Protection and Security

Operating Systems

Oleg Goldshmidt

ogoldshmidt@computer.org

Lecture 11

Basic Notions I

- system resources
  - physical (hardware) resources: CPU, memory, devices, etc.
  - logical (software) resources: files, programs, data structures, objects
- protection: internal mechanisms for controlling the access of processes and/or users to the system resources according to predefined rules and policies
  - specification of controls
  - enforcement

Basic Notions II

- security: protection against unauthorized access to, destruction, alteration, or excessive consumption of the system resources
  - malice
  - incompetence
  - stupidity
- common wisdom: “never attribute to malice what can be adequately explained by stupidity”
- security: does not believe in absense of malice
- security is a broader topic than protection

Protection Goals

- no one is trustworthy
  - users don't trust each other
  - the OS does not trust anybody
  - neither the OS nor the users trust the rest of the universe
- resources must be shared in a multiprogramming environment
- it is essential to reconcile these two notions to increase reliability
  - prevent hostile attacks or mischief
  - prevent, detect, fix errors

Mechanisms and Policies

- protection deals with mechanisms for enforcement of policies
- mechanism: how
- policy: what
- establishing control policies
  - fixed in the system design
  - configured by the system administrator
  - configured by users to protect their own resources
  - application-dependent policies
- protection system must be sufficiently flexible to support a variety of policies that may change over time

Processes and Objects

- computer system: processes and objects
  - hardware objects: CPU, memory segments, devices
  - software objects: files, programs, functions, semaphores, etc.
- objects have names and operations — abstract data types
- operations are object-dependent
  - CPU — execute
  - memory segment — read and write
  - tape — read, write, rewind
  - data files — create, open, read, write, delete
  - program file — also execute, etc.

The Need-To-Know Principle

- a process should be able to access only those resources it currently requires — the need-to-know principle
- useful to limit potential damage if anything goes wrong
  - Murphy's law works!
- when a process calls function `foo()` the latter should access its arguments and its local variables
  - what about globals?
- what if the process is a compiler?
  - what files can the compiler access?
  - who can access the compiler?

Protection Domains

- a process operates within a protection domain
- a domain defines a set of objects and the allowed operations on each object
- the ability to execute an operation on an object is called an access right
- an access right is an ordered pair: `<object-id,rights-set>`
  - example: `<file foo, {read,write}>`
- a domain is a collection of access rights
- domains may overlap

Processes and Protection Domains

- static association
  - relatively simple, but it is easy to violate the need-to-know principle
  - e.g., a process executes in two phases, needs read access to an object in phase 1, write in phase 2
  - static solution: give the process both read and write access rights — violates the need-to-know principle
- dynamic association
  - a mechanism must be provided to switch a process from one domain to another
  - and/or change the contents of a domain
  - if the latter is not possible we can always create a new domain and switch

Domain Realization

- each user may be a domain
  - the set of objects and rights depends on the user id
  - domain switching occurs when the user id is changed
    - one user logs out, another logs in
    - an “effective user id” is changed
- each process may be a domain
  - domain switching corresponds to one process sending a message to another process and waiting for a response
- each procedure may be a domain
  - object set — local variables and arguments
  - domain switching on procedure call

UNIX Protection Domains

- userid-based
- file permissions (`rwx`)
  - everything is a file!
  - including devices, sockets, etc.
- `setuid` bit
  - `-rwsr-xr-x 1 root root 23332 Oct 11 2004 /bin/traceroute`
- directory permissions
  - `r--`, `--x`, `r-x`, `rwx`, etc.
- `setgid` bit
  - `drwxr-sr-x 2 olegg os 768 Jan 7 21:14 Lectures`
  - `drwxrwsr-x 2 olegg os 1112 Jan 8 22:36 Drills`
- must be very careful with executables

Changing User ID

- real, effective, and saved user IDs
  - real are set on login, can only be changed by administrator
  - effective are used for file access permissions
  - effective user IDs are set by the `exec()` functions if the SUID bit is set for the executable
    - the code should revoke any extra privileges as soon as they are no longer needed!
  - if effective user ID is changed, `exec()` saves the value in a saved user ID, to restore later
- `setuid()`, `setgid()`
- superuser can change all the IDs, a regular user can only change the effective ID

Alternative User Domains

- special directory for privileged programs
  - change effective userid when running a program in that directory
  - effective uid — the directory owner
  - less flexible
- not allowing changes of user id (TOPS-20)
  - special facilities for access to privileged facilities
    - start system daemons on boot with privileged user id
    - regular users send requests to the daemons

Access Matrix

- A_{ij} defines the set of operations (access rights) permitted in domain D_i on object O_j

objects	$file_1$	$file_2$	$file_3$	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

