Access Control and Operating System Security

John Mitchell

What is security?

- Functionality
  - If user does ⟨some expected input⟩
    Then system does ⟨some expected action⟩
- Security
  - If a user or outsider does ⟨some unexpected thing⟩
    Then system does not do ⟨any really bad action⟩
- Why is security difficult?
  - What are all possible unexpected things?
  - How do we know that all of them are protected?
  - At what level of system abstraction?

General concepts

- Identify threat model
  - Set of possible actions available to attacker
  - Examples
    - Eavesdropper: intercept packets on network
    - Active network attacker: eavesdrop, forge packets
    - Web attacker: set up bad web site; no network attacks
    - Dictionary attacker: has dictionary of common passwords
    - Timing attacker: measure timing on network, bus, etc.
- Investigate consequences of possible attacks
  - Inherently an analytical problem
  - Experiments, knowledge of past attacks helps

Another important idea

- Functionality
  - Expressed using meaningful user actions
    - E.g., well-formed commands to operating system
- Security
  - Design can be good
  - But implementation can be insecure
    - If implementation allows more actions than design, then attack can succeed as a result of implementation error

This lecture

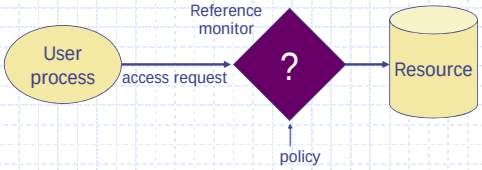
- Operating system security
  - Examples of design features meant to provide security
    - User gets access to resource only if policy allows it
  - Next few lectures: implementation attacks

Outline

- Access Control Concepts
  - Matrix, ACL, Capabilities
- OS Mechanisms
  - Multics
    - Ring structure
  - Amoeba
    - Distributed, capabilities
  - Unix
    - File system, Setuid
  - Windows
    - File system, Tokens, EFS
- Web browser (briefly)
  - "OS of the future"
  - Protect content based on origins instead of user id
- Least privilege
  - Qmail vs Sendmail

Access control

- Assumptions
  - System knows who the user is
    - Authentication via name and password, other credential
  - Access requests pass through gatekeeper
    - System must not allow monitor to be bypassed



Access control matrix [Lampson]

Objects

Subjects	File 1	File 2	File 3	...	File n
User 1	read	write	-	-	read
User 2	write	write	write	-	-
User 3	-	-	-	read	read
...					
User m	read	write	read	write	read

Two implementation concepts

- Access control list (ACL)
  - Store column of matrix with the resource
- Capability
  - User holds a "ticket" for each resource
  - Two variations
    - store row of matrix with user, under OS control
    - unforgeable ticket in user space

	File 1	File 2	...
User 1	read	write	-
User 2	write	write	-
User 3	-	-	read
...			
User m	read	write	write

Access control lists are widely used, often with groups

Some aspects of capability concept are used in Kerberos, ...

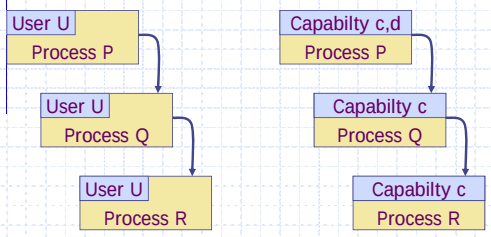
Capabilities

- Operating system concept
  - "... of the future and always will be ..."
- Examples
  - Dennis and van Horn, MIT PDP-1 Timesharing
  - Hydra, StarOS, Intel iAPX 432, Eros, ...
  - Amoeba: distributed, unforgeable tickets
- References
  - Henry Levy, Capability-based Computer Systems
    http://www.cs.washington.edu/homes/levy/capabook/
  - Tanenbaum, Amoeba papers

ACL vs Capabilities

- Access control list
  - Associate list with each object
  - Check user/group against list
  - Relies on authentication: need to know user
- Capabilities
  - Capability is unforgeable ticket
    - Random bit sequence, or managed by OS
    - Can be passed from one process to another
  - Reference monitor checks ticket
    - Does not need to know identify of user/process

ACL vs Capabilities



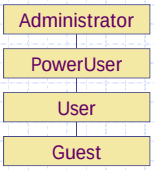
ACL vs Capabilities

- Delegation
  - Cap: Process can pass capability at run time
  - ACL: Try to get owner to add permission to list?
    - More common: let other process act under current user
- Revocation
  - ACL: Remove user or group from list
  - Cap: Try to get capability back from process?
    - Possible in some systems if appropriate bookkeeping
      - OS knows which data is capability
      - If capability is used for multiple resources, have to revoke all or none ...
      - Other details ...

