CS 140 – Summer 2008 – Handout #24: Protection & Security

Readings

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

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 log in.**
  - **This sort of problem (relatively) easy to solve: just make the likelihood small.**
- Malicious abuse:
  - **Someone guesses the password to your favorite web site, and installs a malicious Flash application which installs a key logger.**
  - **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 is acceptable behavior vs. unacceptable behavior.
- Example security policies:
  - **That each user can only allocate 200MB 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

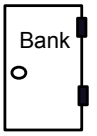
  - **Unix: password**
  - **Credit card companies: social security # + mom's name**
  - **Bar's: driver's license**
  - **(theme: not so reliable in the real world…)**
- Authorization: determine if x is allowed to do y.
  - **Need a simple database to do this.**
- Access enforcement: enforce authorization 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 small: 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/passwords 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.**

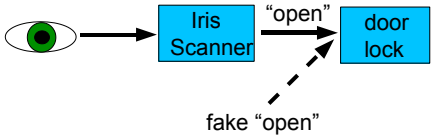
Authentication alternatives

- 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 principal, 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	…
User1	R	W	RW	
User2	RW	W	---	
User3	---	W	---	

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,privileged> pairs.**
    - **Example: AFS acls on directories (e.g. `john rl mary rlw`)**
- Concern: size of ACLs
  - **Allow groups or *role-based access control*.**
  - **Unix example: 9 bits: self, group, other.**

```
drwxr-xr-x 104 root wheel 8192 Aug4 04:02 /etc
```

  - **AFS groups:**

```
cs140-admins rlwidka system:anyuser il
```

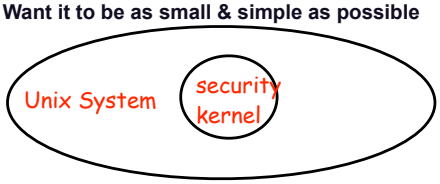
- Access control lists are simple, and are used in almost all file systems.

Capabilities

- With each users, indicate which objects 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 king. 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.**
  - **Shell script: "while(1) { mkdir foo; cd foo; }"**
  - **C program: "while(1) { fork(); malloc(1000)[40] = 1; }"**
- Listener:
  - **Passively watch network traffic.**
  - **Will see e-mail/web/passwords/etc.**
  - **Or just watch for file traffic: Often transmitted in plaintext.**

Other common attacks

- Look for open doors = unsecured data/systems
  - **Example: google search for "passwd", or for misconfigured/broken versions of software**
- Trojan Horse (Imposter)
  - **Example: leave a program around that looks like the login process:**
    - **Fake ATM.**
    - **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.**

Exploiting leaks and bugs

- 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.**
- How to find? Look! Also: Fuzzing, Patches, Bug lists.

Hiding malware: 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 impostor 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 been 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, or (worse?) virtual machine/firmware rootkits
  - **rootkit: change system calls, file system, etc. to hide malware (e.g. doesn't show up with `ps` or `ls`)**
  - **VMM/firmware rootkit: 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/impossible to do: Example: Containment problem**
- Correctness proofs: very hard to do. Only proves 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 does: any program you run could conceivably modify any of your files.

Next: Real-world OS Security themes

- Protection/security very hard to get right
- The design, mechanisms, and interface (API and user interface) provided by the OS can make it easier or harder for the OS and programs to prevent protection violations and provide security and privacy
- Feature interaction is powerful for both good and evil!
  - advantage: do things designers never intended
  - disadvantage: do things designers never intended!
- Current operating systems (and other software)have largely failed at security
- Certain problems (containment) are almost impossible to solve

Example: Unix protection architecture

- Each process has user, group ID (uid, gid)
- Files, directories have permission bits

```
drwxr-xr-x 104 root wheel 8192 Aug4 04:02 /etc
```

- Devices, other objects exported through file system

  `/dev/tty1`, `/dev/disk0s1`, `/proc/xxxx`
- Non-FS object policies? restrict to `root` (uid 0) user
  - Bind any TCP/UDP port < 1024
  - Change current process's uid or gid
  - Mount/unmount file system
  - Create device nodes (/dev/xxxx) in file system
  - Change owner of file
  - Set clock; halt or reboot machine

Example: Unix `/bin/login`

- reads files from `/etc`
  - `passwd`, `group`, `shadow/master.passwd`
  - `username:hashpw:uid:gid:real name:dir:shell`
  - hashpw = one-way hash of (passwd,salt)
- runs as `root`
  - reads username and password from terminal
  - looks up in `/etc/passwd`
  - makes sure hash(passwd,salt) matches
  - if so, set user and group id, and `exec(shell)`
- Unix: hybrid of ACL (permissions) + very coarse Capability (`root` = can do anything to anything!)

Unix workaround: Setuid

- Some legitimate actions require more privs than UID
  - how should users change their passwords?
  - stored in `/etc/passwd` and `/etc/shadow`
- Solution: setuid/setgid programs
  - run with privileges of file's owner or group
  - *real* and *effective* UID/GID
  - real: actual user; effective: for access checks
  - e.g. /usr/bin/passwd – changes user password,
  - /bin/su – acquire new UID with correct password
- Have to be very careful writing setuid code
  - attackers can run setuid programs at any time
  - attacker controls many aspects of program's environment

Unix: sample setuid problems

- setuid shell scripts?
  - not if attacker controls $PATH!!
  - `/tmp/evil/bin:/bin:/usr/bin`
  - don't put "." in your path either
- setuid programs with shared libraries?
  - not if attacker controls $LD_LIBRARY_PATH
  - `/tmp/evil/lib:/lib:/usr/lib`
- setuid programs with debuggers?
  - not if attacker can attach to program, change code/data
  - `gdb /bin/passwd`? `ptrace()` call
- result: all kinds of bad interactions that you need to find!
  - did you find them all?