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

Runtime Techniques

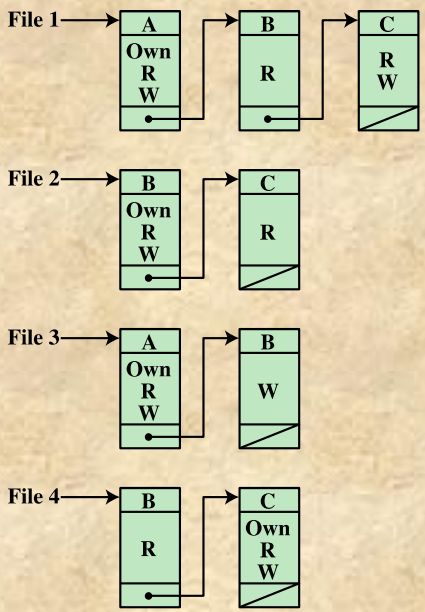
- **Executable address space protection**
  - a possible defense is to block the execution of code on the stack, on the assumption that executable code should only be found elsewhere in the processes address space
  - extensions have been made available to Linux, BSD, and other UNIX-style systems to support the addition of the no-execute bit
- **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
  - a further extension places guard pages between stack frames or between different allocations on the heap

File System Access Control

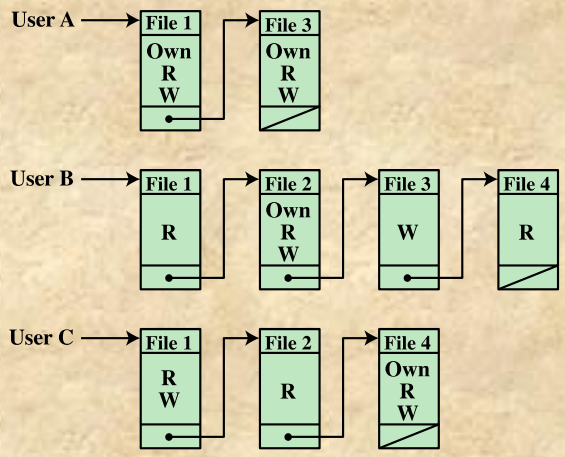
- Identifies a user to the system
- Associated with each user there can be a profile that specifies permissible operations and file accesses
- The operating system can then enforce rules based on the user profile
- The database management system, however, must control access to specific records or even 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

	File 1	File 2	File 3	File 4	Account 1	Account 2
User A	Own R W		Own R W		Inquiry Credit	
User B	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
User C	R W	R		Own R W		Inquiry Debit

