17: Protection/Security

Last Modified:
7/3/2004 1:48:28 PM

Protection

- Protecting processes/users from each other is one of the core OS responsibilities
- Control access of processes or users to resources of the computer system (HW and SW)
  - Ensure resources are operated on my only those processes that have gained proper authorization
  - Enforcing resource limits

Cross-cutting issue

- CPU Scheduling
  - Protection by timer interrupts and OS scheduling policy
- Process Management
  - Protection by access control and enforcement of resource limits (most OS?)
- Virtual Memory
  - Protection by inability to name other processes memory space
- File System
  - User defined access controls per file/directory
- Note: Synchronization more voluntary protection by observing rules within a set of processes/threads that share data (Monitors maybe protection?)

How to do protection?

- From that brief survey of OS topics it is clear that protection can be accomplished in many ways
  - Protection can be based on the design of the system which makes access impossible (can't even name things you shouldn't access)
    - E.g. VM
  - Protection can be controllable by an OS wide policy (OS controls resource allocation)
    - E.g. timer interrupts
  - Protection can be controlled by user definable access controls
    - E.g. User can set FS access controls
- Implies ability to deny authorized access! Ability to enforce the policy!

Principles

- Generally the more restrictive the system the more protection
- "Need to know" principle says only grant those rights absolutely necessary to accomplish a task
  - Start out granting none and see where it breaks, add the smallest new privileges as possible
  - Ex. If a process only needs to read/write one specific file then don't give it access to all the user's files
  - Ex. Don't give full root privileges just because need to open a port < 1024

Policy vs Mechanism

- Mechanism says "what types of access are possible" and "defines the means for identifying authorized vs unauthorized access"
- Policy says "which processes/users should have which kinds of access"
- When building system best to make mechanism match the problem domain rather than a parituclar desired policy
  - More flexible if separate mechanism from policy!
- Example: if your mechanism does not distinguish between read and execute rights then impossible to hand out one without the other; if mechanism does distinguish then policy may never choose to hand out one without the other but it could

Types of access

- The possible types of access depend on the resource
  - CPUs can be executed upon
  - File can be read/written/executed
  - Directories can be read/inserted into/deleted from/traversed without displaying all
  - Tape drives can be read/written/rewound
- Begin by thinking about all the possible actions you might want to allow/disallow on an object

-7

Protection Domain

- Once we determine all the possible resources in the system and all the possible types of access to those resources, the next is to think about all the possible entities to whom we would like to grant/deny rights
- Associate with each entity a "protection domain"
- Define a protection domain as a collection of access rights to specified objects

-8

Typical Domain Granularities

- One domain for OS; one domain for USER
- Domain per user
- Domain per process
- Domain per procedure
- ...

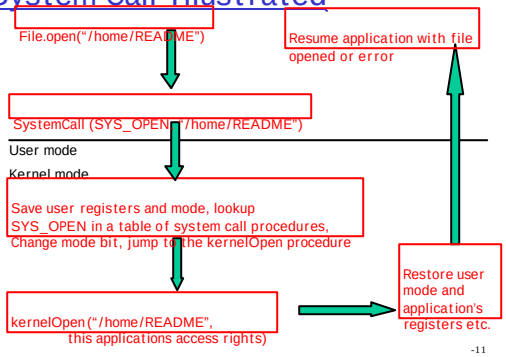
-9

Recall: Kernel/User Mode

- Hardware needs to be able to distinguish the OS from user apps
  - Controls ability to execute privileged instructions etc
- Most architectures have a "mode" value in a protected register
  - When user applications execute, the mode value is set to one thing
  - When the OS kernel executes, the mode value set to something else
  - If code running in user mode, an attempt to execute protected instructions will generate an exception
  - Switching the mode value must of course be protected

-10

System Call Illustrated



-11

Is Kernel/User distinction enough?

- Not if want to distinguish between users!
  - How can we distinguish between users?
- Is user the best thing to base domain on?
  - Do you want all processes you run to have your full privileges?
  - Do you ever need special privileges but not all of root access?