13

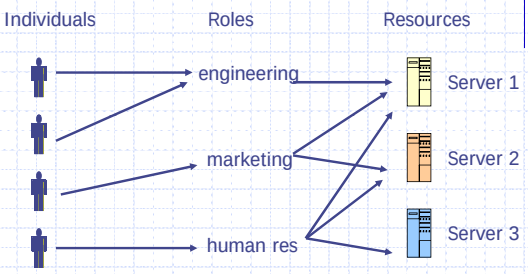
Roles (also called Groups)

- Role = set of users
  - Administrator, PowerUser, User, Guest
  - Assign permissions to roles; each user gets permission
- Role hierarchy
  - Partial order of roles
  - Each role gets permissions of roles below
  - List only new permissions given to each role



14

Role-Based Access Control



Advantage: user's change more frequently than roles

15

Groups for resources, rights

- Permission = ⟨right, resource⟩
- Permission hierarchies
  - If user has right r, and r>s, then user has right s
  - If user has read access to directory, user has read access to every file in directory
- General problem in access control
  - Complex mechanisms require complex input
  - Difficult to configure and maintain
  - Roles, other organizing ideas try to simplify problem

16

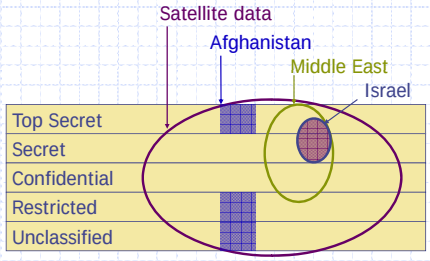
Multi-Level Security (MLS) Concepts

- Military security policy
  - Classification involves sensitivity levels, compartments
  - Do not let classified information leak to unclassified files
- Group individuals and resources
  - Use some form of hierarchy to organize policy
- Other policy concepts
  - Separation of duty
  - "Chinese Wall" Policy

17

Military security policy

- Sensitivity levels
- Compartments



18

Other policy concepts

- Separation of duty
  - If amount is over $10,000, check is only valid if signed by two authorized people
  - Two people must be *different*
  - Policy involves role membership and ≠
- Chinese Wall Policy
  - Lawyers L1, L2 in same firm
  - If company C1 sues C2,
    - L1 and L2 can each work for either C1 or C2
    - No lawyer can work for opposite sides in any case
  - Permission depends on use of other permissions

These policies cannot be represented using access matrix

Example OS Mechanisms

- Multics
- Amoeba
- Unix
- Windows

Multics

- Operating System
  - Designed 1964-1967
    - MIT Project MAC, Bell Labs, GE
  - At peak, ~100 Multics sites
  - Last system, Canadian Department of Defense, Nova Scotia, shut down October, 2000
- Extensive Security Mechanisms
  - Influenced many subsequent systems

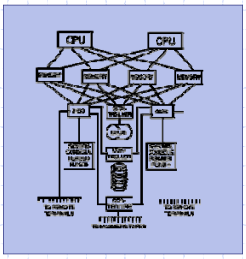
http://www.multicians.org/security.html

E.I. Organick, The Multics System: An Examination of Its Structure, MIT Press, 1972

Multics time period

F.J. Corbato

- Timesharing was new concept
  - Serve Boston area with one 386-based PC



Multics Innovations

- Segmented, Virtual memory
  - Hardware translates virtual address to real address
- High-level language implementation
  - Written in PL/1, only small part in assembly lang
- Shared memory multiprocessor
  - Multiple CPUs share same physical memory
- Relational database
  - Multics Relational Data Store (MRDS) in 1978
- Security
  - Designed to be secure from the beginning
  - First B2 security rating (1980s), only one for years