(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

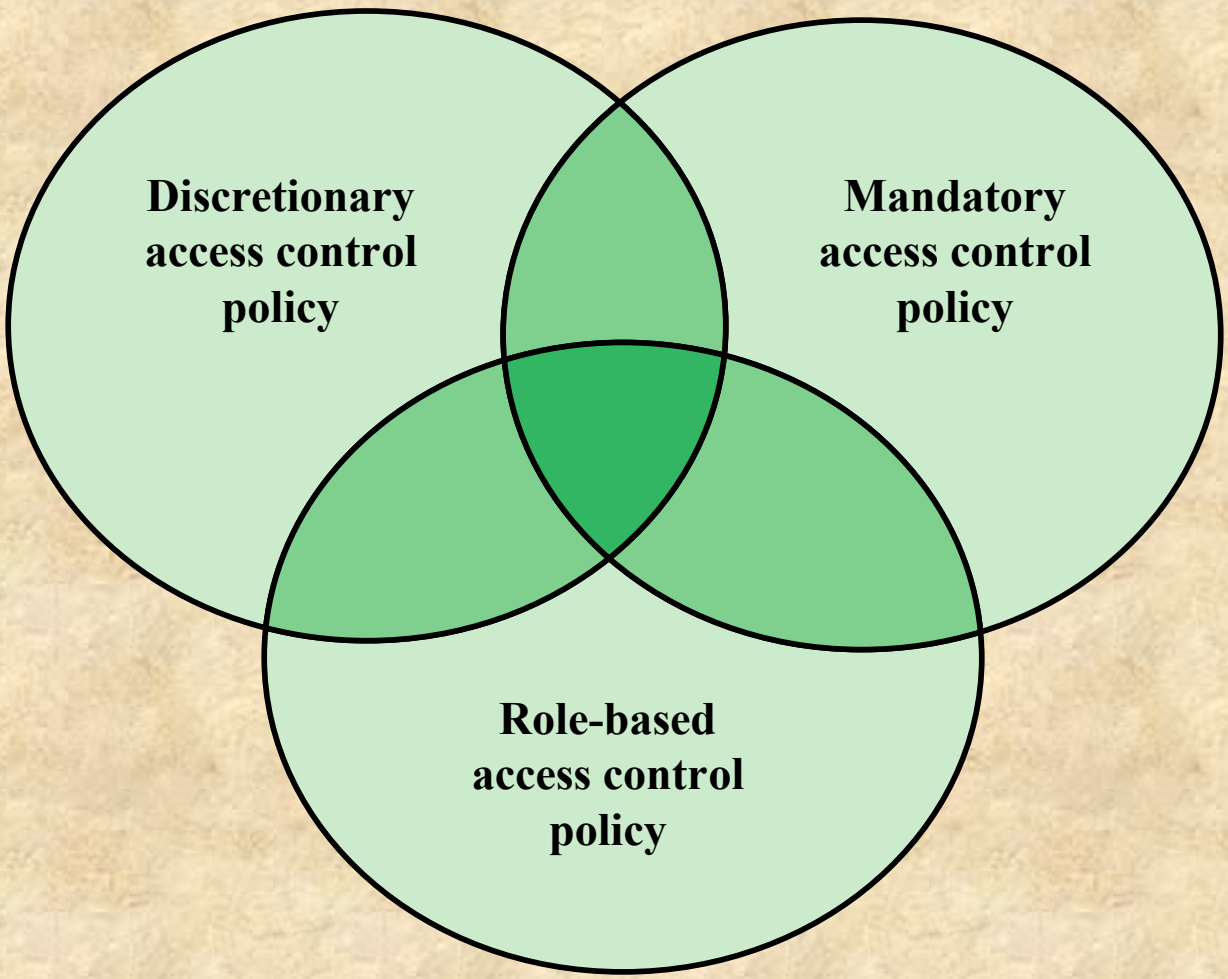


Figure 15.4 Access Control Policies

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

* - copy flag set

Figure 15.5 Extended Access Control Matrix

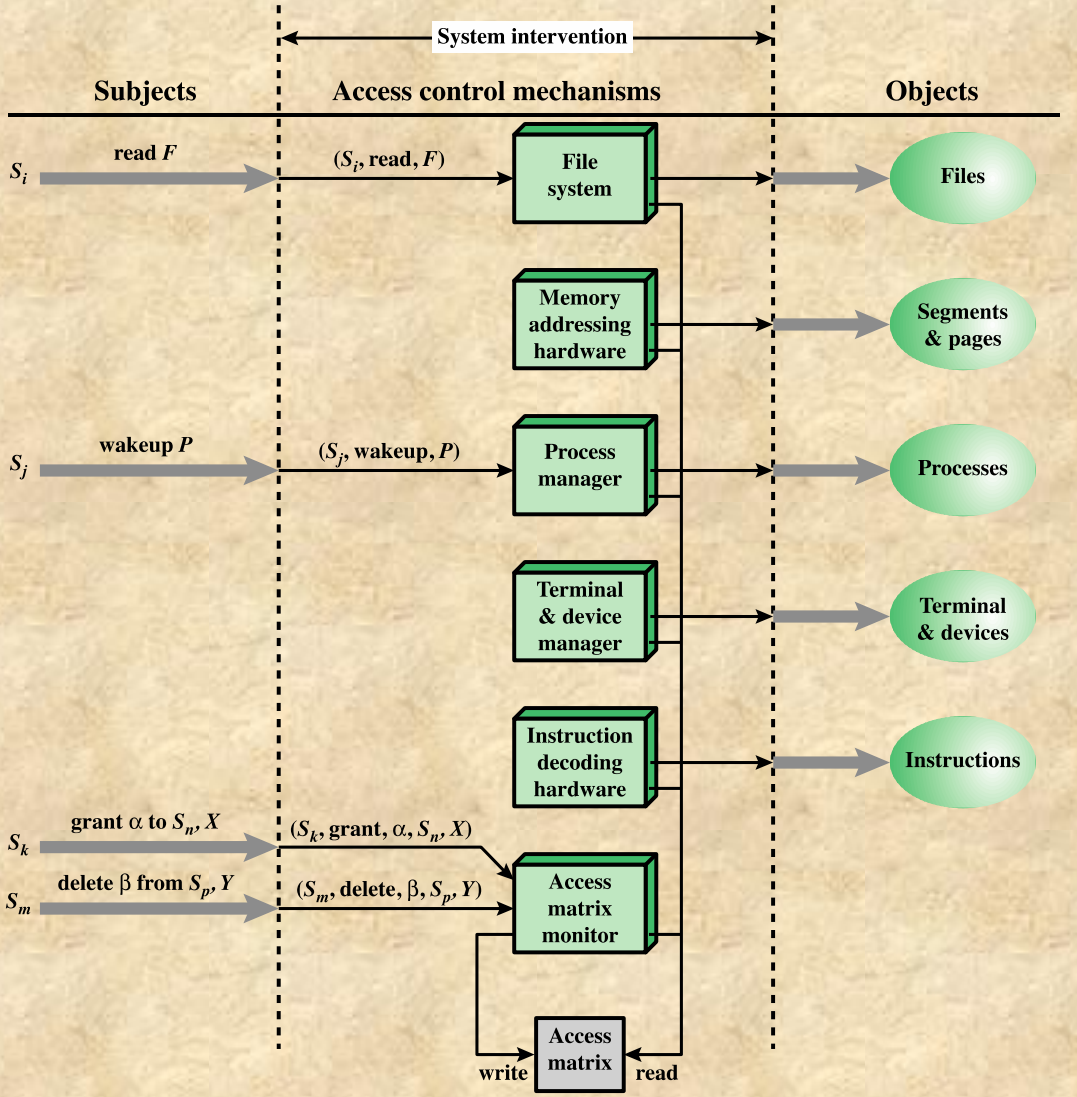


Figure 15.6 An Organization of the Access Control Function

Table 15.1

Access Control System Commands

Rule	Command (by S_o)	Authorization	Operation
R1	**transfer** $\left\{ \begin{matrix} \alpha^* \\ \alpha \end{matrix} \right\}$ **to** S, X	'α^*' in $A[S_o, X]$	store $\left\{ \begin{matrix} \alpha^* \\ \alpha \end{matrix} \right\}$ in $A[S, X]$
R2	**grant** $\left\{ \begin{matrix} \alpha^* \\ \alpha \end{matrix} \right\}$ **to** S, X	'owner' in $A[S_o, X]$	store $\left\{ \begin{matrix} \alpha^* \\ \alpha \end{matrix} \right\}$ in $A[S, X]$
R3	**delete** α **from** S, X	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	delete α from $A[S, X]$
R4	$w \leftarrow$ **read** S, X	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	copy $A[S, X]$ into w
R5	**create object** X	None	add column for X to A; store 'owner' in $A[S_o, X]$
R6	**destroy object** X	'owner' in $A[S_o, X]$	delete column for X from A
R7	**create subject** S	none	add row for S to A; execute **create object** S; store 'control' in $A[S, S]$
R8	**destroy subject** S	'owner' in $A[S_o, S]$	delete row for S from A; execute **destroy object** S

