Operating System Security
CS 239
Computer Security
February 16, 2005

Outline

- Introduction
- Memory protection
- Interprocess communications protection
- File protection

Introduction

- Operating systems provide the lowest layer of software visible to users
- Operating systems are close to the hardware
  - Often have complete hardware access
- If the operating system isn't protected, the machine isn't protected
- Flaws in the OS generally compromise all security at higher levels

Why Is OS Security So Important?

- The OS controls access to application memory
- The OS controls scheduling of the processor
- The OS ensures that users receive the resources they ask for
- If the OS isn't doing these things securely, practically anything can go wrong
- So almost all other security systems must assume a secure OS at the bottom

Single User Vs. Multiple User Machines

- The majority of today's computers usually support a single user
  - Sometimes one at a time, sometimes only one ever
- Some computers are still multi-user
  - Mainframes
  - Servers
  - Network-of-workstation machines
- Single user machines often run multiple processes, though

Server Machines Vs. General Purpose Machines

- Most server machines provide only limited services
  - Web page access
  - File access
  - DNS lookup
- Security problems are simpler for them
- Some machines still provide completely general service, though
- And many server machines <u>can</u> run general services . . .

Downloadable Code and Single User Machines

- Applets and other downloaded code should run in a constrained mode
- Using access control on a finer granularity than the user
- Essentially the same protection problem as multiple users

Mechanisms for Secure Operating Systems

- Most operating system security is based on separation
  - Keep the bad guys away from the good stuff
  - Since you don't know who's bad, separate most things

Separation Methods

- Physical separation
  - Different machines
- Temporal separation
  - Same machine, different times
- Logical separation
  - HW/software enforcement
- Cryptographic separation

The Problem of Sharing

- Separating stuff is actually pretty easy
- The hard problem is allowing controlled sharing
- How can the OS allow users to share exactly what they intend to share?
  - In exactly the ways they intend

Levels of Sharing Protection

- None
- Isolation
- All or nothing
- Access limitations
- Limited use of an object

Protecting Memory

- Most general purpose systems provide some memory protection
  - Logical separation of processes that run concurrently
- Usually through virtual memory methods
- Originally arose mostly for error containment, not security

Security Aspects of Paging

- Main memory is divided into page frames
- Every process has an address space divided into logical pages
- For a process to use a page, it must reside in a page frame
- If multiple processes are running, how do we protect their frames?

Protection of Pages

- Each process is given a page table
  - Translation of logical addresses into physical locations
- All addressing goes through page table
  - At unavoidable hardware level
- If the OS is careful about filling in the page tables, a process can't even name other processes' pages

Security Issues of Page Frame Reuse

- A common set of page frames is shared by all processes
- The OS switches ownership of page frames as necessary
- When a process acquires a new page frame, it used to belong to another process
  - Can the new process read the old data?

Special Interfaces to Memory

- Some systems provide a special interface to memory
- If the interface accesses physical memory,
  - And doesn't go through page table protections,
  - Attackers can read the physical memory
  - Then figure out what's there and find what they're looking for

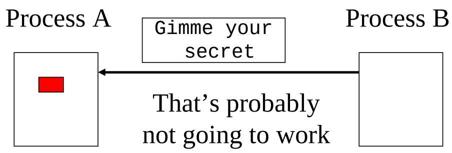
Protecting Interprocess Communications

- Operating systems provide various kinds of interprocess communications
  - Messages
  - Semaphores
  - Shared memory
  - Sockets
- How can we be sure they're used properly?

IPC Protection Issues

- How hard it is depends on what you're worried about
- For the moment, let's say we're worried about one process improperly using IPC to get info from another
  - Process A wants to steal information from process B
- How would process A do that?

Message Security

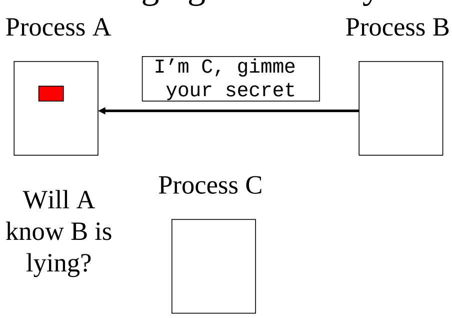


Can process B use message-based IPC to steal the secret?

How Can B Get the Secret?

- He can convince the system he's A
  - A problem for authentication
- He can break into A's memory
  - That doesn't use message IPC
  - And is handled by page tables
- He can forge a message from someone else to get the secret
- He can "eavesdrop" on someone else who gets the secret

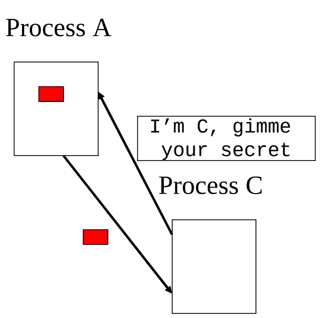
Forging An Identity



Operating System Protections

- The operating system knows who each process belongs to
- It can tag the message with the identity of the sender
- If the receiver cares, he can know the identity

How About Eavesdropping?



Can process B "listen in" on this message?

What's Really Going on Here?

