Operating System Security

Mike Swift
CSE 451
Autumn 2003

Outline

- Overarching goal: safe sharing
- Authentication
- Authorization
- Reference Monitors
- Confinement

Safe Sharing

- Protecting a single computer with one user is easy
  - Prevent everybody else from having access
  - Encrypt all data with a key only one person knows
- Sharing resources safely is hard
  - Preventing some people from reading private data (e.g. grades)
  - Prevent some people from using too many resources (e.g. disk space)
  - Prevent some people from interfering with other programs (e.g. inserting key strokes / modifying displays)

Why is security hard?

- Security slows things down
- Security gets in the way
- Security adds no value if there are no attacks
- Only the government used to pay for security
  - The Internet made us all potential victims

Trusted Computing Base (TCB)

- Think carefully about what you are trusting with your information
  - if you type your password on a keyboard, you're trusting:
    - the keyboard manufacturer
    - your computer manufacturer
    - your operating system
    - the password library
    - the application that's checking the password
  - TCB = set of components (hardware, software, wetware) that you trust your secrets with
- Public web kiosks should *not* be in your TCB
  - should your OS?
    - but what if it is promiscuous? (e.g., IE and active-X extensions)
  - how about your compiler?
    - A great read: "Reflections on Trusting Trust".

Security Techniques

- Authentication – identifying users and programs
- Authorization – determining what access users and programs have to things
  - Complete mediation: check every access to every protected object
- Auditing – record what users and programs are doing for later analysis

Authentication

- How does a computer know who I am?
  - User name / password
    - How do it store the password?
    - How do it check the password?
    - How secure is a password?
  - Public/Private Keys
  - Biometrics
- What does the computer do with this information?
  - Assign you an identifier
    - Unix: 32 bit number stored in process structure
    - Windows NT: 27 byte number, stored in an *access token* in kernel

Storing passwords

- CTSS (1962): password file

Bob: 14: "12.14.52"
David: 15: "allison"
Mary: 16: "!ofotc2n"

- Unix (1974): encrypt passwords with passwords

$K=[0]_{allison}$

Bob: 14: S6Uu0cYDVdTAk
David: 15: J2ZI4ndBL6X.M
Mary: 16: VW2bqvTalBJKg

- Unix (1979): salted passwords

$K=[0]_{allison392}$

Bob: 14: S6Uu0cYDVdTAk: 45
David: 15: J2ZI4ndBL6X.M: 392
Mary: 16: VW2bqvTalBJKg: 152

More Storing Passwords

- Unix-style password file
  - Password file not protected, because information in it can't be used to logon
  - Doesn't work for network authentication
    - Doesn't contain any secret information
- Windows-NT style password file
  - Contains MD4 hash of passwords
  - Hash must be protected because it can be used to log on
    - Hidden from users
    - Encrypted by random key
    - Physical security required

Password Security

- 26 letters used, 7 letters long
  - 8 billion passwords (33 bits)
  - Checking 100,000/second breaks in 22 hours
    - System should make checking passwords slow
- Adding symbols and numbers and longer passwords
  - 95 characters, 14 characters long
  - 10^{27} passwords = 91 bits
  - Checking 100,000/second breaks in 10^{14} years
- SDSC computed 207 billion hashes for 50 million passwords in 80 minutes.
  - Hashing all passwords for one salt takes 20 minutes on a P4

Do longer passwords work?

- People can't remember 14-character strings of random characters
- Random number generators aren't always that good.
- People write down difficult passwords
- People give out passwords to strangers
- Passwords can show up on disk

Authorization

- How does the system know what I'm allowed to do?
  - Authorization matrix:
    - Objects = things that can be accessed
    - Subjects = things that can do the accessing (users or programs)
  - What are the limits?
    - Time of day
    - Ranges of values

	Alice	Bob	Carl
/etc	Read	Read	Read Write
/homes	Read Write	Read Write	Read Write
/usr	None	None	Read

Access Control Lists

- Representation used in Windows NT, Unix for files
- Stored on each file / directory