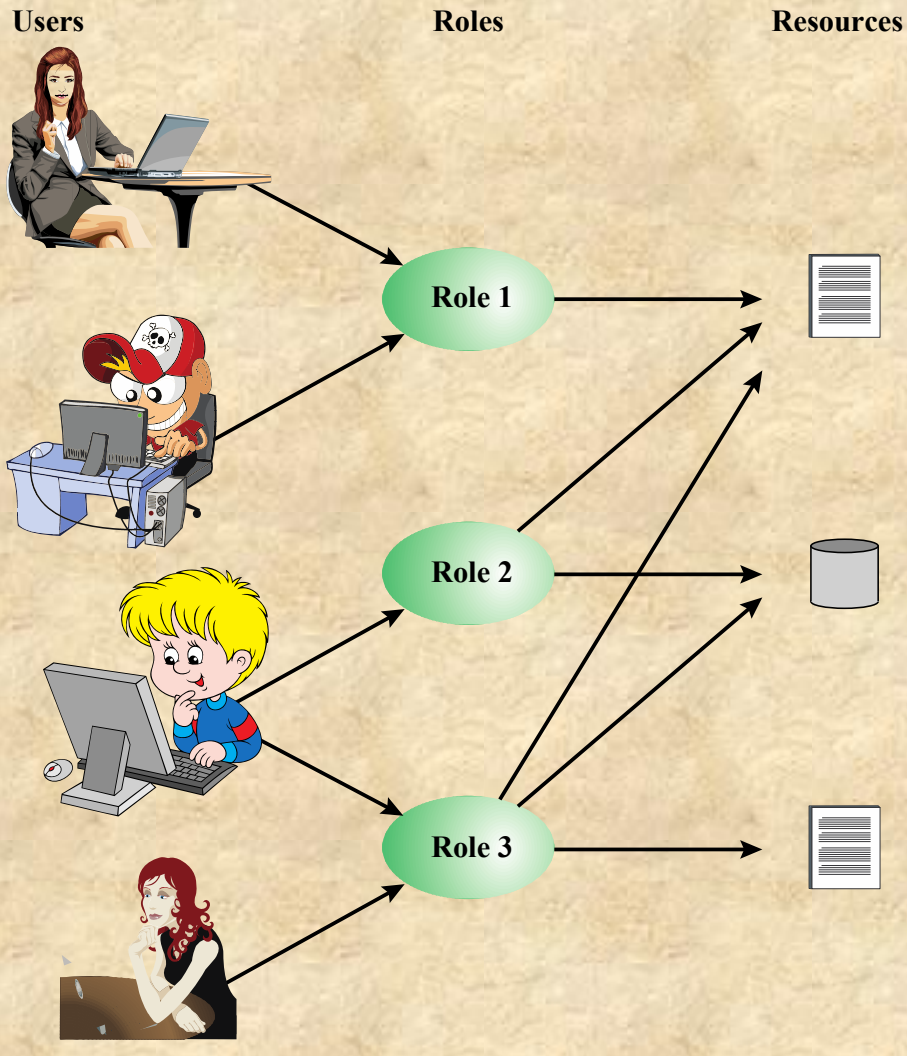


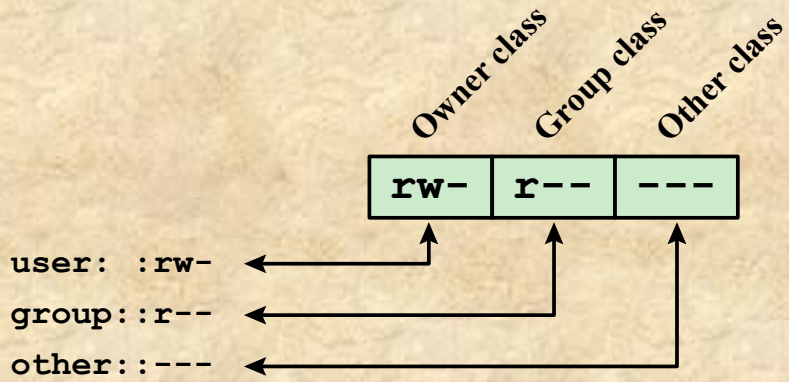
Figure 15.7 Users, Roles, and Resources

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

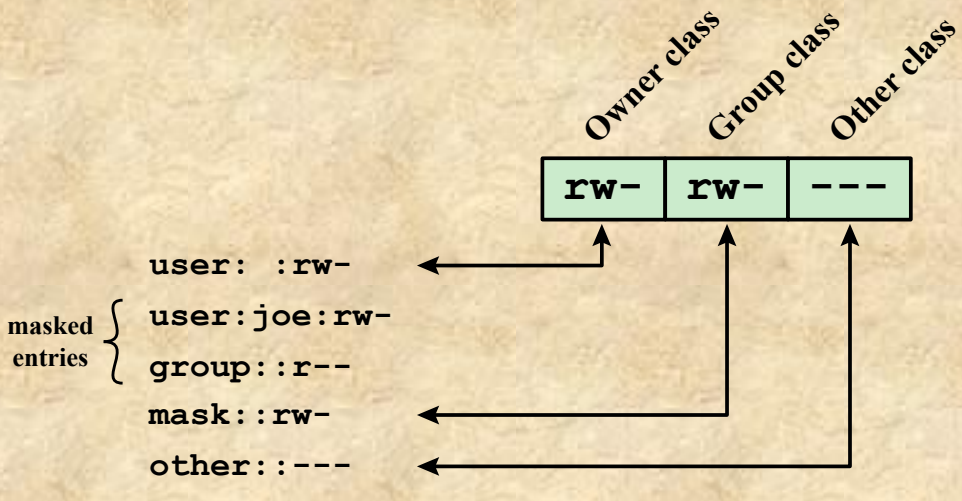
OBJECTS

ROLES	R_1	R_2	R_n	F_1	F_1	P_1	P_2	D_1	D_2
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 15.8 Access Control Matrix Representation of RBAC



(a) Traditional UNIX approach (minimal access control list)



(b) Extended access control list

Figure 15.9 UNIX File Access Control

Operating Systems Hardening

- Basic steps to use to secure an operating system:
  - Install and patch the operating system
  - Harden and configure the operating system to adequately address the identified security needs of the system by:
    - removing unnecessary services, applications, and protocols
    - configuring users, groups and permissions
    - configuring resource controls
  - Install and configure additional security controls, such as antivirus, host-based firewalls, and intrusion detection systems (IDS), if needed
  - Test the security of the basic operating system to ensure that the steps taken adequately address its security needs