-12

Distinguishing users: Logging in

- When a user logs in, they supply a password which is checked against a password list
- In UNIX, passwords stored in file /etc/passwd
  - What is in this file?
- Naïve approach: file with everyone's password in it (but what if that file is compromised)
- Better: keep a file with hash(password)
  - One way hash function makes it hard to get from hash(password) to password but easy to go password to hash(password)
  - Now can distribute the password file in plain text and passwords not revealed

Other attacks?

- Dictionary attack?
  - Compile a list of common passwords (all English words for example) and compute hash(password) on all of them
  - Compare contents of password list to this dictionary list
- Solution? Salt
  - Password file entry = hash (salt+password)
  - Store salt in clear
  - Bad guy can't just use a pre-generated dictionary file – has to have a different one for each person's salt
  - UNIX uses a 12-bit salt
    - so need 2^{12} different dictionary files – one for each salt
    - Is 4096 times harder hard enough?

Better passwords?

- Words in English dictionary? 250,000
  - http://www.askoxford.com/asktheexperts/faq/aboutwords/numberwords
- Possible 8 character passwords if just letters: 52^8= 53,459,728,531,456
- If add digits: 62^8
- If add punctuation (32 punctuation characters??): 94^8

Distinguishing users (con't)

- Some systems allow other machines to vouch for the identify of a user
- Ex. Rsh/rcpy etc allow user to specify a list of users and machines allowed to act like them (without a password)
- Example: .rhosts says allow jnm @ * to log in as me
  - Then if there is an jnm account on any machine it can act like me
  - Even if is says jnm @ mymachine other machines can masquerade as mymachine
  - Bad stuff!

Logging in

- Recall: in last stages of boot process, OS creates a process called init
- Init does various important housecleaning activities including maintaining a process for each terminal port (tty)
- Getty then executes the login program on that tty
- Login gets username/password from user, reads /etc/password, computes hash(salt+password) and compares
- If login successful, login will spawn a shell process for the user
- Shell and all its children run with that user's privileges

User's processes

- OS will keep maintain memory protection (even amongst processes belonging to the same user)
- OS will also check file permissions for all files the process attempts to access/create
- More on file permissions later..

Root

- Root is just a special userId
- Can correspond to many user names in /etc/password, but any user with userId 0 is root
- OS gives processes with userId 0 special privileges e.g.:
  - Opening privileged ports
  - Reading/writing/executing all files
  - Becoming any other user
  - Exceeding the FS quotas (like FFS's 10% of reserve)

SetUid

- SetUid allows a process to be run *by* one user but *with the permissions* of another user
  - SetUid/Setgid system calls
  - SetUid is also characteristic of a program in the file system
- E.g. A SetUid root program could be run by normal users but would run with root privileges
- Good idea to set up a special userId with just the privileges you need and setUid that user rather than root

Careful

- If become root (or any user) once, can make a setuid program that can be used any time!
  - Some systems require all setUid programs to be in a special directory that can be monitored
- Alternative: daemon process running with root privileges to which users can send requests for actions
  - Careful with these too – many attacks focused on holes in these!

Domain per process

- Good for programmer to be able to limit the protection domain of a process to the minimal set necessary to accomplish a task
  - Why do I have to give every process I run my full access rights!
  - Trojan horses?
- Even within a process, the rights necessary may vary over the lifetime of the process
  - If only need to certain privileges to initialize, why keep them for the entire life of the process when they might be exploited later

limit/ulimit getrlimit/setrlimit

- Limit resource usage of a process and its descendents
- Examples limits
  - Limit data segment/heap/stack
  - Limit amount of address space mapped (VM limit)
  - Limit max CPU time
  - Limit size of created files and number of files
  - Limit max core file size
- Each descendent gets to reach the limit not cumulative – so still can exceed with lots of children
- Soft/hard limits
  - Any user can decrease or increase up to hard limit
  - Only root can raise hard limits

Other limits

- Quota – allows limiting users consumption of hard disk space
- Chroot – makes a specified directory the root of a processes file system such that it cannot access the rest of the file system
- Free BSD has "jail" for confining root to a subset of special privileges
  - http://docs.freebsd.org/44doc/papers/jail/jail.html

Pluggable Authentication Modules (PAM)

- Linux pluggable user log in procedures
  - Allow various password systems, smart cards, anything behind a standard interface
- Applications like login or ftpd needn't be rewritten for each new mechanism
- PAM also allows setting per user resource limits (similar to ulimit)

Domain per Procedure

- ???