- process-domain association is decided by the OS
- the contents of the matrix are decided by users

Domain Switching

- add domains to A_{ij} as objects, define a switch operation

objects	F_1	F_2	F_3	P_1	D_1	D_2	D_3	D_4
D_1	R		R			S		
D_2				P			S	S
D_3		R	X					
D_4	RW		RW		S			

- how to change A_{ij}? is it an object? is every entry an object?
- need additional operations: copy, owner, and control

Copy And Owner Rights

objects	F_1	F_2	F_3
D_1	X		W_c
D_2	X	R_c	X
D_3	X		

objects	F_1	F_2	F_3
D_1	X		W_c
D_2	X	R_c	X
D_3	X	R	

- variants: copy, transfer, limited copy

obj	F_1	F_2	F_3
D_1	OX		W
D_2		R_cO	R_cOW_c
D_3	X		

obj	F_1	F_2	F_3
D_1	OX		W
D_2		OR_cW_c	R_cOW_c
D_3		W	W

Control Rights

- copy and owner allow modifying entries in a column
- control allows modifying entries in a row
  - applicable only to domain objects

objects	F_1	F_2	F_3	P_1	D_1	D_2	D_3	D_4
D_1	R		R			S		
D_2				P			S	SC
D_3		R	X					
D_4	W		W		S			

- confinement problem: information held by an object can leak outside of its execution environment

Implementation Of Access Matrix I

- global table: ordered triples `<domain,object,rights>`
  - large, often cannot be held in memory, needs additional I/O
  - e.g., an object is globally readable
    - needs an entry for every domain
- access list for objects: describes columns of the matrix, ordered pair `<domain,rights>` for each object
  - can add a default set of rights, check it before searching the list

Implementation Of Access Matrix II

- capability list for domains: describes rows of the matrix, ordered pair `<object,rights>` per domain
  - capability list itself is a protected object, maintained by the OS and inaccessible to processes
  - protection can be achieved, e.g., by using segmentation
- lock-key mechanism
  - each object has a list of unique bit patterns (“locks”)
  - each domain has a list of unique bit patterns (“keys”)
  - a process running in a domain can access an object only if it has a key that matches one of the locks
  - the key lists must be protected, managed by OS

Comparison Of The Implementations I

- access lists
  - correspond directly to the needs of users
  - domain rights are difficult to determine
  - every access requires a search
- capability lists
  - useful for localizing information for a particular process
  - the protection system must only verify that the capability is valid
  - revocation may be inefficient

Implementation Comparison II

- compromise: lock-key scheme
  - can be effective and flexible, depending on the length of the keys
  - revocation is simple: change the locks
- combination of access lists and capabilities is usually used
  - file access list is checked when the file is opened
  - after that all operations are indexed into the file table

Revocation Of Access Rights I

- types of revocation
  - immediate or delayed?
  - selective or general?
  - partial or total?
  - temporary or permanent?
- simple for access lists
  - immediate
  - all the other variations are possible

Revocation Of Access Rights II

- capabilities are more difficult to revoke
  - distributed through the system — must find them first
  - reacquisition — delete capabilities from each domain periodically, make the processes try to reacquire
  - back-pointers from objects to all associated capabilities — costly
  - indirection — capabilities point to table entries, revoke by deleting the entry from the table — does not allow selective revocation
  - keys — unique per capability bit pattern, compared to a per object “master key” — revoke by changing master keys
    - lists of keys to allow selective revocation

Language-Based Protection

- protection must be available as a tool for application programming
- typing variables, prototyping functions
  - can be stated portably
  - can be more flexible than kernel-based protection
  - sometimes conformance can be checked at compile time
  - casting spoils the picture
  - know thy compiler, use its warning options

Security: Basic Notions I

- prevents misuse, destruction, destabilization, etc.
- security violations may be intentional or accidental
  - starts with a security problem (“hole”)
  - one or more “exploits” are developed
    - may be released “into the wild”, made available to everybody, including bored kids, professional criminals, terrorists, political and military enemies, etc.
  - actual security violations occur
- types of exploits
  - local or remote
  - can be classified according to (maximal) severity of potential damage

Security: Basic Notions II

- types of violations
  - unauthorized access to data: read, write, create, delete
  - unauthorized access to programs: execute
  - denial of service
- physical security
  - physical access to a machine usually means that one can do anything to it
- humans are the weakest link,
  - can be negligent, “socially engineered”, etc.
- authentication: best combining “something you know” with “something you have”

