Security and Operating Systems

Security and Operating Systems

- What is operating system security?
- How do operating systems contribute to system security?
- Alternatively, if we're trying to develop a secure system, what do we demand of the OS?
- Today's lecture concentrates on what the OS can/should/does do

What is Security?

- Informal: *Security is keeping unauthorized entities from doing things you don't want them to do.*
- More formal: Confidentiality, integrity, availability
- What is the operating system's role?

Internal Roles

- We've discussed a lot of internal features: privileged mode, memory protection, file access permissions, etc.
- What do these accomplish?
- What is the *real* goal?

Protecting Whom?

- Internal features protect the operating system against users
- This necessary but not sufficient
- File permissions protect users (and the OS) against other users
- Again, this is necessary but not sufficient

User Authentication

- File permissions are based on user identity, which is based on authentication
- How does an OS authenticate users?
- Many methods: something you know, something you have, something you are

Something You Know: Passwords

- Very common
- Very easily guessed
- Originally stored in plaintext, but that's a very bad idea
- Today, passwords are usually stored hashed
- However — some network authentication schemes, such as challenge/response, require plaintext (or equivalent)

Hashed Passwords

- Store $f(\text{PW})$, where f is not invertible
- When user enters PW, calculate $f(\text{PW})$ and compare
- To guard against precomputation attacks, assign a random *salt* at password change time and store $\langle \text{salt}, f(\text{PW,salt}) \rangle$
- Attackers can still run password-guessing programs, so most operating systems use access control to protect the hashed passwords

Challenge/Response Authentication

- Ask the user questions no one else would know the answer to
- Note: your mother's maiden name doesn't qualify!
- If the user has calculating ability, do it cryptographically
- The server knows PW and sends a random number N
- Both sides calculate $f(\text{PW}, N)$, where f is something like an encryption algorithm
- Note that an eavesdropper who sees N and $f(\text{PW}, N)$ can still do password-guessing

THe Human Element

"Humans are incapable of securely storing high-quality cryptographic keys, and they have unacceptable speed and accuracy when performing cryptographic operations. They are also large, expensive to maintain, difficult to manage, and they pollute the environment. It is astonishing that these devices continue to be manufactured and deployed, but they are sufficiently pervasive that we must design our protocols around their limitations."
—*Network Security: Private Communication in a Public World*

Something You Have: Tokens

- Some sort of (generally tamper-resistant) device
- Perhaps it does the challenge/response
- A very popular one has a cryptographic secret and the time of day, and calculates $f(K, T)$
- Often combined with a PIN, to guard against loss or theft

Something You Are: Biometrics

- Fingerprint readers are becoming common
- Iris scans are probably more secure
- Biometrics generally have a fale positive rate and a false negative rate — be careful how you set your parameters!
- Biometrics works best if stored locally — over the network, all that's seen is a string of bits
- Watch out for spoofing attacks

Attack Techniques

- Trojan horses — "come and get it" attack
- Login spoofing
- Buggy software — the big one

Trojan Horses

- Trick someone into executing a program that does nasty things
- (Many viruses and worms spread that way)
- How can the OS protect users?
- Unix-type file permissions don't help — the attack program can change permissions
- Need mandatory access control (MAC)

Sandboxes

- A better idea is for the OS to provide *sandboxes* — an enviornment where the program can execute but can't affect the rest of the machine
- Strong isolation is conceptually pretty easy — run the program on a separate machine, or under VMware or the like
- There are other, more elegant mechanisms that attempt to provide the same feature at lower cost; most are limited to root
- The trick — and it's a very difficult one — is permitting limited interaction with the outside world while still protecting security

Race Conditions

- Race conditions can be a security issue
- Consider a privileged program that checks if a file is readable and then tries to open it as root
- The attacker can pass it a symlink; in the interval between the two operations, the attacker removes the symlink and replaces it with a link to a protected file
- The OS must provide (and the application must use) atomic operations to open the file as that user

Login Spoofing


```
NetBSD/i386 (clic) (tty00)

login:
```

- Is that the real login prompt?
- A fake one could capture your login and password
- (We see similar things today with fake ATMs!)

Trusted Path

- A *trusted path* is a user-initiated sequence that is guaranteed to get you to the real OS
- Example: cntl+alt+delete on Windows
- Well, it was supposed to be one...
- But — you have to train people not to log in unless they've initiated the sequence
- Must protect all password prompts that way

Viruses and Worms

- *Viruses* spread by themselves within a machine, but require human intervention to infect other machines
- *Worms* spread between machines, though they may require human assistance (i.e., opening an attachment) to infect another machine
- What can the OS do to stop these?

