Operating Systems Principles

Security and Privacy

Mark Kampe
(markk@cs.ucla.edu)

Security and Privacy

12A. Operating Systems Security
12B. Authentication
12C. Authorization
12D. Trust
12E. At-Rest Encryption

Operating System Security – Goals

- privacy
  - keep other people from seeing your private data
- integrity
  - keep other people from changing your protected data
- trust
  - programs you run cannot compromise your data
  - remote parties are who they claim to be
  - binding commitments and authoritative records
- controlled sharing
  - you can grant other people access to your data
  - but they can only access it in ways you specify

Security Concepts

- *principals*
  - (e.g. users) own, control, and use protected objects
- *agents*
  - (e.g. programs) act on behalf of principals
- *authentication*
  - confirming the identity of requesting principal
  - confirming the integrity of a request
- *credentials*
  - information that confirms identity of requesting principal
- *authorization*
  - determining if a particular request is allowed
- *mediated access*
  - agents must access objects through control points

Why Security is Difficult

- complexity of our software and systems
  - millions of lines of code, thousands of developers
  - rich and powerful protocols and APIs
  - numerous interactions with other software
  - constantly changing features and technology
  - absence of comprehensive validation tools
- determined and persistent adversaries
  - commercial information theft/black-mail
  - national security, sabotage

Security – Key Elements

- reliable authentication
  - we must be sure who is requesting every operation
  - we must prevent masquerading of people/processes
- trusted policy data
  - policy data <u>accurately describes</u> desired access rules
- reliable enforcement mechanisms
  - all operations on protected objects must be checked
  - it must be impossible to circumvent these checks
- audit trails
  - reliable records of who did what, when

External (user) Authentication

- authentication done by trusted "login" agent
  - typically based on passwords and/or identity tokens
  - movement towards biometric authentication
- ensuring secure passwords
  - they must not be guess-able or brute-force-able
  - they must not be steal-able
- ensuring secure authentication dialogs
  - protection from crackers: humanity checkers
  - protection from snoopers: challenge/response
  - protection from fraudulent servers: certificates
- evolving encryption technology can assist us here

Cryptographic Hash Functions

- "one-way encryption" function: H(M)
  - H(M) is much shorter than M
  - it is inexpensive to compute H(M)
  - it is infeasible to compute M(H)
  - it is infeasible to find an M': H(M') = H(M)
- uses
  - store passwords as H(pw)
    - verify by testing H(entered) = stored H(pw)
  - secure integrity assurance
    - deliver H(msg) over a separate channel

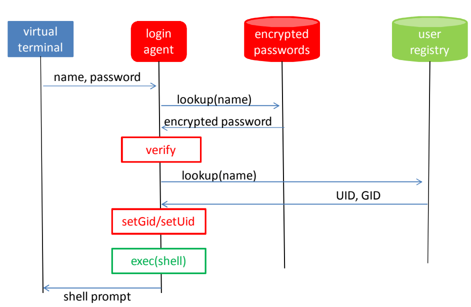
challenge/response authentication

- untrusted authentication
  - client/server distrust one-another & connecting wire
  - both claim to know the secret password
  - neither is willing to send it over the network
- client and server agree on a complex function
  - response = F(challenge,password)
  - F may be well known, but is very difficult to invert
- server issues random challenge string to client
  - server & client both compute F(challenge,password)
  - client sends response to server, server validates it
- man-in-middle cannot snoop, spoof, or replay

Internal (process) Authentication

- OS associates credentials with each process
  - stored, within the OS, in the process descriptor
  - automatically inherited by all child processes
  - identify the agent on whose behalf requests are made
- they are the basis for access control decisions
  - they are consulted when accessing protected data
  - they are reported in audit logs of who did what
- they are established by a privileged system call
  - only a small number of trusted programs can use it
  - they must be carefully written, reviewed, and tested

UNIX Credential Establishment



The Authorization Matrix

Principal	Object 1	Object 2	Object 3	…
User 1	Read/write	Read/write	Read-only	
User 2	read	Read/write		
User 3		Read-only	Read/write	

- A[*principal,object*] = principal's access to object
  - row list of represents principal's capabilities
  - column represents objects's access control list

(The Authorization Matrix)

- provides the answer to access control questions
  - can subject S perform operation O on object X?
  - this can be abstractly thought of as a matrix A[S,X]
- there are two obvious real representations
  - what things a subject is allowed to do (capabilities)
  - who can access an object (access control lists)
- updating this matrix is a critical operation
  - errors in the data will result in incorrect decisions
  - updating this data is, itself a controlled operation (e.g. is S allowed to change access control data for X?)