- On a single machine, what is a message send, really?
- A message is copied from a process buffer to an OS buffer
  - Then from the OS buffer to another process' buffer
- If attacker can't get at processes' internal buffers and can't get at OS buffers, he can't "eavesdrop"

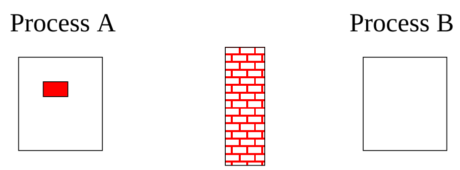
Other Forms of IPC

- Semaphores, sockets, shared memory, RPC
- Pretty much all the same
  - Use system calls for access
  - Which belong to some process
  - Which belongs to some principal
  - OS can check principal against access control permissions at syscall time

So When Is It Hard?

- Always possible that there’s a bug in the operating system
  - Allowing masquerading, eavesdropping, etc.
  - Or, if the OS itself is compromised, all bets are off
- What if the OS has to prevent cooperating processes from sharing information?

The Hard Case



Process A wants to tell the secret to process B
But the OS has been instructed to prevent that
Can the OS prevent A and B from colluding to get the secret to B?

Dangers for Operating System Security

- Bugs in the OS
  - Not checking security, allowing access to protected resources, etc.
- Privileged users and roles
  - Superusers often can do anything
- Untrusted applications and overly broad security domains

File Protection

- How do we apply these access protection mechanisms to a real system resource?
- Files are a common example of a typically shared resource
- If an OS supports multiple users, it needs to address the question of file protection

Unix File Protection

- A model for protecting files developed in the 1970s
- Still in very wide use today
  - With relatively few modifications
- But not very flexible

Unix File Protection Philosophy

- Essentially, Unix uses a limited ACL
- Only three subjects per file
  - Owner
  - Group
  - Other
- Limited set of rights specifiable
  - Read, write, execute
  - Special meanings for some file types

Unix Groups

- A set of Unix users can be joined into a group
- All users in that group receive common privileges
  - Except file owners always get the owner privileges
- A user can be in multiple groups
- But a file has only one group

Setuid and Setgid

- Unix mechanisms for changing your user identity and group identity
- Either indefinitely or for the run of a single program
- Created to deal with inflexibilities of the Unix access control model
- But the source of endless security problems

Why Are Setuid Programs Necessary?

- The print queue is essentially a file
- Someone must own that file
- How will other people put stuff in the print queue?
  - Without making the print queue writeable for all purposes
- Typical Unix answer is run the printing program setuid
  - To the owner of the print queue

Why Are Setuid Programs Dangerous?

- Essentially, setuid programs expand a user's security domain
- In an encapsulated way
  - Abilities of the program limit the operations in that domain
- Need to be damn sure that the program's abilities are limited

Some Examples of Setuid Dangers

- Setuid programs that allow forking of a new shell
- Setuid programs with powerful debugging modes
- Setuid programs with "interesting" side effects
  - E.g., `lpr` options that allow file deletion

Domain and Type Enforcement

- A limited version of capabilities
- Meant to address the dangers of setuid
- Allows system to specify security domains
  - E.g., the printing domain
- And to specify data types
  - E.g., the printer type

Using DTE

- Processes belong to some domain
  - Can change domains, under careful restrictions
- Only types available to that domain are accessible
  - And only in ways specified for that domain

A DTE Example

- Protecting the FTP daemon from buffer overflow attacks
- Create an FTP domain
- Only the FTP daemon and files in the FTP directory can be executed in this domain
  - And these executables may not be written within this domain
- Executing the FTP daemon program automatically enters this domain

What Happens On Buffer Overflow?

- The buffer overflow attack allows the attacker to request execution of an arbitrary program
  - Say, `/bin/sh`
- But the overflowed FTP daemon program was in the FTP domain
  - And still is
- `/bin/sh` is of a type not executable from this domain
  - So the buffer overflow can't fork a shell

Unix File Access Control and Complete Mediation

- Unix doesn't offer complete mediation
- File access is checked on open to a file
  - For the requested modes of access
- Opening program can use the file in the open mode for as long as it wants
  - Even if the file's access permissions change
- Substantially cheaper in performance

Physical Implementation of Unix Access Control

- Effectively, requires 9 bits per file
  - Setuid and setgid adds two bits
- Stored in the file's inode
  - Possible because they're so small
- Checking them again requires re-examining the inode

Pros and Cons of Unix File Protection Model

+ Low cost
+ Simple and easy to understand
+ Time tested

– Lacking in flexibility
  - In granularity of control
    - Subject and object
  - In what controls are possible

– No complete mediation

Access Control Lists for File Systems

- The file system access control mechanism of choice in modern operating systems
- Used in many systems -
  - Andrew
  - Windows NT/2000/XP
  - Solaris 2.5 and higher

Windows NT ACLs for Files

- Integrated into the overall NT access control mechanism
- Uses NT concept of security descriptors
  - Specifying objects
- And security IDs
  - Specifying subjects

More On Windows NT File ACLs

- The NT model also allows creation of groups
  - With their own security IDs
- The security model is object-based
  - So the types of permissions that can be granted are flexible and extensible