Operating System Installation: Initial Setup and Patching

System security begins with the installation of the operating system

Ideally new systems should be constructed on a protected network

The initial installation should comprise the minimum necessary for the desired system, with additional software packages included only if they are required for the function of the system

The overall boot process must also be secured

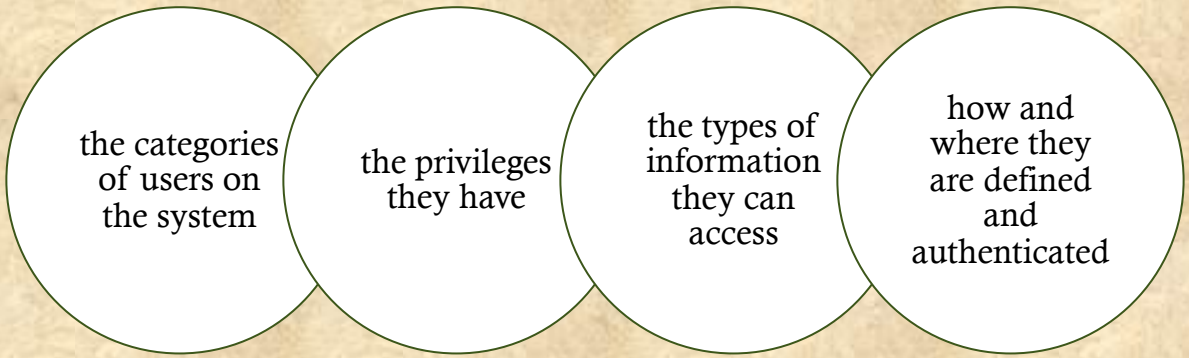
Care is also required with the selection and installation of any additional device driver code, since this executes with full kernel level privileges, but is often supplied by a third party

Remove Unnecessary Services, Applications, and Protocols

- The system planning process should identify what is actually required for a given system so that a suitable level of functionality is provided, while eliminating software that is not required to improve security
- When performing the initial installation the supplied defaults should not be used, but rather the installation should be customized so that only the required packages are installed
- Many of the security-hardening guides provide lists of services, applications, and protocols that should not be installed if not required
- Strong preference is stated for not installing unwanted software, rather than installing and then later removing or disabling it as many uninstall scripts fail to completely remove all components of a package
  - should an attacker succeed in gaining some access to a system, disabled software could be re-enabled and used to further compromise a system
  - it is better for security if unwanted software is not installed, and thus not available for use at all

Configure Users, Groups, and Authentication

The system planning process should consider:



- Restrict elevated privileges to only those users that require them
- At this stage any default accounts included as part of the system installation should be secured
- Those accounts which are not required should be either removed or at least disabled
- System accounts that manage services on the system should be set so they cannot be used for interactive logins
- Any passwords installed by default should be changed to new values with appropriate security
- Any policy that applies to authentication credentials and to password security is configured

Configure Resource Controls

- Once the users and their associated groups are defined, appropriate permissions can be set on data and resources to match the specified policy
- This may be to limit which users can execute some programs or to limit which users can read or write data in certain directory trees
- Many of the security-hardening guides provide lists of recommended changes to the default access configuration to improve security

