

# CS 140: Operating Systems

## Lecture 21: Protection

Mendel Rosenblum

### Readings

- ◆ Silberchatz, Galvin, Gagne:
  - 7<sup>th</sup> edition: chapters 14, 15
  - 6<sup>th</sup> edition: chapters 18, 19
- ◆ Read Thompson article "Reflections on Trusting Trust".
- ◆ Read Lampson paper "A Note on the Confinement Problem".
- ◆ Both on website.


### Protection

- ◆ The purpose of a protection system is to prevent accidental or intentional misuse of a system.
- ◆ Accidents
  - A program mistakenly deletes the root directory. No one can login.
  - This sort of problem (relatively) easy to solve: just make the likelihood small.
- ◆ Malicious abuse:
  - A high school brat breaks the password for the B of A accounting system and transfers \$3 billion to his account to buy Britney Spears CDs.
  - This kind of problem very hard to completely eliminate (no loopholes, can't play on probabilities)

### Policy versus mechanism

- ◆ A good way to look at the problem is to separate policy (what) from mechanism (how)
- ◆ A protection system is the mechanism to enforce a security policy
  - Roughly the same set of choices, no matter what policy.
- ◆ A security policy delineates what acceptable behavior and unacceptable behavior.
- ◆ Example security policies:
  - That each user can only allocate 40MB of disk.
  - That no one but root can write to the password file.
  - That you can't read my mail.

### Core components of a protection mechanism

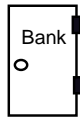
- ◆ Authentication: make sure we know who we are talking to
  - 
  - Unix: password
  - Credit card companies: social security # + mom's name
  - Bar's: driver's license
  - (theme: not so reliable in the real world...)
- ◆ Authorization: determine if x is allowed to do y.
  - Need a simple database to do this.
- ◆ Access enforcement: enforce authorization decision
  - Must make sure there are no loopholes
  - This is really really hard. A single flaw ruins the whole system.

### There is no perfect protection system

- ◆ Very simple point, very easy to miss:
  - Protection can only increase the effort ("work factor") needed to do something bad. It cannot prevent it.
- ◆ Even assuming a technically perfect system, there are always the four Bs:
  - Burglary: if you can't break into my system, you can always steal it (called "physical security").
  - Bribery: find whoever has access to what you want and bribe them.
  - Blackmail. Or photograph them in a compromising position.
  - Bludgeoning. Or just beat them until they tell you.
- ◆ Every system has holes, it just depends on what they look like.

### Social versus technical enforcement

- ◆ Can use technical mechanisms to enforce security policies. Example, put up a door and lock it.




- ◆ Or social mechanism: make it illegal to steal.
  - Pro:** immediately “installed” at all locations, is zero-overhead, backwards compatible, etc.
  - Con:** requires a reasonably effective means of detection and enforcement. Laws slow to enact and crude. works best when distance is smaller: hard to extradite from Sweden.

### Authentication



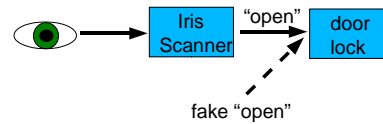
- ◆ Usually done with passwords.
  - This is usually a relatively weak form of authentication, since it's something that people have to remember. Empirically is typically based on girlfriend/boyfriend name.
- ◆ Passwords should not be stored in a directly-readable form
  - Use some sort of one-way-transformation (a “secure hash”) and store that
  - if you look in /etc/passwd will see a bunch of gibberish associated with each name. That is the password.
- ◆ Problem: to prevent guessing (“dictionary attacks”) passwords should be long and obscure
  - Unfortunately easily forgotten and usually written down.

### Authentication alternatives

- ◆ Badge or key 
- ◆ Does not have to be kept secret
  - usually some sort of picture ID worn on jacket (e.g., at military bases)
- ◆ Should not be forgeable or copiable
- ◆ Can be stolen, but the owner should know if it is (but what to do? If you issue another, how to invalidate old?)
- ◆ This is similar to the notion of a “capability” that we'll see later

### Biometrics

- ◆ Biometrics
  - Authentication of a person based on a physiological or behavioral characteristic.
- ◆ Example features:
  - Face, Fingerprints, Hand geometry, Handwriting, Iris, Retinal, Vein, Voice.
- ◆ Strong authentication but still need a “Trusted Path”.



### Authorization

- ◆ Once identity known, what is Bob allowed to do?
  - More generally: must be able to determine what each “principle” is allowed to do with what.**
- ◆ Can be represented as an “access matrix” with one row per principle, one column per resource.

	File A	Printer 1	TTY 3	...
Usr1	R	W	RW	
Usr2	RW	W	---	
Usr3	---	W	---	

### Practical authorization

- ◆ Full access matrix too bulky; so store in one of two condensed ways
  - Capabilities: use matrix rows: for each principle, what resources is it allowed to use?**
  - Access control lists: use matrix column: for each resource, who is allowed to use it?**

	File A	Printer 1	TTY 3	...
Usr1	R	W	RW	
Usr2	RW	W	---	
Usr3	---	W	---	

Surprise: implications of each choice very different!