Multics Access Model

- Ring structure
  - A ring is a domain in which a process executes
  - Numbered 0, 1, 2, ... ; Kernel is ring 0
  - Graduated privileges
    - Processes at ring i have privileges of every ring j > i
- Segments
  - Each data area or procedure is called a segment
  - Segment protection ⟨b1, b2, b3⟩ with b1 ≤ b2 ≤ b3
    - Process/data can be accessed from rings b1 ... b2
    - A process from rings b2 ... b3 can only call segment at restricted entry points

Multics process

- Multiple segments
  - Segments are dynamically linked
  - Linking process uses file system to find segment
  - A segment may be shared by several processes
- Multiple rings
  - Procedure, data segments each in specific ring
  - Access depends on two mechanisms
    - Per-Segment Access Control
      - File author specifies the users that have access to it
    - Concentric Rings of Protection
      - Call or read/write segments in outer rings
      - To access inner ring, go through a "gatekeeper"
- Interprocess communication through "channels"

Amoeba

Server port	Obj #	Rights	Check field

- Distributed system
  - Multiple processors, connected by network
  - Process on A can start a new process on B
  - Location of processes designed to be transparent
- Capability-based system
  - Each object resides on server
  - Invoke operation through message to server
    - Send message with capability and parameters
    - Sever uses object # to indentify object
    - Sever checks rights field to see if operation is allowed
    - Check field prevents processes from forging capabilities

Capabilities

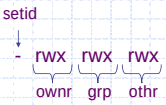
Server port	Obj #	Rights	Check field

- Owner capability
  - When server creates object, returns owner cap.
    - All rights bits are set to 1 (= allow operation)
    - Check field contains 48-bit rand number stored by server
- Derived capability
  - Owner can set some rights bits to 0
  - Calculate new check field
    - XOR rights field with random number from check field
    - Apply one-way function to calculate new check field
  - Server can verify rights and check field
    - Without owner capability, cannot forge derived capability

Protection by user-process at server; no special OS support needed

Unix file security

- Each file has owner and group
- Permissions set by owner
  - Read, write, execute
  - Owner, group, other
  - Represented by vector of four octal values



- Only owner, root can change permissions
  - This privilege cannot be delegated or shared
- Setid bits – Discuss in a few slides

Question

- Owner can have fewer privileges than other
  - What happens?
    - Owner gets access?
    - Owner does not?
- Prioritized resolution of differences

  if user = owner then *owner* permission
  else if user in group then *group* permission
  else *other* permission

Effective user id (EUID)

- Each process has three Ids (+ more under Linux)
  - Real user ID (RUID)
    - same as the user ID of parent (unless changed)
    - used to determine which user started the process
  - Effective user ID (EUID)
    - from set user ID bit on the file being executed, or sys call
    - determines the permissions for process
      - file access and port binding
  - Saved user ID (SUID)
    - So previous EUID can be restored
- Real group ID, effective group ID, used similarly

Process Operations and IDs

- Root
  - ID=0 for superuser root; can access any file
- Fork and Exec
  - Inherit three IDs, except exec of file with setuid bit
- Setuid system calls
  - seteuid(newid) can set EUID to
    - Real ID or saved ID, regardless of current EUID
    - Any ID, if EUID=0
- Details are actually more complicated
  - Several different calls: setuid, seteuid, setreuid

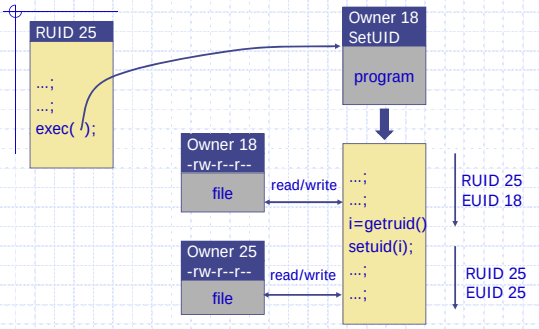
31

Setid bits on executable Unix file

- Three setid bits
  - Setuid – set EUID of process to ID of file owner
  - Setgid – set EGID of process to GID of file
  - Sticky
    - Off: if user has write permission on directory, can rename or remove files, even if not owner
    - On: only file owner, directory owner, and root can rename or remove file in the directory