Install Additional Security Controls

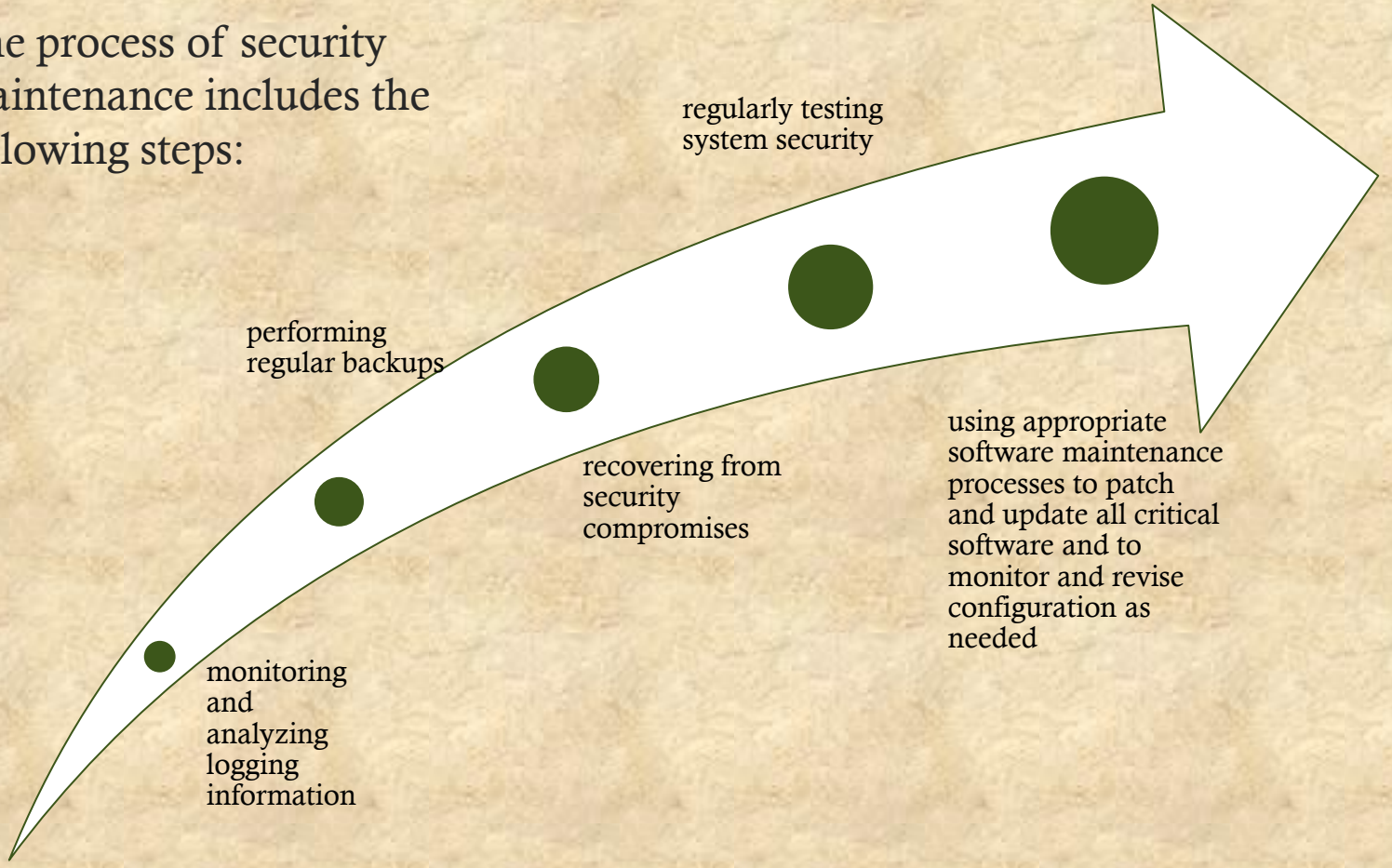
- Further security improvement may be possible by installing and configuring additional security tools such as antivirus software, host-based firewall, IDS or IPS software, or application white-listing
- Some of these may be supplied as part of the operating systems installation, but not configured and enabled by default
- Given the wide-spread prevalence of malware, appropriate antivirus is a critical security component
- IDS and IPS software may include additional mechanisms such as traffic monitoring or file integrity checking to identify and even respond to some types of attack
- White-listing applications limits the programs that can execute in the the system to just those in an explicit list

Test the System Security

- The final step in the process of initially securing the base operating system is security testing
- The goal is to ensure that the previous security configuration steps are correctly implemented and to identify any possible vulnerabilities that must be corrected or managed
- Suitable checklists are included in many security-hardening guides
- There are also programs specifically designed to review a system to ensure that a system meets the basic security requirements and to scan for known vulnerabilities and poor configuration practices
- This should be done following the initial hardening of the system and then repeated periodically as part of the security maintenance process

Security Maintenance

- The process of security maintenance includes the following steps:



Logging

- Effective logging helps ensure that in the event of a system breach or failure, system administrators can more quickly and accurately identify what happened and more effectively focus their remediation and recovery efforts
- Logging information can be generated by the system, network, and applications
- The range of logging data acquired should be determined during the system planning stage
- Logging can generate significant volumes of information so it is important that sufficient space is allocated for them
- A suitable automatic log rotation and archive system should be configured to assist in managing the overall size of the logging information
- Some form of automated analysis is preferred as it is more likely to identify abnormal activity
  - manual analysis of logs is tedious and is not a reliable means of detecting adverse events

Data Backup and Archive

- Performing regular backups of data on a system is another critical control that assists with maintaining the integrity of the system and user data
- The needs and policy relating to backup and archive should be determined during the system planning stage
  - key decisions include whether the copies should be kept online or offline and whether copies should be stored locally or transported to a remote site
- Backup
  - the process of making copies of data at regular intervals, allowing the recovery of lost or corrupted data over relatively short time periods of a few hours to some weeks
- Archive
  - the process of retaining copies of data over extended periods of time, being months or years, in order to meet legal and operational requirements to access past data

