May 19: Design Principles and Confinement

- Principles of Secure Design
  - One set; other sets say basically the same thing

- Confinement, non-VM isolation
  - Library operating systems
  - Sandboxes
  - Program modification
  - Covert channels
Basis of Design Principles

• Simplicity
  – Less to go wrong
  – Fewer possible inconsistencies
  – Easy to understand

• Restriction
  – Minimize access
  – Inhibit communication
Least Privilege

• A subject should be given only those privileges necessary to complete its task
  – Function, not identity, controls
  – Rights added as needed, discarded after use
  – Minimal protection domain
Related: Least Authority

- **Principle of Least Authority (POLA)**
  - Often considered the same as Principle of Least Privilege
  - Some make distinction:
    - *Permissions* control what subject can do to an object directly
    - *Authority* controls what influence a subject has over an object (directly or indirectly, through other subjects)
Fail-Safe Defaults

• Default action is to deny access
• If action fails, system as secure as when action began
Economy of Mechanism

- Keep it as simple as possible
  - KISS Principle
- Simpler means less can go wrong
  - And when errors occur, they are easier to understand and fix
- Interfaces and interactions
Complete Mediation

• Check every access
• Usually done once, on first action
  – UNIX: access checked on open, not checked thereafter
• If permissions change after, may get unauthorized access
Open Design

• Security should not depend on secrecy of design or implementation
  – Popularly misunderstood to mean that source code should be public
  – “Security through obscurity”
  – Does not apply to information such as passwords or cryptographic keys
Separation of Privilege

• Require multiple conditions to grant privilege
  – Separation of duty
  – Defense in depth
Least Common Mechanism

• Mechanisms should not be shared
  – Information can flow along shared channels
  – Covert channels

• Isolation
  – Virtual machines
  – Sandboxes
Least Astonishment

- Security mechanisms should be designed so users understand why the mechanism works the way it does, and using mechanism is simple
  - Hide complexity introduced by security mechanisms
  - Ease of installation, configuration, use
  - Human factors critical here
Related: Psychological Acceptability

• Security mechanisms should not add to difficulty of accessing resource
  – Idealistic, as most mechanisms add *some* difficulty
    • Even if only remembering a password
  – Principle of Least Astonishment accepts this
    • Asks whether the difficulty is unexpected or too much for relevant population of users
Key Points

• Principles of secure design underlie all security-related mechanisms

• Require:
  – Good understanding of goal of mechanism and environment in which it is to be used
  – Careful analysis and design
  – Careful implementation
Library Operating Systems

- Often process can optimize use of system resources better than the operating system.
- Goal is to move as much of operating system as is feasible to user level.
  - This minimizes context switches.
  - It maximizes process flexibility.
Library Operating Systems

• It’s a library(-ies) that provide operating system functionality at the user level

• Develop kernel that:
  – Uses hardware protection to prevent processes from accessing memory space of others
  – Controls access to physical resources that must be shared by executing processes
  – Everything else is in user space
Example

• V++ Cache Kernel tracks OS that are in use
  – Also handles process co-ordination

• Page faults
  – Application kernel loads new page mapping descriptor into Cache Kernel
Example

- Exokernel separates resource protection, resource management
- Aegis: small kernel providing interface to hardware resources
- ExOS: interface to Aegis that enables process to use resources appropriately
  - Also provides resource protection
Drawbridge

- Library OS, security monitor for Windows
  - Security monitor provides interface to underlying OS
- Processes use library OS to access security monitor interface
  - All interactions go through it
  - Library OS also provides application services (frameworks, rendering engines)
Drawbridge

• Handles kernel dependencies using emulator at lowest level of library OS
  – So all server dependencies, Windows subsystems moved into user layer
  – User interaction by emulated device drivers that tunnel I/O between desktop, security monitor

• Processes isolated from one another
Drawbridge Validation

- Malware deleting all registry keys affected only that process
- Keystroke logger captured keystrokes only for that process
- Attacks causing Internet Explorer to escape normal (protected) mode all mitigated
Sandboxes

• An environment in which actions are restricted in accordance with security policy
  – Limit execution environment as needed
    • Program not modified
    • Libraries, kernel modified to restrict actions
  – Modify program to check, restrict actions
    • Like dynamic debuggers, profilers
Example: Janus

• Implements sandbox in which system calls checked
  – Framework does runtime checking
  – Modules determine which accesses allowed

• Configuration file
  – Instructs loading of modules
  – Also lists constraints
# basic module
basic

# define subprocess environment variables
putenv IFS=“\t\n “ PATH=/sbin:/bin:/usr/bin TZ=PST8PDT

# deny access to everything except files under /usr
path deny read,write *
path allow read,write /usr/*
# allow subprocess to read files in library directories
# needed for dynamic loading
path allow read /lib/* /usr/lib/* /usr/local/lib/*
# needed so child can execute programs
path allow read,exec /sbin/* /bin/* /usr/bin/*
How It Works

- Framework builds list of relevant system calls
  - Then marks each with allowed, disallowed actions

- When monitored system call executed
  - Framework checks arguments, validates that call is allowed for those arguments
    - If not, returns failure
    - Otherwise, give control back to child, so normal system call proceeds
Use

• Reading MIME Mail: fear is user sets mail reader to display attachment using Postscript engine
  – Has mechanism to execute system-level commands
  – Embed a file deletion command in attachment …

• Janus configured to disallow execution of any subcommands by Postscript engine
  – Above attempt fails
Examples Limiting Environment

- **Java virtual machine**
  - Security manager limits access of downloaded programs as policy dictates

- **Sidewinder firewall**
  - Type enforcement limits access
  - Policy fixed in kernel by vendor

- **Domain Type Enforcement**
  - Enforcement mechanism for DTEL
  - Kernel enforces sandbox defined by system administrator
Program Modification

• Idea is to change program itself to comply with a stated security policy
• Program can be rewritten to embed constraints in it
• Compiler can apply constraints as program being compiled
  – Same for interpreter
• Loader can apply constraints as program is loaded for execution
Rewriting Program

Software fault isolation

• Untrusted modules go into virtual segments
• Flow of control remains in the segment
• All memory accesses from within the segment go to locations within the segment
Implementations

• Put each module in separate segment
  – *Unsafe instruction* access address that can’t be verified to be in the segment

• Statically analyze program, identify all unsafe instructions

• When executed, check address is in segment
  – Check segment identifier of (virtual) address
  – Replace segment identifier of (virtual) address with identifier of the segment
System Calls

- In untrusted modules, could pose problems
  - Close an open file trusted module depends on
  - So replace system calls with calls to arbitration code in its own segment
  - Arbitration code determines whether to invoke system call

- Alternative: trusted, untrusted processes
  - Trusted process handles all security-sensitive accesses