32

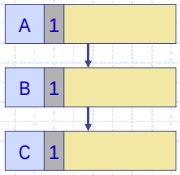
Example



33

Compare to stack inspection

- Careful with Setuid !
  - Can do anything that owner of file is allowed to do
  - Be sure not to
    - Take action for untrusted user
    - Return secret data to untrusted user



Note: anything possible if root; no middle ground between user and root

34

Setuid programming

- Be Careful!
  - Root can do anything; don' t get tricked
  - Principle of least privilege – change EUID when root privileges no longer needed
- Setuid scripts
  - This is a bad idea
  - Historically, race conditions
    - Begin executing setuid program; change contents of program before it loads and is executed

35

Unix summary

- Good things
  - Some protection from most users
  - Flexible enough to make things possible
- Main bad thing
  - Too tempting to use root privileges
  - No way to assume some root privileges without all root privileges

36

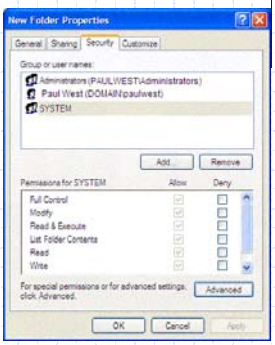
Access control in Windows (NTFS)

- Some basic functionality similar to Unix
  - Specify access for groups and users
    - Read, modify, change owner, delete
- Some additional concepts
  - Tokens
  - Security attributes
- Generally
  - More flexibility than Unix
    - Can define new permissions
    - Can give some but not all administrator privileges

37

Sample permission options

- Security ID (SID)
  - Identity (replaces UID)
    - SID revision number
    - 48-bit authority value
    - variable number of Relative Identifiers (RIDs), for uniqueness
  - Users, groups, computers, domains, domain members all have SIDs



38

Permission Inheritance

- Static permission inheritance (Win NT)
  - Initially, subfolders inherit permissions of folder
  - Folder, subfolder changed independently
  - *Replace Permissions on Subdirectories* command
    - Eliminates any differences in permissions
- Dynamic permission inheritance (Win 2000)
  - Child inherits parent permission, remains linked
  - Parent changes are inherited, except explicit settings
  - Inherited and explicitly-set permissions may conflict
    - Resolution rules
      - Positive permissions are additive
      - Negative permission (deny access) takes priority

39

Tokens

- Security Reference Monitor
  - uses tokens to identify the security context of a process or thread
- Security context
  - privileges, accounts, and groups associated with the process or thread
- Impersonation token
  - thread uses temporarily to adopt a different security context, usually of another user

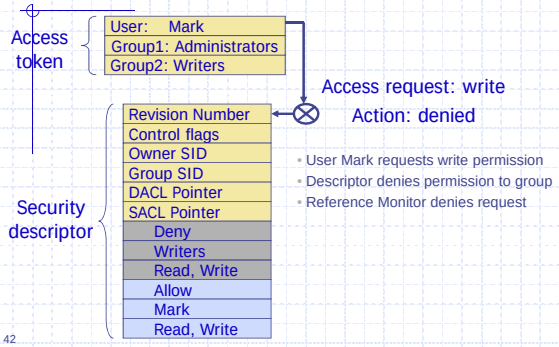
40

Security Descriptor

- Information associated with an object
  - who can perform what actions on the object
- Several fields
  - Header
    - Descriptor revision number
    - Control flags, attributes of the descriptor
      - E.g., memory layout of the descriptor
  - SID of the object's owner
  - SID of the primary group of the object
  - Two attached optional lists:
    - Discretionary Access Control List (DACL) – users, groups, ...
    - System Access Control List (SACL) – system logs, ..

41

Example access request



42

Impersonation Tokens (=setuid?)