Capabilities and ACLs

- Capabilities – per agent access control
  - record, for each principal, what it can access
  - each granted access is called a "capability"
  - a capability is required to access any system object
- Access Control Lists – per object access control
  - record, for each object, which principals have access
  - each protected object has an Access Control List
  - OS consults ACL when granting access to any object
- Either must be protected & enforced by the OS

Access Control Lists vs. Capabilities

- Access Control Lists
  - short to store and easy to administer
- Capabilities make very convenient handles
  - if you have the capability, you can do the operation
  - without one, you can't even ask for operations
- many operating systems actually use both
  - ACLs describe what accesses are allowed
  - when access is granted, a Capability is issued
  - capability is used as handle for subsequent operations

Unix files – access control lists

- Subject Credentials:
  - user and group ID, established by password login
- Supported operations:
  - read, write, execute, chown, chgrp, chmod
- Representation of ACL information:
  - rules (owner:rwx, group:rwx, others:rwx)
  - owner privileges apply to the file's owner
  - group privileges apply to the file's owning group
  - others privileges apply to all other users
  - only owner can chown/chgrp/chmod

Unix File Access – example

given a file with:

user ID: 100

group ID: 15

file protection: r w x r - x r - -

UID/GID	read	write	execute	chmod
100/001	yes	yes	yes	yes
001/015	yes	no	yes	no
001/001	yes	no	no	no
000/###*	yes	yes	yes	yes

* In UNIX, a process with UID=0 (super user) can do anything

Unix files also have capabilities

- if a process wants to read or write a file
  - it must open the file, requesting read or write access
  - open will check permissions before granting access
  - if operation permitted, OS returns a file descriptor
- the user file descriptor is a capability
  - it is an unforgable token conferring access to the file
  - it confers a specific access (r/w) to a specific file
  - a required argument to the read/write system calls
  - without a file descriptor reads/writes are impossible

Truly Unforgeable Capabilities

- real capabilities come from a trusted source (OS)
  - who checks access permissions before granting them
  - having a capability conveys access to the resource
- resource references must be unforgable
  - otherwise people could forge references for anything
- ensure this by keeping them inside the OS
  - give the user an index into a per-process table
    - e.g. user file descriptors are index into a per-process array
  - process can only refer to capabilities by index number
- a system call can pass capabilities to others
  - because only the OS can create the table entries

Very Hard-to-forge Capabilities

- random cookies from sparse name spaces
  - they can be verified, but are very difficult to forge
  - this is easily achieved with encryption techology
- resource mgr decrypts cookie on each request
  - determine which object is to be used
  - ensure requester has adequate access for operation
- this is also a very common approach
  - product activation codes (product, version)
  - heavily exploited in distributed systems
- such cookies are easily exchanged in messages

Enforcing Access Control

- protected resources must be inaccessible
  - hardware protection must be used to ensure this
  - only the OS can make them accessible to a process
- to get access, issue request to resource manager
  - resource manager consults access control policy data
- access may be granted directly
  - resource manager maps resource into process
- access may be granted indirectly
  - resource manager returns a “capability” to process
  - capability can be used in subsequent requests

Access Mediation

- Per-Operation Mediation (e.g. file)
  - all operations are via requests
  - we can check access on every operation
  - revocation is simple (cancel the capability)
  - access is relatively expensive (system call/request)
- Open-Time Mediation (e.g. shared segment)
  - one-time access check at open time
  - if permitted, resources is mapped in to process
  - subsequent access is <u>direct</u> (very efficient)
  - revocation may be difficult or awkward

Principle of Least Privilege

- operate with minimum possible privileges
  - surrender privileges when no longer needed
  - operate in the most restricted possible context
- allow minimum possible access to resources
  - apply multiple levels of protection
- trust, but verify
  - sanity check requests before performing them
- minimize amount of privileged software
  - minimize the attack surface
  - minimize amount of code to be audited

Quis Custodiet ipsos Custodes?