Access Control Scheme

- When a user logs on to a Windows system a name/password scheme is used to authenticate the user
- If the logon is accepted a process is created for the user and an access token is associated with that process object
  - the access token includes a security ID (SID) which is the identifier by which this user is known to the the system for purposes of security
  - the token also contains SIDs for the security groups to which the user belongs

The access token serves two purposes:

- it keeps all necessary security information together to speed access validation
- it allows each process to modify its security characteristics in limited ways without affecting other processes running on behalf of the user

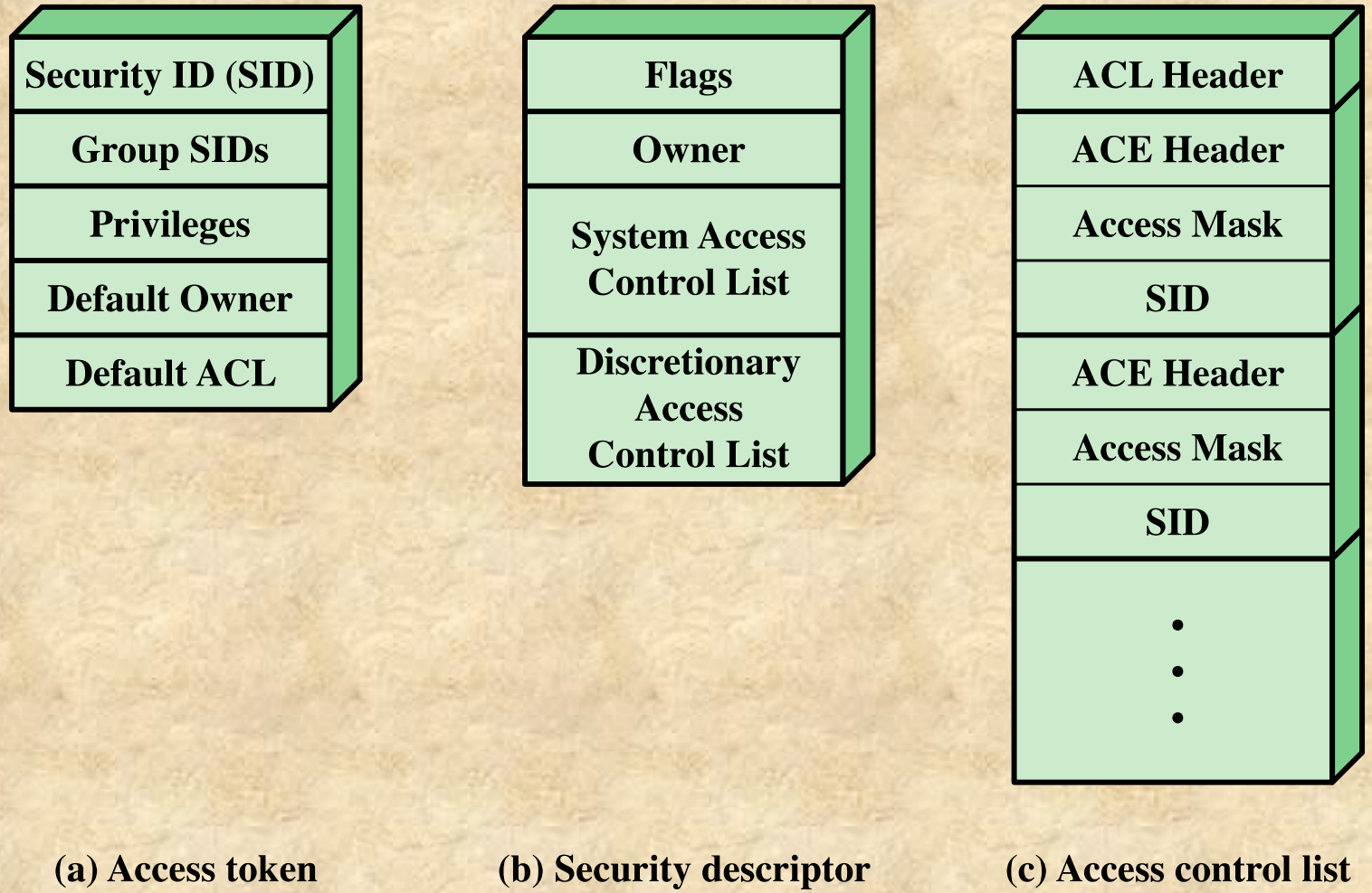


Figure 15.10 Windows Security Structures