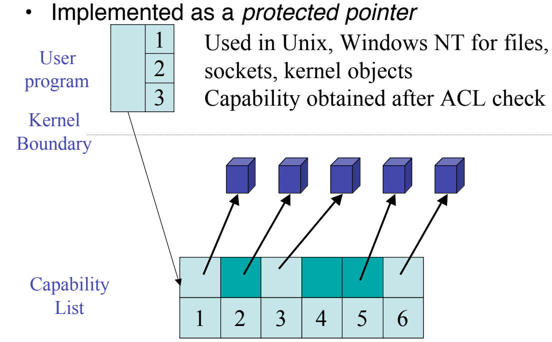
Bob	Read, Write, Delete
Students	Read
Everyone	Read

Unix:
  Fixed set of permissions (read,write,delete)
  Three sets of subjects (owner, group, world)
Windows NT
  Arbitrary number of entries
  16 permissions per object

Capabilities

- Once granted, can be used to get access to an object
- Implemented as a *protected pointer*

Used in Unix, Windows NT for files,
sockets, kernel objects
Capability obtained after ACL check



Which one is better

- ACLs:
  - Can have large numbers of objects
  - Easy to grant access to many objects at once
  - Require expensive operation on every access
- Capabilities
  - Hard to manage huge number of capabilities
  - They have to come from somewhere
  - They are fast to use (just pointer dereferences)
- Most systems use both
  - ACLs for opening an object (e.g. fopen())
  - Capabilities for performing operations (e.g. read())

Protection Domain Concept

- A protection domain is the set of objects and permissions on those objects that executing code may access
  - e.g. a process
    - memory
    - files
    - sockets
  - also: a device driver, a user, a single procedure
- Capabilities:
  - protection domain defined by what is in the capability list
- ACLs
  - protection domain defined by the complete set of objects code could access

How does this get implemented?

- Originally:
  - every application had its own security checking code,
  - Separate set of users
  - Separate set of objects
  - Separate kinds of ACLs, capabilities
- This makes the trusted computing base) huge!!!
  - You have to trust all applications do to this correctly!
- Now: Reference monitor
  - Manages identity
  - Performs all access checks
  - Small, well-tested piece of code

Modern security problems

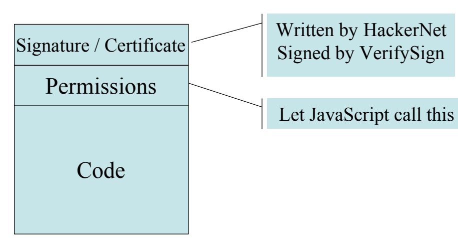
- Confinement
  - How do I run code that I don't trust?
    - E.g. RealPlayer, Flash
  - How do I restrict the data it can communicate?
  - What if trusted code has bugs?
    - E.g. Internet Explorer
- Concepts:
  - Least Privilege: programs should only run with the *minimal* amount of privilege necessary
- Solutions:
  - Restricted contexts - let the user divide their identity
  - ActiveX – make code writer identify self
  - Java – use a virtual machine that intercepts all calls
  - Binary rewriting - modify the program to force it to be safe

Restricted Contexts

- Add extra identity information to an a process
  - e.g. both username and program name (mikesw:navigator)
- Use both identities for access checks
  - Add extra security checks at system calls that use program name
  - Add extra ACLs on objects that grant/deny access to the program
- Allows user to sub-class themselves for less-trusted programs

ActiveX

- All code comes with a public-key signature
- Code indicates what privileges it needs
- Web browser verifies certificate
- Once verified, code is completely trusted



Java

- All problems are solved by a layer of indirection
  - All code runs on a virtual machine
  - Virtual machine tracks security permissions
  - Allows fancier access control models - allows stack walking
- JVM doesn't work for other languages
- Virtual machines can be used with all languages
  - Run virtual machine for hardware
  - Inspect stack to determine *subject* for access checks

Com.msft.sql-srv.query
Com.sun.jdbc-odbc.stmt
Java.jdbc.Statement
edu.washington.cse451

Binary Rewriting

- Goal: enforce code safety by *embedding* checks in the code
- Solution:
  - Compute a mask of accessible addresses
  - Replace system calls with calls to special code

Original Code:

```
lw   $a0, 14($s4)
jal  ($s5)
move $a0, $v0
jal $printf
```

Rewritten Code:

```
and $t6,$s4,0x001fff0
lw   $a0, 14($t6)
and  $t6,$s5, 0x001fff0
jal  ($t6)
move $a0, $v0
jal $sfi_printf
```