Lecture 14
October 21, 2022
Ring-Based Access Control

- Process (segment) accesses another segment
  - read
  - execute
- Gate is an entry point for calling segment
- Rights:
  - r read
  - w write
  - a append
  - e execute
Reading/Writing/Appending

• Procedure executing in ring $r$
• Data segment with access bracket $(a_1, a_2)$
• Mandatory access rule
  • $r \leq a_1$ allow access
  • $a_1 < r \leq a_2$ allow $r$ access; not $w, a$ access
  • $a_2 < r$ deny all access
Executing

• Procedure executing in ring $r$

• Call procedure in segment with *access bracket* $(a_1, a_2)$ and *call bracket* $(a_2, a_3)$
  • Often written $(a_1, a_2, a_3)$

• Mandatory access rule
  • $r < a_1$ allow access; ring-crossing fault
  • $a_1 \leq r \leq a_2$ allow access; no ring-crossing fault
  • $a_2 < r \leq a_3$ allow access if through valid gate
  • $a_3 < r$ deny all access
Versions

• Multics
  • 8 rings (from 0 to 7)

• Intel’s Itanium chip
  • 4 levels of privilege: 0 the highest, 3 the lowest

• Older systems
  • 2 levels of privilege: user, supervisor
Locks and Keys

• Associate information (*lock*) with object, information (*key*) with subject
  • Latter controls what the subject can access and how
  • Subject presents key; if it corresponds to any of the locks on the object, access granted

• This can be dynamic
  • ACLs, C-Lists static and must be manually changed
  • Locks and keys can change based on system constraints, other factors (not necessarily manual)
Cryptographic Implementation

- Enciphering key is lock; deciphering key is key
  - Encipher object $o$; store $E_k(o)$
  - Use subject's key $k'$ to compute $D_k(E_k(o))$
  - Any of $n$ can access $o$: store
    $$o' = (E_1(o), ..., E_n(o))$$
  - Requires consent of all $n$ to access $o$: store
    $$o' = (E_1(E_2(...(E_n(o))...))$$
Example: IBM

- IBM 370: process gets access key; pages get storage key and fetch bit
  - Fetch bit clear: read access only
  - Fetch bit set, access key 0: process can write to (any) page
  - Fetch bit set, access key matches storage key: process can write to page
  - Fetch bit set, access key non-zero and does not match storage key: no access allowed
Example: Cisco Router

• Dynamic access control lists
  
  ```
  access-list 100 permit tcp any host 10.1.1.1 eq telnet
  access-list 100 dynamic test timeout 180 permit ip any host 10.1.2.3 time-
  range my-time
  time-range my-time
    periodic weekdays 9:00 to 17:00
  line vty 0 2
  login local
  autocommand access-enable host timeout 10
  ```

• Limits external access to 10.1.2.3 to 9AM–5PM
  • Adds temporary entry for connecting host once user supplies name, password to router
  • Connections good for 180 minutes
    • Drops access control entry after that
Type Checking

• Lock is type, key is operation
  • Example: UNIX system call *write* won’t work on directory object but does work on file
  • Example: split I&D space of PDP-11
  • Example: countering buffer overflow attacks on the stack by putting stack on non-executable pages/segments
    • Then code uploaded to buffer won’t execute
    • Does not stop other forms of this attack, though …
More Examples

• LOCK system:
  • Compiler produces “data”
  • Trusted process must change this type to “executable” before program can be executed

• Sidewinder firewall
  • Subjects assigned domain, objects assigned type
    • Example: ingress packets get one type, egress packets another
  • All actions controlled by type, so ingress packets cannot masquerade as egress packets (and vice versa)
Sharing Secrets

• Implements separation of privilege

• Use \((t, n)\)-threshold scheme
  • Data divided into \(n\) parts
  • Any \(t\) parts sufficient to derive original data

• Or-access and and-access can do this
  • Increases the number of representations of data rapidly as \(n, t\) grow
  • Cryptographic approaches more common
Shamir’s Scheme

• Goal: use $(t, n)$-threshold scheme to share cryptographic key encoding data
  • Based on Lagrange polynomials
  • Idea: take polynomial $p(x)$ of degree $t-1$, set constant term ($p(0)$) to key
  • Compute value of $p$ at $n$ points, *excluding* $x = 0$
  • By algebra, need values of $p$ at any $t$ distinct points to derive polynomial, and hence constant term (key)
Reference Monitor

• *Reference monitor* is access control concept of an abstract machine that mediates all accesses to objects by subjects

• *Reference validation mechanism* (RVM) is an implementation of the reference monitor concept.
  • Tamperproof
  • Complete (always invoked and can never be bypassed)
  • Simple (small enough to be subject to analysis and testing, the completeness of which can be assured)
    • Last engenders trust by providing evidence of correctness

• Note: RVM is almost always called a reference monitor too
Examples (Or, What Should Be Examples)

- *Security kernel* combines hardware and software to implement reference monitor

- *Trusted computing base (TCB)* consists of all protection mechanisms within a system responsible for enforcing security policy
  - Includes hardware and software
  - Generalizes notion of security kernel
Policy and Reference Monitor

- Reference monitor implements a given policy
  - It has a tamperproof authorization database
  - Also maintains an audit trail (record of security-related events) for review
Example Problem

• Server balances bank accounts for clients

• Server security issues:
  • Record correctly who used it
  • Send only balancing info to client

• Client security issues:
  • Log use correctly
  • Do not save or retransmit data client sends
Generalization

