Test Conditions

- Same as the midterm: closed book, no laptop, etc.
- Roughly 1/3 from the first half, 2/3 from the second half or combined questions
- Remember to review readings
- Same style of questions

Primary Themes

- Access control
- Structure
- Combining mechanisms

Changing Passwords

- Controls all system access — very sensitive
- **`/etc/passwd`** must be world-readable — legacy effect
- Use read permission to protect hashed password
- Root has too much power via **`/etc/passwd`**
- Features used: access control, locking, authentication, setUID, filtering

Web Servers

- Large, complex programs
- Serve files, run scripts, run user-written programs
- How does access control work?

Complex Application-Specific Access Control

- ACLs for subtrees, passwords, IP addresses, more
- OS access controls
- Access to “privileged” ports
- Does the code and/or the administrator get all this right?

OS Permissions

- Server starts as root, but runs as **`www`**
- Files served are readable, but not owned, by **`www`**
- OS permission mechanisms protect the system against the web server

Scripts

- Scripts are programs, and hence can be buggy
- What permissions do scripts have?
- Plug-ins run with Apache's permissions
- On-machine attacks can bypass script permissions

Confining an Application

- Protect resources — files, CPU time, memory,disk space, network identity, network access rights
- Protect some with OS mechanisms
- Can’t do things as well as we’d like; in particular, hard to permit easy access to this for all applications

Chroot()

- Confine process to a subtree
- Only useable by root
- Vulnerable to root compromise within the confined application
- Not easy to set up

Sandboxes

- Janus — traps system calls
- Java VM — relies on properties of Java language, plus verification by byte code verifier and class loader
- Virtual machines

Virtual Machines

- Emulate real machine
- Trap privileged operation; map to user's resources: virtual disk, virtual keyboard, virtual Ethernet, etc.
- Has strengths and weaknesses of a real machine
- Good for analyzing malware — but some such programs detect it

Covert Channels

- Subtle way of passing information to violate MAC
- Storage and timing channels
- Noisy — use error-correcting codes

Malware – Viruses

- Difference between viruses, worms, and Trojan horses
- Program, boot sector, and macro viruses
- Scanner, replicator, payload
- A-V software
- Encrypted and polymorphic viruses
- Viruses vs. DAC and MAC

Trojan Horses

- Functions
- Spreading patterns
- “Legal” ones?

Back Doors

- Ken Thompson’s C compiler trick
- Eric Allman’s Sendmail back door
- Source repositories

Program Structure

- Strive for bug-resistance: inherently safe(r) software
- Program structure has a major impact
- Minimize the chances of a bug; minimize the impact

Strategies

- Separate security-critical sections
- Use strong isolation between such (small) sections and the rest

FTPD

- Uses YACC grammar to parse (simple) input
- YACC is fine, but the grammar is poorly structured
- Two-command sequences aren't recognized by the grammar; possible to get other commands in between
- Hole due to static buffers in **`getpwnam()`**
- Proper fix 1: restructure grammar
- Proper fix 2: split out login sequence into separate program

Designing an E-Commerce Site

- Real *systems* are composed of many components
- Separation and connectivity count
- Many danger points

Danger Points

- Component management
- Link to back-end systems
- NOCs need access to everything
- Customer care
- Backups
- Emergency operations

The Database

- Most vital component
- Isolate to separate machine; use end-to-end authentication
- Actually, have several databases
- Limit information flow

Log Files

- Easiest way to figure out what happened (and maybe if something happened)
- Logs produced by many different components
- Need log file correlator
- Log files need to be protected
- Need automated log file scanner

Analyzing a System

- Multiple levels of detail
- Program audits — look for usual error situations
- Use audit tools — **`grep`** isn't sufficient
- Really need flow analysis

Higher-level Audits

- Look for separate elements, flows, barriers, untrusted inputs
- Who talks to whom? How?
- What sorts of authentication and filtering are used?
- Different problem severities

Attacking

- Tiger teams — benefits, limits, conditions
- Strace and ltrace
- Look for privileged operations, symbols, interesting strings

Higher-leval Attacks

- Infiltration, physical, social
- (Are these in scope for your tiger team?)
- Process helps
- Try to spot or block reconnaissance

Forensics

- Lots of information left lying around
- Hard to *really* delete data
- Main memory often has keys, plaintext