Access Matrix

- Now we've figured out all the objects we want to protect, the types of access we might want to grant and the entities to whom we will grant them
- Result = Access Matrix
  - Rows of matrix can be domains
    - Regardless of granularity of domain
    - If domain per user then row per user
  - Columns are objects or resources
  - Values at entry(i,j) says rights domain i has to object j

Access Matrix

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Figure A

Implementation of Access Matrices

- 2D array – how hard can that be?
- Well its not hard but it is big and often filled with lots of 0's
  - If most domains include have permissions to only a few objects then will be lots of wasted space
  - Avoid this by chopping up the access matrix and compressing
- OS may also choose to divide up into logical sections (I.e. all protection info related to files in one place and all protection info related to users in another)
- Also compression from domain = groups of users
- Also compression from inheritance

Access List

- Chop access matrix into columns and don't list domains that have no access
- With each object store the list of domains that can access it and in what ways
- A domain that is not present in the list has no access rights
- Easy to set a default set of right to an object and then only need to enter exceptions to the default

Capabilities

- Chop access matrix into rows and don't list objects for which you have no rights
- With each domain store the list of objects it can access and in what ways
- Sometimes simply knowing the name of an object gives you access
- Managed by the OS (not managed by process/users directly)
  - Usually process given a handle and the capability pointed to by the handle but stored in the OS
  - Present capability on every access

Speed of access?

- With pure access lists, access list must be searched on each access = slow
- Capabilities on the other hand can be obtained once and then presented with each access
  - Fast as validity check on capability
  - If stored in OS and process just gets a handle then can assume valid

Revocation of Access Rights

- Does revocation take place immediately or is there some propagation delay? If there is a delay is it bounded?
- When a given right is revoked does can it effect just one domain or all? (example: changing a lock vs removing one user from an access list)
- Can we revoke just a few rights to an object or must we revoke them all?
- Can access be permanently revoked or can it be revoked and later obtained again?

Access lists vs capabilities

- With access lists, revocation is easy
  - List of rights held with object, simply edit it in one place
  - Revocation is immediate and can be flexible whether it is general/selective, total/partial and permanent/temporary
- Capabilities make it harder
  - List of rights stored with each domain
  - How do we find everyone with a given right?

Support for revocation in capability based systems

- Periodically have rights time out and force them to be reacquired so can bound time till revocation takes place (not immediate)
- Maintain back-pointers to all domains holding a capability so can find and revoke at any time(costly!)
- Maintain a master key for each object
  - When grant capability give copy of master key
  - To revoke, change master key
  - Then everyone will have to reacquire (not selective)

Combining access lists and capabilities

- In many OS, on first access search access list
- Then enter a capability in the OS for this process and return a "handle" to this capability to process
- Example: file handles
  - When open a file, search access list in file system
  - If open succeeds, enter an open file pointer in the address space of the process along with pointer to file buffers, vnode, etc
  - Return a file descriptor or file handle which is simply an offset into an open file table
  - Use file descriptor on each additional access
  - OS uses open file info but doesn't recheck permissions for each access

Experiment

- Write a program to open a file and then access it many times (maybe ask user before each access)
- After open done successfully and a couple accesses done ok change permissions in the file system to disallow access
- Does it allow additional accesses or not?

Right to the access matrix?