- OS can do a very good job of enforcement
  - if reasonably designed, reviewed, and implemented
- What does the OS enforce?
  - all access is according to access control database
- Enforcement is only as good as the policy data
  - human beings set up the authorization policy data
  - they may misunderstand our intentions
  - they may make errors in entering the rules
  - they may deliberately violate our intentions
- These are problems the OS cannot solve

Privileged Users – the big hole

- OS Maintenance requires extraordinary privileges
  - installing and configuring system software
  - backing up and restoring file systems
- many systems have privileged users
  - authorized to update system files
  - authorized to perform privileged operations
  - often there is a Super-User, who can do anything
- users with these passwords are dangerous
  - they can make mistakes or do mischief
  - they can leak the passwords to others

Finer Granularity Authorization

- "super users" are dangerous
  - they are permitted to do anything
    - not merely a single particular privileged operation
  - accidentally mistyped commands can be disastrous
    - ordinary file protections do not prevent them
- finer granularities of privilege
  - backups, file system allocation, user creation, etc.
- finer granularities of operations
  - privilege granted for only one operation at a time
  - confirmation dialogs in system management tools

Role Based Access Control (RBAC)

- system management is not "a person"
  - it is a role that some people, sometimes, perform
- don't predicate authorization decisions on identity
  - users are authorized to perform roles
  - they must declare that they are operating in a role
    - checks their authorization to function in the role
    - creates credentials to authorize role based operations
  - privileged operations check role credentials
    - specifically check for role-specific privileges
- superior authorization control
  - fine grained operation control for limited periods
  - audit records record the "real person" who took the actions

Trusted Computing Base

- All protection information stored in OS
  - applications cannot directly access/modify it
- OS creates and maintains process state
  - OS can associate a principal w/each process
- OS implements file, process, IPC operations
  - OS can mediate all access to these objects
  - no way to access without going through OS
- This is a foundation on which apps run
  - apps can depend on processes and files
  - higher level services can depend on these

Trust Worthy Software

- very carefully developed
  - designed with security as a primary goal
  - stringent design and code review processes
  - extensive testing
  - open source helps, but is a two-edged sword
- obtained from a trusted source
  - who can certify its authenticity
  - who has a high stake in its correctness
  - who maintains and updates it well

Trusted Applications

- Not all trusted code is in the OS kernel
  - file system management and back-up
  - login and user-account management
  - network services (remote file systems, email)
- These applications have special privileges
  - they can execute privileged system calls
  - they can access files that belong to multiple users
  - they can access otherwise protected devices
  - they can compromise system security

Special Application Privileges

- privileged daemons ... started by the OS
  - many system daemons run as the super user
  - others are run as the owner of key resources
- privileged commands ... run by users
  - UNIX SetUID/SetGID load modules
  - run with the credentials of the program's owner
  - may be able to create/set their own credentials
    - e.g. login, sudo
  - these must be very carefully designed/reviewed

Can we trust trusted applications?

- most complex programs have many bugs
  - unfortunately even the best code is imperfect
  - some bugs just make the program fail
  - some bugs make the programs do the wrong thing
- real example: login buffer overflow bug
  - login program checks entered passwd w/correct one
  - buffer for real passwd is after buffer for entered one
  - entering a very long password overwrites real one
- determined hackers will find & exploit such bugs

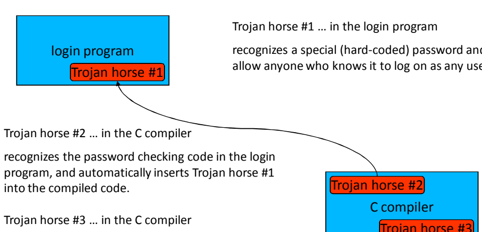
the login buffer overflow bug

```
char inbuf[80];                   /* buffer for user entered password     */
char pwbuf[80];                   /* buffer for real password (encrypted) */
....
getpwent( uname, pwbuf );         /* get real (encrypted) password        */
stty( 0, no_echo );               /* no echo, character at a time input   */
write(1,"password: ", 9);         /* prompt user for password             */
p = inbuf;
do { read(0, p, 1);               /* read password entered by user        */
        } while (*p++) != '\n');  /* until a newline character is entered */
pwencrypt(inbuf);                 /* encrypt what the user entered        */
if (strncmp(inbuf, pwbuf, 8) == 0)  /* see if it matches real password    */
   ... he's in
```