Access Controls Won't Do It

- One sometimes hears that "Windows is infested with these things because it has no (effective) file protection"
- File protection would prevent OS contamination, but worms can and do spread with user permissions
- The IBM Christmas Card "Virus" (1987) relied on a Trojan Horse emailed shell script
- The Morris Internet Worm (1988) was multi-exploit, multi-platform and didn't violate any OS protections

Blocking Executables

- Operating systems can tried to block suspicious content
- It's very hard to do — lots of ways to sneak stuff in
- Windows XP SP2 "tags" downloaded files — anything that's tagged is deemed non-executable
- But what about things like bug fixes, that you should permit to be downloaded?

Certified Systems

- In the early 1980s, the U.S. Defense Department created the so-called *Orange Book* (*DoD Trusted Computer System Evaluation Criteria*) and its companions
- The Orange Book described a set of secure system levels, from **D** (no security) to **A1** (formally verified)
- The higher levels had more features; more importantly, they had higher *assurance*

Military Classification Model

- Documents are classified at a certain level
- People have certain clearances
- You're only allowed to see documents that you're cleared for

Classifications

- Levels: Confidential, Secret, Top Secret
- Compartments: Crypto, Subs, NoForn
- ("NoForn" is "No foreign nationals")
- To read a document, you must have at least as high a clearance level *and* you must be cleared for each compartment
- Systems that support this are known as *multi-level security systems*

Examples

Pat is cleared for **Secret**, *Subs*
Chris is cleared for **Top Secret**, *Planes*
We have the following files:

warplan	**Top Secret**	*Troops, Subs, Planes*
runway	**Confidential**	*Planes*
sonar	**Top Secret**	*Subs*
torpedo	**Secret**	*Subs*

Who can read which file?

Examples

- Pat cannot read `warplan`; she isn't cleared high enough and she doesn't have *Troops* or *Planes* clearance
- Chris can't read it, either; he doesn't have *Subs* or *Planes* clearance
- Chris can read `runway`; Pat can't
- Pat can't read `sonar`; she has *Subs* clearance but only at the **Secret** level
- She can, however, read `torpedo`

Assurance

- Why do you think the OS is correct?
- How good a job did the developers do?
- Were back doors inserted by nasty developers?
- Higher levels of the Orange Book demanded more assurance, including good documentation, structured design, formal test plans, and security-cleared programmers

The Fate of the Orange Book

- The British, and later Western Europe, produced the *Information Technology Security Evaluation Criteria*
- The Canadians produced their version
- The US produced the *Federal Criteria*
- They were all merged into the *Common Criteria*

The Common Criteria

- The Orange Book mixed features and assurance
- The Common Criteria separated those
- It also permitted varying *Protection Profiles*
- The Protection Profile says what it's supposed to accomplish
- The feature list says how it does that
- The assurance level is how confident you should be

Bad Models, No Sales

- The Orange Book's security model wasn't interesting commercially
- Getting a system evaluated was time-consuming and expensive
- Evaluated systems lagged well behind commercial ones, and were more expensive
- It was designed for 1980s-style timesharing systems
- "Minor" things like networking weren't included...
- Many of the same problems affect Common Criteria systems

Logging

- What's going on on your systems?
- If there's a penetration, will you know?
- Will you be able to figure out how it happened?

Shadow Hawk

Shadow Hawk Busted Again
As many of you know, Shadow Hawk (a/k/a Shadow Hawk 1) had his home searched by agents of the FBI...
When he was tagged by the feds, he had been downloading software (in the form of C sources) from various AT&T systems. According to reports, these included the Bell Labs installations at Naperville, Illinois and Murray Hill, New Jersey.

—Phrack Issue 16, File 11, November 1987

How was Shadow Hawk Detected?

- He had broken into some Bell Labs machines
- He tried to use `uucp` — a dial-up file transfer/email system that came with Unix — to grab `/etc/passwd` files from other machines
- Uucp logged all file transfer requests
- Several people at Murray Hill had automated jobs that scanned the log files for anything suspicious

What to Log?

- Everything?
- Possibly takes too much storage, though disk space is cheap
- Serious potential privacy risk
- Can you process that much data?
- But — *must* log security-sensitive events

Solaris Basic Security Module (BSM)

- BSM can do a lot of logging
- Some categories: login/logout, ioctl, file write, network events, mount/unmount, fork, exec, and more
- Great care taken to protect log files
- Again, though — can you process the data?

It's the Application

- The real purpose of an operating system is to run certain applications
- The issue isn't how secure the OS is, it's how secure the applications are
- Again, most worms don't violate OS security

The Challenge

- A *useful* secure OS should make it easier to write secure applications
- That means things like *useful* sandboxes
- Need more flexible permission model; DAC is too simple and MAC is too restrictive

We Don't Have Them

- In my opinion, *no* commercial OS satisfies these criteria
- Of course, most applications are designed for the facilities we do have
- We'll always have buggy code; the trick is to build an application and an OS that will *mostly* resist attack and will protect the important assets