Operating System Security
CS 239
Security for Networks and System Software
May 20, 2002

Outline

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

Introduction

- Threats to single machines are of the same character as threats to network communications
- But very different in their mechanisms and solutions

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

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

Embedded Systems

- An increasingly large number of objects contain embedded computers
  - With limited capabilities and access
- The future will undoubtedly see security problems for them
  - First for embedded processors in security systems, probably

Downloadable Code and Single User Machines

- Applets and other downloaded code should run in a limited 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

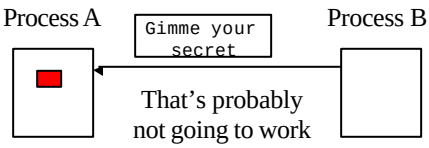
Protecting Interprocess Communications

- 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

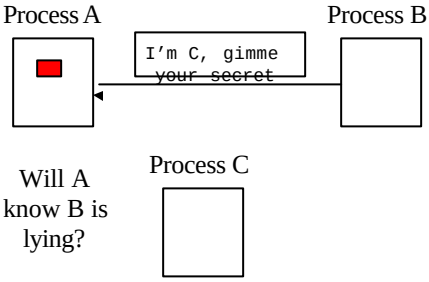


Can process B use message-based IPC to steal the secret?

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

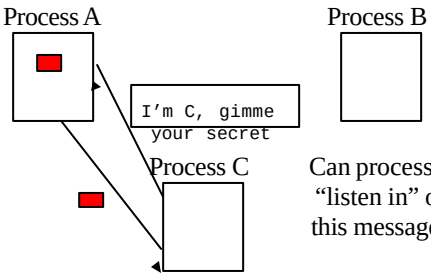


Will A know B is lying?

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

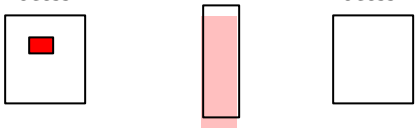
- Semaphores, sockets, shared memory, RPC
- Pretty much all the same
  - Need system call to perform them
  - System call to get access belongs to some process
  - Process belongs to some principal
  - OS can check principal against access control permissions at syscall time

So When's It Hard?

- What if the OS has to prevent cooperating processes from sharing information?

The Hard Case

Process A Process B



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?

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 for a long time 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

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

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
  - Solaris 2.5

Solaris 2.5 ACLs for Files

- In addition to the standard Unix permissions
- Allows ACL-style listing of users and groups
  - With separate permissions for each
- Does not expand set of allowable permissions

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

Authentication in Single Machine Systems

- Most single machine system security mechanisms are based on controlling access
- Access control only works if you have good authentication
- Various means are used to provide authentication in operating systems

Process Authentication

- Memory protection is based on process identity
  - Only the owning process can name its own virtual memory pages
- Because VM is completely in OS control, pretty easy to ensure that processes can't fake identities

How the OS Authenticates Processes

- System calls are issued by a particular process
- The OS securely ties a process control block to the process
  - Not under user control
- Thus, the ID in the process control block can be trusted

How Do Processes Originally Obtain Access Permission?

- Most OS resources need access control based on user identity or role
  - Other than virtual memory pages and other transient resources
- How does a process get properly tagged with its owning user or role?
- Security is worthless if OS carefully controls access on a bogus user ID

Users and Roles

- In most systems, OS assigns each potential user an ID
- More sophisticated systems recognize that the same user works in different *roles*
  - Effectively, each role requires its own ID
  - And secure methods of setting roles

Securely Identifying Users and Roles

- Passwords
- Identification devices
- Challenge/response systems
- Physical verification of the user

Passwords

- Authentication by what you know
- One of the oldest and most commonly used security mechanisms
- Authenticate the user by requiring him to produce a secret
  - Known only to him and to the authenticator
  - Or, if one-way encryption used, known only to him

Problems With Passwords

- They have to be unguessable
  - Yet easy for people to remember
- If networks connect terminals to computers, susceptible to password sniffers
- Unless fairly long, brute force attacks often work on them