- Process uses security attributes of another
  - Client passes impersonation token to server
- Client specifies impersonation level of server
  - Anonymous
    - Token has no information about the client
  - Identification
    - server obtain the SIDs of client and client's privileges, but server cannot impersonate the client
  - Impersonation
    - server identify and impersonate the client
  - Delegation
    - lets server impersonate client on local, remote systems

43

An Analogy

Operating system

- Primitives
  - System calls
  - Processes
  - Disk
- Principals: Users
  - Discretionary access control
- Vulnerabilities
  - Buffer overflow
  - Root exploit

Web browser

- Primitives
  - Document object model
  - Frames
  - Cookies / localStorage
- Principals: “Origins”
  - Mandatory access control
- Vulnerabilities
  - Cross-site scripting
  - Universal scripting

44

Components of browser security policy

- Frame-Frame relationships
  - canScript(A,B)
    - Can Frame A execute a script that manipulates arbitrary/nontrivial DOM elements of Frame B?
  - canNavigate(A,B)
    - Can Frame A change the origin of content for Frame B?
- Frame-principal relationships
  - readCookie(A,S), writeCookie(A,S)
    - Can Frame A read/write cookies from site S?

45

Principles of secure design

- Compartmentalization
  - Principle of least privilege
  - Minimize trust relationships
- Defense in depth
  - Use more than one security mechanism
  - Secure the weakest link
  - Fail securely
- Keep it simple
- Consult experts
  - Don't build what you can easily borrow/steal
  - Open review is effective and informative

46

Compartmentalization

- Divide system into modules
  - Each module serves a specific purpose
  - Assign different access rights to different modules
    - Read/write access to files
    - Read user or network input
    - Execute privileged instructions (e.g., Unix root)
- Principle of least privilege
  - Give each module only the rights it needs

47

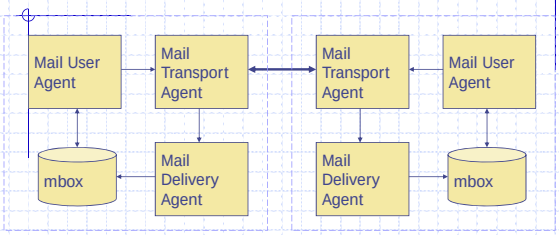
Example: Mail Transport Agents

- Sendmail
  - Complicated system, many past vulnerabilities
  - Sendmail runs as root
    - Root privilege needed to bind port 25
    - No longer needed after port bind established
      - But most systems keep running as root
    - Root privileges needed later to write to user mailboxes
- Qmail
  - Simpler system designed with security in mind

Qmail was written by Dan Bernstein, starting 1995
$500 reward for successful attack; no one has collected

48

Simplified Mail Transactions



- Message composed using an MUA
- MUA gives message to MTA for delivery
  - If local, the MTA gives it to the local MDA
  - If remote, transfer to another MTA

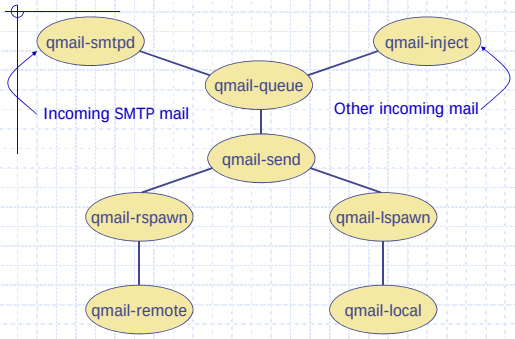
49

Qmail design

- Least privilege
  - Each module uses least privileges necessary
  - Only one setuid program
    - setuid to one of the other qmail user IDs, not root
    - No setuid root binaries
  - Only one run as root
    - Spawns the local delivery program under the UID and GID of the user being delivered to
    - No delivery to root
    - Always changes effective uid to recipient before running user-specified program
- Other secure coding ideas