Trojan Horses

- accidental bugs in trusted software create holes
  - what if the software was designed with evil intent?
- the original "Trojan Horse" and the fall of Troy
  - the Greeks built it, left it, and departed
  - the Trojans thought it was a tribute to their valor
  - the Trojans brought it into the city and had a party
  - that night, soldiers came out and destroyed Troy
- modern "Trojan Horses" (pfishing)
  - pretend to be the login program
  - pretend to be financial institution web-page

Ken Thompson's 3-part Trojan Horse



Trojan horse #1 ... in the login program

recognizes a special (hard-coded) password and will allow anyone who knows it to log on as any user.

Trojan horse #2 ... in the C compiler

recognizes the password checking code in the login program, and automatically inserts Trojan horse #1 into the compiled code.

Trojan horse #3 ... in the C compiler

recognizes the code generator in the C compiler, and automatically inserts both Trojan horses (#2 and #3) into the compiled code.

None of these can be found by reading the code of either the login program or compiler.

Bypassing Mediation

- OS can enforce authorization policy
  - control the operations processes can perform
- OS enforcement has exceptions and limits
  - privileged users can override file protection
  - passwords can be observed/stolen/guessed
  - bugs may enable malware to gain privileges
  - backups can be accessed w/o the OS
  - file systems can be accessed w/o OS
  - data stored in the cloud is beyond our protection

At-Rest Encryption

- added data protection, beyond file protection
- Disk (or file system) level
  – password must be given at boot or mount time
  – driver or file system does encrypt/decrypt
  – protects computer against unauthorized access
- File level
  – password must be given when file is opened
  – application (or library) does encrypt/decrypt
  – protects file against unauthorized access

Assignments

- for the next lecture:
  – Challenges of Distributed Systems
  – Arpaci ch 47 … distributed dystems
  – Saltzer sections 11.3-4 … distributed security
  – Secure Socket Layer … private session protocol
  – RESTful interfaces … a new interface paradigm
  – Resource Leases … distributed before/after
  – Distributed Transactions … distributed all/none
  – Distributed Consensus … a hard problem

Supplementary Slides

Why Should we Trust the OS

- Can we trust the supplier's intentions?
  – do they have the right business incentives?
  – will their customers keep them honest?
- Can we trust the supplier's processes?
  – design and code review processes
  – testing processes (including penetration)
  – security bug fixes and patches
  – security bug frequency and severity
- Open Source … a two edged sword

Direct Access to Resources

- resource is mapped into process address space
  – process manipulates resource w/normal instructions
  – examples: shared data segment or video frame buffer
- advantages
  – access check is performed only once, at grant time
  – very efficient, process can access resource directly
- disadvantages
  – process may be able to corrupt the resource
  – access revocation may be awkward

Indirect Access to Resources

- resource is not directly mapped into process
  – process must issue service requests to use resource
  – examples: network and IPC connections
- advantages
  – only resource manager actually touches resource
  – resource manager can ensure integrity of resource
  – access can be checked, blocked, revoked at any time
- disadvantages
  – overhead of system call every time resource is used

Can we trust the OS?

- trusted software is developed with great care
  - it is very carefully designed, reviewed, and tested
  - it may be audited/certified by a respected third party
- but we obtain software from insecure places
  - e.g. down-loading drivers, applications and plug-ins
- how can we know new software is good?
  - is it authentic, or a cleverly crafted Trojan horse?
  - has an originally good program been infected?
- we need tamper-proof certificates of authenticity

Computer Viruses

- a biological virus is the simplest form of life
  - so simple that people argue about whether it is alive
- a biological virus can only do three things:
  - penetrate cells and get to the nucleus
  - force the cell to replicate many more copies of itself
  - copies spread to other cells, the process continues
- a computer virus is completely analogous
  - enter computer, copy itself, spread to other computers
  - enters system through e-mail or infected software
  - some merely reproduce, others are destructive