CIS3110 Lecture 14 Summary Notes: Security and Protection

The Multics Model and Unix Evolution (Security and Protection: 16.1, 16.2, 16.3, 17.1-17.9)

- **Multics Model**: Introduced the concept of permission escalation through "gates" (specific function entry points) where permission changes are carefully controlled
- **Unix Evolution**:
  - Unix developed as an outgrowth of the Multics project
  - Windows NT (and later versions) are derivatives of Unix through the Mach project
  - Most modern operating systems are based on these foundational designs

File System Permissions (Protection: 17.2, 17.3)

- **Unix File Permissions**: User, group, and other bit flags (read, write, execute)
  - Modified using the `chmod` (Change Mode) command
  - Represents user-based restricted domains
- **Hidden Files**:
  - In Unix: Files starting with a dot (.) are hidden from normal directory listings
  - In Windows: Hidden bit explicitly set on files

User Privileges and Root Access (Security: 16.2, Protection: 17.4, 17.5)

- **Root User in Unix**:
  - Has global permission to set or revoke any permissions
  - Can perform any system operation
  - Different from kernel privileges (supervisor bit)
- **Kernel vs. Root**:
  - Kernel runs before user space is initialized
  - Kernel has hardware-level access (can reconfigure memory, talk to devices)
  - Root is a user account with maximum permissions in user space
  - Even root needs to invoke the kernel through a trap to access hardware

Initialization Process (Security: 16.3)

- **Boot Sequence**:

1. Kernel loads into memory
2. Kernel probes devices, configures disks, checks file systems
3. "Goes multi-user" by starting the `init` process
4. `init` runs as root and spawns all other system processes

Permission Mechanisms (Protection: 17.6, 17.7, 17.8)

- **Set User ID (SUID) Bit**:
  - When set on an executable file, the program runs with the permissions of the file owner
  - Allows permission escalation in a controlled manner
  - Can only be set by the file owner or root
  - Common example: Web server running as user "www"
- **Daemon Processes**:
  - Processes whose parents have exited (no longer attached to command line)
  - Provide services under specific user accounts
  - Communication through sockets (network interfaces)
- **Chroot and Jail**:
  - `chroot`: Changes the root directory for a process, restricting file system access
  - `jail` (BSD): Extended version of chroot that also restricts network access, programs, and users
  - Lightweight compared to full virtualization (Docker)
- **Page Permission Bits**:
  - Read-only bit: Prevents writing to a page
  - Execute bit: Prevents running code from data pages (helps prevent worms)
  - Random segment offsets: Makes it harder to locate the stack in memory

Windows Security Model (Security: 16.1, 16.3, Protection: 17.2)

- **Administrator vs. Root**:
  - Administrator is a set of properties applied to accounts (not a specific account)
  - Windows often requires administrator privileges for regular operations
  - Less clear separation between system and administrator accounts
  - Problematic approach to security: running everyday applications with full permissions
- **Windows Registry**:

- Hierarchical database for configuration settings
- Key prefixes include:
  - `HKEY_CURRENT_USER`: User-specific settings
  - `HKEY_LOCAL_MACHINE`: Machine-specific settings
- Most keys protected from non-administrator users
- Source of many permission issues with developer tools

Access Control Models (Protection: 17.3, 17.4)

- **Access Matrix**:
  - Maps permission domains to resources and allowed operations
  - Also defines which domains can switch to other domains
  - Can specify data transfer policies (copy, ownership transfer, limited copy)
- **Access Lists**:
  - Alternative to matrices that saves space but requires linear search
  - List of values associated with each domain

Security Implementation Challenges (Security: 16.2, 16.3)

1. **Confinement Problem**:
   - Difficulty preventing data escape once code can access a domain
   - Root cause of many data breaches
2. **Super User Issue** (God Mode Problem):
   - Single point of failure where one user has complete control
   - Anyone gaining root/administrator access bypasses all security measures
3. **Messaging Security**:
   - All I/O creates potential eavesdropping points
   - Even kernel-mediated communication has trust assumptions
   - No guaranteed secure communication channel
4. **Trust Definition Problem**:
   - Users must accept the system administrator's definition of trust
   - In cloud environments, users have little control over the trust environment

Security Solutions (Security: 16.1, 16.2, 16.3)

- **Cryptography**:
  - Encodes data using ciphers (algorithms) and keys
  - Types:
    - Symmetric: Same key used to encrypt and decrypt
    - One-way/Hashing: Used for passwords, cannot be decrypted
  - Password storage: Uses one-way encryption with "salt" to prevent dictionary attacks
  - Shadow password files separate authentication data from user info
- **Java Security Model**:
  - Virtual machine provides a controlled execution environment
  - Features:
    - Method isolation
    - Entry/exit data inspection
    - Private heap allocation
    - Class-specific security domains
    - Applet sandbox (restrictive environment for web code)
    - Type safety requirements
    - Stack inspection
    - Security policy enforcement
  - Permission model:
    - `doPrivileged` blocks for controlled escalation
    - Permission checks verify both permission ownership and awareness
    - Only authorized code can request higher privileges