### Access Control Lists (ACLs)

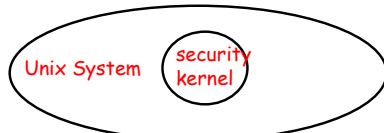
- ◆ With each object, indicate which users are allowed to perform which operations.  
 In most general form, each object has a list of <user,priviledged> pairs.  
 Example: AFS acIs on files
- ◆ Concern: size of ACLs  
 Allow groups. Unix example: 9bits self, group, other.
- ◆ Access control lists are simple, and are used in almost all file systems.

### Capabilities

- ◆ With each users, indicate which files may be accessed and in what ways.  
 Store a lists of <object, privilege> pairs which each user.  
 Called a *Capability List*
- ◆ Capabilities frequently do both naming and protection  
 Can only “see” an object if you have a capability for it.  
 Default is no access.
- ◆ Capabilities are used in system that need to be very secure.
- ◆ Examples:  
 Page tables?  
 Unlisted phone numbers? Non-linked URL?

### Access enforcement

- ◆ Some part of the system must be responsible for **Enforcing access controls.**  
 Protecting authorization information.  
 This part of the system is god. It can do anything it wants.  
 If it has a bug, the entire system can be destroyed.  
 Want it to be as small & simple as possible



Most systems (NT, Unix) let entire OS run in trusted mode...

### Weakest link

- ◆ Security is only as strong as the weakest link in the protection system.
- ◆ Hardware?
- ◆ Software?
- ◆ Users!!!

### Some easy attacks

- ◆ Abuse of valid privilege:  
 On Unix, super-user can do anything. Read your mail, send mail in your name, etc.  
 More prosaic: You delete the code for cs140 project 4. Your partner is not happy.
- ◆ Spoiler/Denial of service (DoS)  
 Use up all resources and make system crash.  
 S script to: “while(1) { mkdir foo; cd foo; }”  
 C program: “while(1) { fork(); malloc(1000)[40] = 1; }”
- ◆ Listener:  
 Passively watch network traffic.  
 Will see anyone’s password as they type it into telnet.  
 Or just watch for file traffic: Will be transmitted in plaintext.

### More attacks

- ◆ Trojan Horse (Imposter)  
 Example: leave a program around that looks like the login process:  
 Fake ATM machine.  
 Fake deposit slips.  
 Need to authenticate the computer!!!  
 Ctl-alt-del in Windows NT/2000/XP.
- ◆ Worm or virus  
 A Trojan Horse or similar program that is also capable of spreading itself from machine to machine.

### Examples of Penetration

- ◆ Tenex page-fault caper: team cracked passwords by aligning on page boundary, measuring time to check validity. (timing attack)
- ◆ Sendmail/finger worm: first big Internet worm in Fall of 1988. Two attacks:
  - Sendmail attack: take advantage of “debug” command left enabled to execute code as super-user. Setup Trojan Horses on machine.**
  - Fingerd attack: give long name to fingerd, which overflows buffer, modifies stack in clever way, causing procedure to be executed with root privilege. Very common internet attack these days.**

### Another example: Thompson’s Unix hack

- ◆ An undetectable Trojan horse
- ◆ Have login program recognize special login.
- ◆ Too obvious, change compiler so when compiling login program it adds special login check.
- ◆ Still visible in compiler, add code to compiler-compiler to add code to compiler to add code to login program!
- ◆ Different is only in bootstrap compiler!

Note: Once the system has been penetrated, it may be impossible to secure it again, hooks could have been left around for imposter to regain control.

### More observations

- ◆ It is not always possible to tell when the system has been penetrated, since the villain can clean up all traces behind himself.
  - One example: Company denied break-in even after guy confessed!**
- ◆ If we can never be sure there are no bugs, then we can never be sure that the system is secure, since bugs could provide loopholes in the protection mechanisms.
- ◆ Rootkits
  - Under the OS, undetectable!**

### Solutions: nothing works perfectly

- ◆ Logging: record all important actions and uses of privilege in an indelible file.
  - Can be used to detect break-in attempts**
  - Useful if you have human to look at logs.**
  - Need to protect logs!**
- ◆ Principle of minimum privilege (“need-to-know” principle): each piece of the system access to a minimum amount of information. (e.g. file system can’t touch page tables, etc.)
  - Hard to do: Example: Containment problem**
- ◆ Correctness proofs: very hard to do. Only proofs systems matches spec, not spec is right.

### Computers vs. Humans Differences

- ◆ Computer memory is volatile, humans don’t forget.
- ◆ Anonymity is easy to come by in computers (can’t really tell who is doing what).
- ◆ We are much more trusting of computers than of people: privileges are giving away freely in huge doses: any program you run could conceivably modify any of your files.