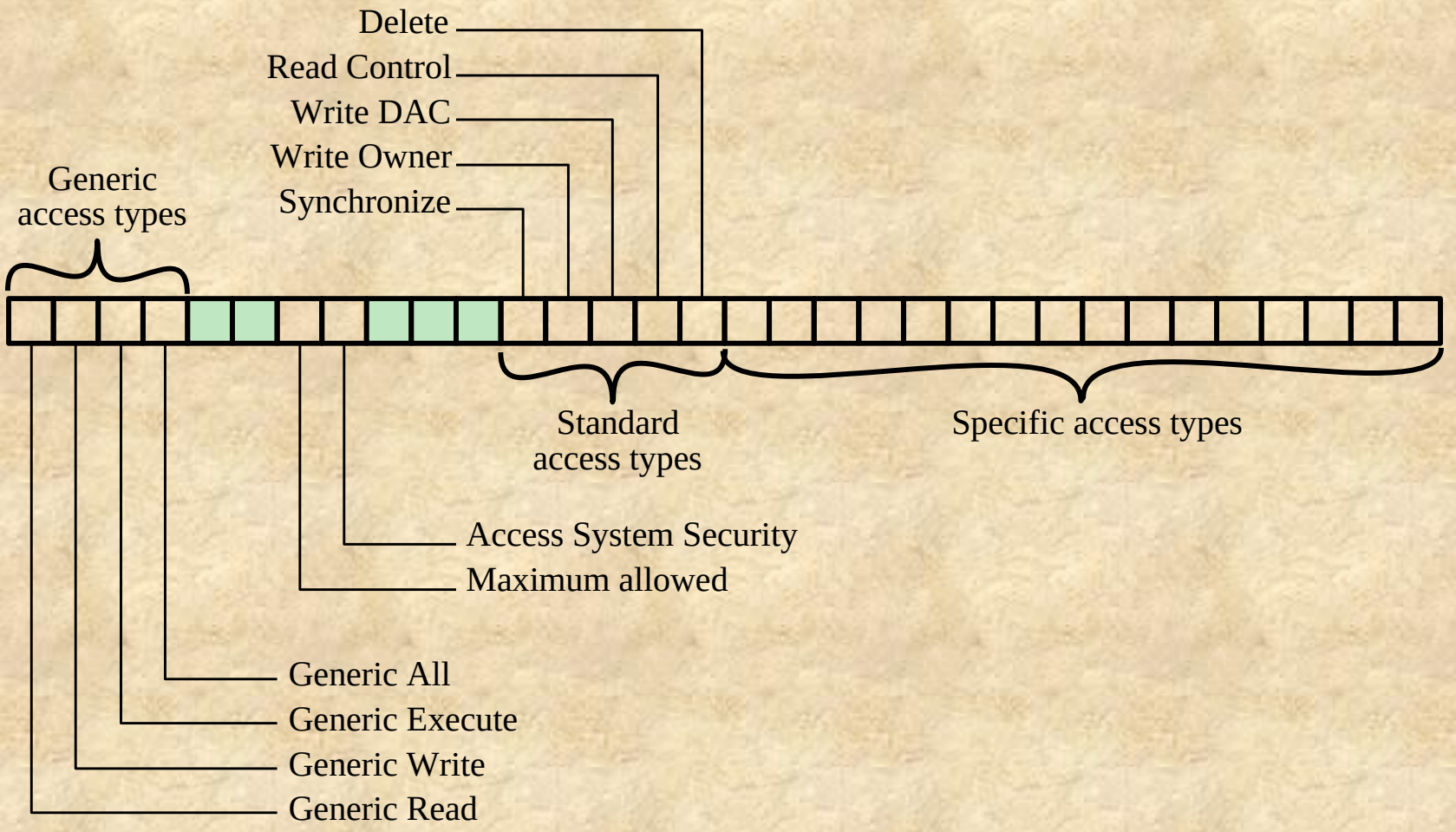


Figure 15.11 Access Mask

Summary

- Intruders and malicious software
  - system access threats
  - countermeasures
- Buffer overflow
  - buffer overflow attacks
  - compile time defenses
  - runtime defenses
- Access control
  - file system access control
  - access control policies
- UNIX access control
  - traditional UNIX file access control
  - access control lists in UNIX
- Operating systems hardening
  - OS installation: initial setup and patching
  - remove unnecessary services, application, and protocols
  - configure users, groups and authentication
  - install additional security controls
  - test the system security
- Security maintenance
  - logging
  - data backup and archive
- Windows security
  - access control scheme
  - access token
  - security descriptors