- In addition to object in the matrix, we can also think about rights to the matrix itself
  - Who can add rights to an entry?
  - Who can switch which domain is active?
  - Who can add domains?
- Additional rights
  - Copy right – allow copying of rights to other domains
  - Transfer – migrate rights from one domain to another (different than copying)
  - Owner right – addition of new rights or removal of rights
  - Switch right – ability to switch to a domain, consider domains as object
  - ...

Access Lists in Unix FS

- Unix FS usually contain access lists with each file
- Not very extensive access lists though!
  - Usually just able to specify read, write and execute rights for three groups: user, group and world
- Can imagine more extensive access list information than this?
  - PRO: more flexible
  - CON: more storage

More extensive mechanisms

- More extensive list of possible rights?
  - Larger list of possible rights to files (not just read/write/execute)
- Finer granularity control of who accesses?
  - Allow list of users rather than user/group/all
- Finer grain mechanism allows policies that better match “need to know” principle

AFS access control lists

- Ability to specify additional types of access rights on a directory
  - Administer, delete, insert, lookup, read, write
  - Group into categories
    - Read access – just read
    - Write access – all but administer
    - None
    - All
- Can specify a separate set of access rights for all users and groups (not just single user and group)

AFS Example

- Example:

  % **fs setacl -dir . -acl pat:friends rl smith write**

  % **fs listacl -path** .

  Access list for . is

  Normal rights:

  pat:friends rl

  smith rlidwk

Windows NT family

- Designed with protection/security in mind from the beginnning
- Protection for files, devices, mailslots, pipes, jobs, processes, threads, events, mutexes, semaphores, timers, registry keys,…
- Even earned a security rating from the government
  - Secure logon facility
  - Discretionary access control: allow owner to specify who can access object in what way
  - Security auditing
  - Object reuse protection: zero out all objects before reallocate

-43

NT Access Control Lists

- Two types
- DACL
  - Specify types of access to object
  - List of access control entries that can either specify to allow or deny access
- SACL
  - Specify auditing to be done on access to object
  - Specify both who should audited and what ops should be audited

-44

Hydra

- Multiprocessor OS from CMU 1974
- Extremely fine grained and flexible protection system
- Used capabilities
- Early "object-oriented" system – with OS support for objects
- Extensible security system
  - Users could define new types of objects to be protected

-45

Hydra Objects

- Each object has with it a collection of access rights
  - Manipulated by OS so unforgeable
  - Very early OOP concepts
  - Each object defined by data, operations that can be applied and collection of access rights to it
- Kernel provided operations for the definition of new types of object and associated rights

-46

Hydra procedures

- Each procedure has its code and a list of caller independent capabilities and caller dependent capabilities (holes)
- Local Name Space (LNS)
  - When call a procedure fill in "holes" with your own current capabilities and gain the caller independent capabilities to form a current set of capabilities
- Process = stack not just of procedures but also of capabilities!!
- Great flexibility!
  - Each procedure can upgrade rights for just that procedure and also base access on right of caller

-47

Hydra vs OOP Programming

HYDRA	Programming Language
Object	Variable (Object)
Type	Type
Capability	Pointer + (Access Type?)
Local Name Space	Activation Record
Procedure	Procedure/subroutine
Templates	Formal parameter specification
Call Mechanism	Subroutine call

-48

Hydra Pros and Cons

- Very flexible system
  - Implement "need to know" principle to level of every object and every procedure!
- Requires domain switch for every procedure call and access rights for each object
  - GOOD: Each procedure has only rights required
  - BAD: Expensive to check constantly
    - OS trap per procedure call
- Modern OS support for protection not this extensive
  - As we have extra performance and security more of a worry....

Language-Based Protection

- How far can you get with just language support and not OS support?
- Java VM?
- Do you trust your compiler?
  - Great read for this week "Reflections on Trusting Trust"

Protection vs Security

- So far we have been dealing with protection
- Protection deals with internal access controls
  - Users must log in
  - Access to resources tracked at certain granularity
  - Access is granted by way of access list or capability
- Security on the other hand deals more with external access controls
  - Much more wide reaching!
  - Physical security
  - Psychological attacks
  - Etc.