• Client sends request, data to server
• Server performs some function on data
• Server returns result to client
• Access controls:
  • Server must ensure the resources it accesses on behalf of client include *only* resources client is authorized to access
  • Server must ensure it does not reveal client’s data to any entity not authorized to see the client’s data
Confinement Problem

• Problem of preventing a server from leaking information that the user of the service considers confidential
Total Isolation

• Process cannot communicate with any other process
• Process cannot be observed

Impossible for this process to leak information
  • Not practical as process uses observable resources such as CPU, secondary storage, networks, etc.
Example

• Processes $p$, $q$ not allowed to communicate
  • But they share a file system

• Communications protocol:
  • $p$ sends a bit by creating a file called 0 or 1, then a second file called $send$
    • $p$ waits until $send$ is deleted before repeating to send another bit
  • $q$ waits until file $send$ exists, then looks for file 0 or 1; whichever exists is the bit
    • $q$ then deletes 0, 1, and $send$ and waits until $send$ is recreated before repeating to read another bit
Covert Channel

- A path of communication not designed to be used for communication
- In example, file system is a (storage) covert channel
Rule of Transitive Confinement

• If $p$ is confined to prevent leaking, and it invokes $q$, then $q$ must be similarly confined to prevent leaking

• Rule: if a confined process invokes a second process, the second process must be as confined as the first
Isolation

- Constrain process execution in such a way it can only interact with other entities in a manner preserving isolation
  - Hardware isolation
  - Virtual machines
  - Library operating systems
  - Sandboxes
- Modify program or process so that its actions will preserve isolation
  - Program rewriting
  - Compiling
  - Loading
Hardware Isolation

• Ensure the hardware is disconnected from any other system
  • This includes networking, including wireless

• Example: SCADA systems
  • 1st generation: serial protocols, not connected to other systems or networks; no security defenses needed, focus being on malfunctions
  • 2nd generation: serial networks connected to computers not connected to Internet
  • 3rd generation: TCP/IP protocol running on networks connected to Internet; need security defenses for attackers coming in over Internet

• Example: electronic voting systems
  • Physical isolation protects systems from attackers changing votes remotely
  • Required in many U.S. states, such as California: never connect them to any network
Virtual Machine

• Program that simulates hardware of a machine
  • Machine may be an existing, physical one or an abstract one
  • Uses special operating system, called *virtual machine monitor (VMM)* or *hypervisor*, to provide environment simulating target machine

• Types of virtual machines
  • Type 1 hypervisor: runs directly on hardware
  • Type 2 hypervisor: runs on another operating system

• Existing OSes do not need to be modified
  • Run under VMM, which enforces security policy
  • Effectively, VMM is a security kernel
VH$_i$ is virtual machine $i$

T2H$_i$ is type-2 hypervisor $i$

<table>
<thead>
<tr>
<th>VH$_A$</th>
<th>VH$_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>VH$_5$</td>
<td>VH$_6$</td>
</tr>
<tr>
<td>T2H$_1$</td>
<td>T2H$_2$</td>
</tr>
</tbody>
</table>

Type-1 Hypervisor

Operating System

Physical Hardware
VMM as Security Kernel

• VMM deals with subjects (the VMs)
  • Knows nothing about the processes within the VM
• VMM applies security checks to subjects
  • By transitivity, these controls apply to processes on VMs
• Thus, satisfies rule of transitive confinement
Example: Xen Hypervisor

• Xen 3.0 hypervisor on Intel virtualization technology
• Two modes, VMX root and non-root operation
• Hardware-based VMs (HVMs) are fully virtualized domains, support unmodified guest operating systems and run in non-root operation mode
  • Xen hypervisor runs in VMX root mode
• 8 levels of privilege
  • 4 in VMX root operation mode
  • 4 in VMX root operation mode
  • No need to virtualize one of the privilege levels!
Xen and Privileged Instructions

• Guest operating system executes privileged instruction
  • But this can only be done as a VMX root operation
• Control transfers to Xen hypervisor (called VM exit)
• Hypervisor determines whether to execute instruction
• After, it updates HVM appropriately and returns control to guest operating system (called VM entry)
Problem

• Physical resources shared
  • System CPU, disks, etc.

• May share logical resources
  • Depends on how system is implemented

• Allows covert channels
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: Capsicum

• Framework developed to sandbox an application
• Capability provides fine-grained rights for accessing, manipulating underlying file
• To enter sandbox (capability mode), process issues cap_enter
• Given file descriptor, create capability with cap_new
  • Mask of rights indicates what rights are to be set; if capability exists, mask must be subset of rights in that capability
• At user level, library provides interface to start sandboxed process and delegate rights to it
  • All nondelegated file descriptors closed
  • Address space flushed
  • Socket returned to creator to enable it to communicate with new process
Example: Capsicum (con’t)

• Global namespaces not available
  • So system calls that depend on that (like `open(2)`) don’t work
    • Need to use a modified `open` that takes file descriptor for containing directory
  • Other system calls modified appropriately
    • System calls creating memory objects can create anonymous ones, not named ones (as those names are in global namespace)

• Subprocesses cannot escalate privileges
  • But a privileged process can enter capability mode

• All restrictions applied in kernel, not at system call interface
Program Confinement and TCB

• Confinement mechanisms part of trusted computing bases
  • On failure, less protection than security officers, users believe
  • “False sense of security”

• Must ensure confinement mechanism correctly implements desired security policy