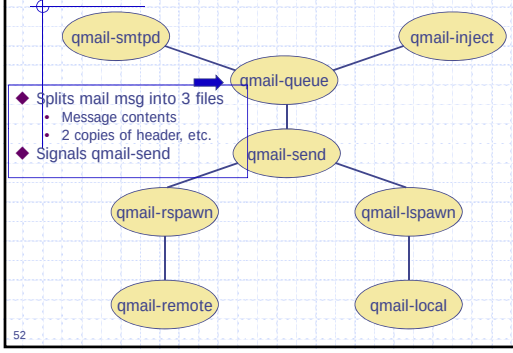
50

Structure of qmail



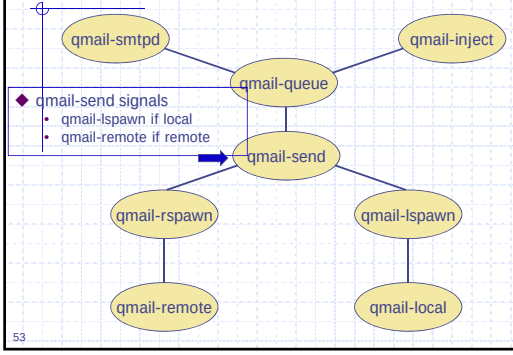
51

Structure of qmail



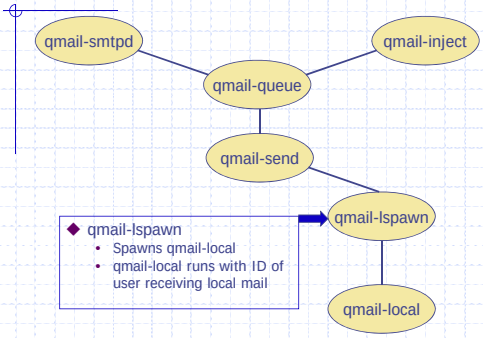
52

Structure of qmail



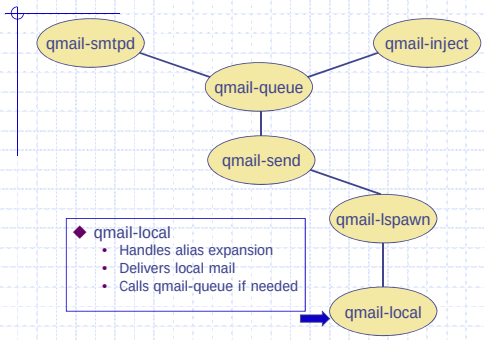
53

Structure of qmail



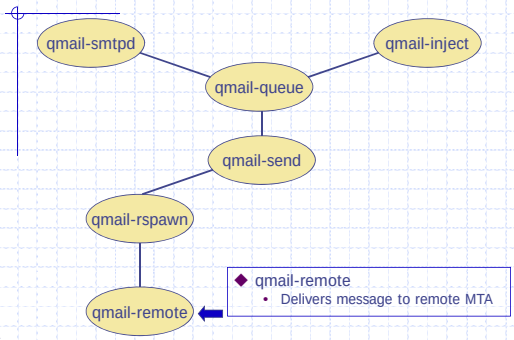
54

Structure of qmail



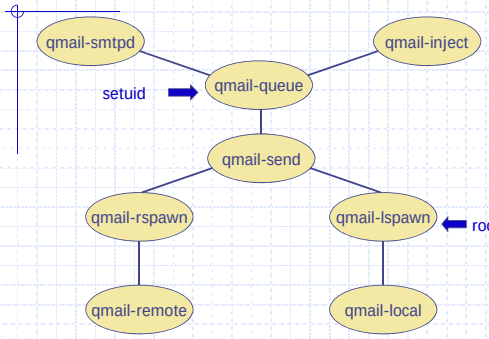
55

Structure of qmail



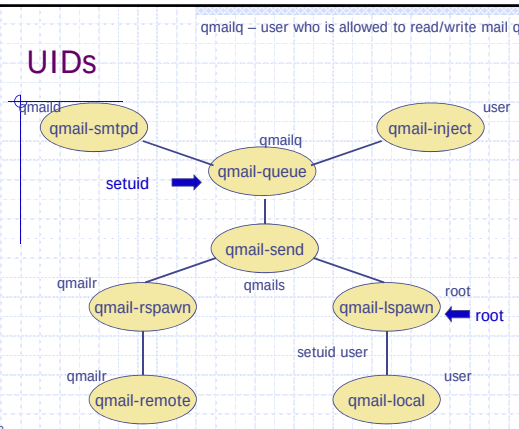
56

Least privilege



57

UIDs



58

Principles, sendmail vs qmail

- Do as little as possible in setuid programs
  - Of 20 recent sendmail security holes, 11 worked only because the entire sendmail system is setuid
  - Only qmail-queue is setuid
    - Its only function is add a new message to the queue
- Do as little as possible as root
  - The entire sendmail system runs as root
    - Operating system protection has no effect
  - Only qmail-start and qmail-lspawn run as root.

59

60