Example

- We discussed how difficult it would be to guess someone's password
  - We considered things like the length of the key and the types of valid characters
  - We also discussed briefly the tendency of people to choose passwords from a much narrower space
- Security would also consider
  - Physical intimidation/bribes to get people's passwords
  - Physical access to a machine
  - Stunts like pretending to be a system administrator to get someone to voluntarily reveal their password

Physical Security

- Are you sure someone can just walk into your building and
  - Steal floppies or CD-ROMs that are lying around?
  - Bring in a laptop and plug into your dhcp-enable ethernet jacks?
  - Reboot your computer into single user mode? (using a bios password?)
  - Reboot your computer with a live CD-ROM and mount the drives?
  - Sit down at an unlocked screen?
- Can anyone sit down *outside* your building and get on your DHCP-enable 802.11 network?

Social Engineering

- Using tricks and lies that take advantage of people's trust to gain access to an otherwise guarded system.
  - Social Engineering by Phone: "Hi this is your visa credit card company. We have a charge for $3500 that we would like to verify. But, to be sure it's you, please tell me your social security number, pin, mother's maiden name, etc"
  - Dumpster Diving: collecting company info by searching through trash.
  - Online: "hi this is Alice from my other email account on yahoo. I believe someone broke into my account, can you please change the password to "Sucker"?
  - Persuasion: Showing up in a FedEx or police uniform, etc.
  - Bribery/Threats

Administrators

- Persons managing the security of a valued resource consider five steps:

1. Risk assessment: the value of a resource should determine how much effort (or money) is spent protecting it.
   - E.g., If you have nothing in your house of value do you need to lock your doors other than to protect the house itself?
   - If you have an $16,000,000 artwork, you might consider a security guard. (can you trust the guard?)
2. Policy: define the responsibilities of the organization, the employees and management. It should also fix responsibility for implementation, enforcement, audit and review.

Administrators

3. Prevention: taking measures that prevent damage.
   - E.g., firewalls or one-time passwords (e.g., s/key)
4. Detection: measures that allow detection of when an asset has been damaged, altered, or copied.
   - E.g., intrusion detection, trip wire, computer forensics
5. Recovery/Response: restoring systems that were compromised; patch holes.

Outtakes

- Rings in Multics

- Ulimit
  - Linux/tasks.h
  - Understanding the Linux kernel p 78-80
  - http://www.experts-exchange.com/Operating_Systems/Linux/Q_20291950.html
- http://seifried.org/lasg/users

System Management Tasks

- In Unix usually boils down to ability to read/write protected files
  - Editing /etc/password, /etc/group etc
  - Starting services with /etc/rc
  - Adding devices/mounting file systems
- In Windows NT family, boils does to registry access permissions
  - Adding users and groups
  - Managing devices
  - Starting services
- Different interface similar functionality

Windows NT Access Control Lists

- DENY/ALLOW entries
- Obey first matching entry
- Safer to put deny entries first

- Two types: DACL (access) and SACL (auditing)
- FS permissions vs privileges
  - Some permissions bypassed if have appropriate privilege
  - E.g. if have backup privilege (SE_BACKUP_NAME)can read any file regardless of FS permissions
  - E.g. Bypass traverse checking privilege allows user to access C:\foo\bar\baz even if they don't have access to C:\foo

NT

- Inheritance of access rights
  - If object doesn't have ACL go up tree to parents until find something that is inheritable

-61

Novell Access Lists

- Object Rights
  - Browse, create, delete, inheritance control, rename
- Attribute Rights
  - Compare (test but not read), read, add/delete self, write
- Note to other objects besides files
  - Ex. Compare used to say is this your password (but not to read password)
  - Ex. Add/delete self to mailing list : can't read rest of list or write others but can add/remove self

-62

Novell

- NDS specific
- Permissions set on FS are SRWECMFA
- IRF/IRM (inherited rights filter vs inherited rights mask)
  - Set on object to stop the flow of certain permissions from the parent

-63