Passwords

- must be protected
- must be non-obvious
- must be practically impossible to brute-force (“dictionary attacks”)
- the user must be able to remember them
- policies
  - revocation (and/or locking)
  - expiration
  - change policies
- how many different passwords can you keep in mind?
  - single sign-on, “wallets”
- UNIX-type passwords (“one-way functions”)

Passwords And Secrets

- problem with passwords: humans divulge them all the time
  - e.g., never give your password to a sysadmin — a real sysadmin never needs it
- how to authenticate non-interactively?
- passing secrets using an untrusted messenger
  - sender puts the message in a strongbox, locks it, gives to the messenger
  - recipient cannot open the box, but can lock it with his own lock and send back
  - sender opens his lock, sends the box again
  - recipient thanks the messenger, unlocks the box

Public Key Infrastructure

- generate a pair of keys, keep one private, exchange public keys with persons or systems you wish to exchange information with securely
  - encrypt with recipient's public key — the message can be decrypted only if the other side has the corresponding private key
  - digital signatures: encrypt with one's private key, then with recipient's public key — recipient decrypts with her private key and then with the sender's public key
  - how to exchange keys securely?
  - need to keep private keys secure forever
- various programs use PKI — `ssh` and `SSL` are well-known examples

Trojans And Backdoors I

- normal situation: a program is written by one person (team, company, government, etc.), executed by another person (team, company, government, etc.)
- the program writer(s) can have a lot of fun (and occasionally profit) with “side effects”
- very difficult to protect against — comprehensive code audits can only be done by professionals, are very costly
- usually there is nothing to inspect — no source code is provided at all
- trust (or carelessness?) is involved — what do you install on your computer?

Trojans And Backdoors II

- backdoors may be intentional or left for debugging and tracing purposes
- leaves the system open to the creator of the backdoor
  - design a killer satellite and take control of it (“Under Siege 2”)
  - skim rounding errors off a large number of bank accounts (numerous fictional and real life examples)
- even “innocent” backdoors can be exploited
- do you trust your compiler?
  - “Reflections on Trusting Trust” by Ken Thompson — `http://www.acm.org/classics/sep95/`

Smashing Stacks For Fun And Profit

- imagine a program that accepts input, puts it in a buffer
- imagine that the size of the input is not checked
- the buffer may overflow, overwriting other memory
- e.g., the return address on the stack may be overwritten
  - by carefully crafted input data!
- the function will execute, jump to a wrong address on return
- that address may contain a command to execute code
  - that was a part of the input to begin with
- some common functions that don't check the size of input: `strcpy(3)`, `gets(3)`, `sprintf(3)`, `vsprintf(3)`

Worms

- infect a target machine with a “vector”
- “vector” connects to originating machine to download the worm
- worm searches for other machines to propagate to
- the original Morris worm (1988) propagated via `rsh`
  - `rsh` allowed users to specify “trusted” machines from which passwordless connections were possible (good for scripting, bad for security)
- plus buffer overflows in `finger`
- plus exploited debugging code in `sendmail` to mail copies of itself
- plus launched a dictionary attack on system passwords

Morris's Worm Replication

- for each new machine looks for copies of itself
- if an active copy found, exits, except every 7th instance
- would probably not be detected if exited always
- proceeding on every 7th duplication could be intended to fool the defenses
- brought down a significant fraction of Sun and VAX machines in the net
- didn't do real damage (but could)
- Morris got 3 years probation, 400 hours of community service, $10,000 fine, probably $100,000 in legal costs (more than 15 years ago)
- went on to develop Yahoo! Stores with Paul Graham

Viruses I

- exploit security holes (e.g., buffer overflows) in systems and applications
- applications that are integrated with the system too tightly (e.g., for performance reasons) may do serious damage
- systems that do not distinguish between regular users and administrators are especially vulnerable
  - in particular single-user personal systems — why have different accounts for one person?
  - single-user personal systems are not administered professionally

Viruses II

- any application may be vulnerable and exploitable: Office, image libraries, sound applications, compression software, etc.
- be careful what you install
- be careful where you go on the net
- keep your systems current!
- use AV software, keep that current!

Networks, Firewalls, VPNs

- it is useful to have company computers connected
- it is useful to connect the company to the net
- it is useful to allow others (e.g., customers) to access certain services
- solution: firewalls and DMZ
- configuring a firewall
  - mostly open: allow everything, close undesirable services
  - mostly closed: close everything, open the necessary services
  - shall we allow all connections originating inside the company?
- VPN — protocols for secure remote communications

Encryption

- open algorithms, secure keys
- cannot implement good security through obscurity!
- based on one-way functions
  - e.g., multiplication is easy, but factoring large numbers may be prohibitively different
- random numbers play an essential role
  - anything slightly non-random can be exploited to assist decryption