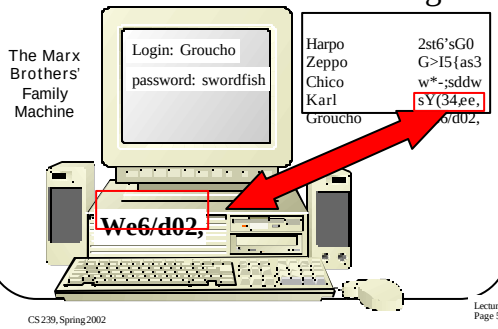
Proper Use of Passwords

- Select good ones
- Change them often
- Never write them down
- Never tell anyone else

Handling Passwords

- The OS must be able to check passwords when users log in
- So must the OS store passwords?
- Not really
  - It can store an encrypted version
- Encrypt the offered password and compare it to the stored version

Standard Password Handling



Is Encrypting the Password File Enough?

- What if an attacker gets a copy of your password file?
- No problem, the passwords are encrypted
  - Right?
- Yes, but . . .

Dictionary Attacks on the Encrypted Password File

Now you can hack the Communist Manifesto!

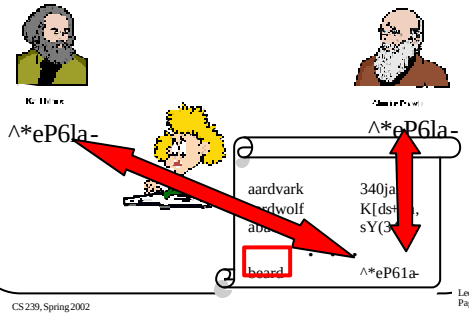
sY(34,ee

Rats!!!!

A Serious Issue

- All Linux machines use the same one-way function to encrypt passwords
- If someone runs the entire dictionary through that function,
  - Will they have a complete list of all encrypted dictionary passwords?

Illustrating the Problem



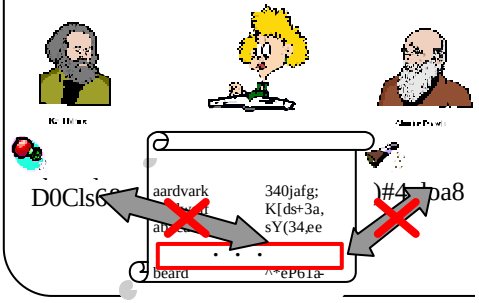
The Real Problem

- Not that Darwin and Marx chose the same password
- But that anyone who chose that password got the same encrypted result
- So the attacker need only encrypt every possible password once
- And then she has a complete dictionary usable against anyone

Salted Passwords

- Combine the plaintext password with a random number
  - Then run it through the one-way function
- The random number need not be secret
- It just has to be different for different users

Did It Fix Our Problem?



Identification Devices

- Authentication by what you have
- A smart card or other hardware device that is readable by the computer
- Authenticate by providing the device to the computer

Problems With Identification Devices

- If lost or stolen, you can't authenticate yourself
  - And someone else can
  - Often combined with passwords to avoid this problem
- Unless cleverly done, susceptible to sniffing attacks
- Requires special hardware

Challenge/Response Authentication

- Authentication by what questions you can answer correctly
- The system asks the user to provide some information
- If it's provided correctly, the user is authenticated

Differences From Passwords

- Challenge/response systems ask for different information every time
- Or at least the questions come from a large set
- Best security achieved by requiring what amounts to encryption of the challenge
  - But that requires special hardware
  - Essentially, a smart card

Problems With Authentication Through Challenge/Response

- Either the question is too hard to answer without special hardware
- Or the question is too easy for intruders to spoof
- Still, commonly used in real-world situations
  - E.g., authenticating you by asking your mother's maiden name

Authentication Through Physical Verification

- Authentication based on who you are
- Things like fingerprints, voice patterns, retinal patterns, etc.
- To authenticate to the system, let it measure the appropriate physical characteristics

Problems With Physical Verification

- Requires very special hardware
  - Possibly excepting systems that examine typing patterns
- May not be as foolproof as you think
- Many characteristics vary too much for practical use
- Generally not helpful for authenticating programs or roles