Administrivia

- **Antonio and I will be out of town this week**
  - No office hours for me tomorrow
  - This week's only, section from Wed 7pm → Fri 1:45pm
  - Still 7th floor 719 Broadway
  - I'll be around after section to make up office hours
- **First lab has been posted, get accounts on class machines**
  - Fill out account request form Antonio is passing out

Clarification of notation

- **Simple notation for describing protocols**
- **With symmetric keys: $\{$message$\}_K$**
  - Let $(K_s, K_i) \leftarrow K$, $c \leftarrow \mathrm{Encrypt}(K_s, \mathrm{message})$
  - $\{\mathrm{message}\}_K$ means $\{c, \mathrm{MAC}(K_i, c)\}$
- **With public signature key K: $\{$message$\}_{K^{-1}}$**
  - means $\{\mathrm{message}, \mathrm{Sign}(K^{-1}, \mathrm{message})\}$
- **With public encryption key K: $\{$message$\}_K$**
  - means $\{\mathrm{Encrypt}(K, \mathrm{messgae})\}$

DAC vs. MAC

- **Most people familiar with discretionary access control (DAC)**
  - Example: Unix user-group-other permission bits
  - Might set a file `private` so only group `friends` can read it
- **Discretionary means anyone with access can propagate information:**
  - `Mail sigint@enemy.gov < private`
- **Mandatory access control**
  - Security administrator can restrict propagation
  - Abbreviated MAC (NOT a message authentication code)

Bell-Lapadula model

- **View the system as subjects accessing objects**
  - The system input is requests, the output is decisions
  - Objects can be organized in one or more hierarchies, H (a tree enforcing the type of decendents)
- **Four modes of access are possible:**
  - execute – no observation or alteration
  - read – observation
  - append – alteration
  - write – both observation and modification
- **The current access set, b, is (subj, obj, attr) tripples**
- **An access matrix M encodes permissible access types (subjects are rows, objects columns)**

Security levels

- **A *security level* is a (c, s) pair:**
  - $c =$ classification – E.g., unclassified, secret, top secret
  - $s =$ category-set – E.g., Nuclear, Crypto
- (c_1, s_1) ***dominates*** (c_2, s_2) **iff** $c_1 \geq c_2$ **and** $s_2 \subseteq s_1$
- **Subjects and objects are assigned security levels**
  - level(S), level(O) – security level of subject/object
  - current-level(S) – subject may operate at lower level
  - $f = (\text{level}, \text{level}, \text{current-level})$
- **State of system is 4-tuple** (b, M, f, H)

Security properties

- **The simple security or *ss-property*:**
  - For any $(S, O, A) \in b$, if A includes observation, then level(S) must dominate level(O)
  - E.g., an unclassified user cannot read a top-secret document
- **The star security or *-*property*:**
  - If a subject can observe O_1 and modify O_2, then level(O_2) dominates level(O_1)
  - E.g., cannot copy top secret file into secret file
  - More precisely, given $(S, O, A) \in b$:
    level(O) dominates current-level(S) if $A = a$
    level(O) is current-level(S) if $A = w$
    level(O) is dominated by current-level(S) if $A = r$

Straw man MAC implementation

- **Take an ordinary Unix system**
- **Put labels on all files and directories**
- **Each user has a security level**
- **Determine current security level dynamically**
  - When user logs in, start with lowest curent-level
  - Increase current-level as higher-level files are observed
  - If user's level does not dominate current, kill program
  - If program writes to file it doesn't dominate, kill it
- **Is this secure?**

No: Covert channels

- **System rife with *storage channels***
  - Low current-level process executes another program
  - New program reads sensitive file, gets high current-level
  - High program exploits covert channels to pass data to low
- **E.g., High program inherits file descriptor**
  - Can pass 4-bytes of information to low prog. in file offset
- **Other storage channels:**
  - Exit value, signals, terminal escape codes, …
- **If we eliminate storage channels, is system secure?**

No: Timing channels

- **Example: CPU utilization**
  - To send a 0 bit, use 100% of CPU is busy-loop
  - To send a 1 bit, sleep and relinquish CPU
  - Repeat to transfer more bits
- **Example: Resource exhaustion**
  - High prog. allocate all physical memory if bit is 1
  - Low program tries to allocate memory; if it fails, bit is 1
- **More examples: Disk head position, processor cache/TLB polution, …**

An approach to eliminating covert channels

- **Observation: Covert channels come from sharing**
  - If you have no shared resources, no covert channels
  - Extreme example: Just use two computers
- **Problem: Sharing needed**
  - E.g., read unclassified data when preparing classified
- **Approach: Strict partitioning of resources**
  - Strictly partition and schedule resources between levels
  - Occasionally reapportion resources based on usage
  - Do so infrequently to bound leaked information
  - In general, only hope to bound bandwidth covert channels
  - Approach still not so good if many security levels possible

Declassification

- **Sometimes need to prepare unclassified report from classified data**
- **Declassification happens outside of system**
  - Present file to security officer for downgrade
- **Job of declassification often not trivial**
  - E.g., Microsoft word saves a lot of undo information
  - This might be all the secret stuff you cut from document

Biba integrity model

- **Problem: How to protect integrity**
  - Suppose text editor gets trojaned, subtly modifies files, might mess up attack plans
- **Observation: Integrity is the converse of secrecy**
  - In secrecy, want to avoid writing less secret files
  - In integrity, want to avoid writing higher-integrity files
- **Use integrity hierarchy parallel to secrecy one**
  - Now only most privilegted users can operate at lowest integrity level
  - If you read less authentic data, your current integrity level gets raised, and you can no longer write low files

DoD Orange book

- **DoD requirements for certification of secure systems**
- **4 Divisions:**
  - D – been through certification and not secure
  - C – discretionary access control
  - B – mandatory access control
  - A – like B, but better verified design
  - Classes within divisions increasing level of security

Divisions C and D

- **Level D: Certifiably insecure**
- **Level C1: Discretionary security protection**
  - Need some DAC mechanism (user/group/other, ACLs, etc.)
  - TCB needs protection (e.g., virtual memory protection)
- **Level C2: Controlled access protection**
  - Finer-graunlarity access control
  - Need to clear memory/storage before reuse
  - Need audit facilities
- **Many OSes have C2-security packages**
  - Is, e.g., C2 Solaris "more secure" than normal Solaris?

Division B

- **B1 - Labeled Security Protection**
  - Every object and subject has a label
  - Some form of reference monitor
  - Use Bell-LaPadula model and some form of DAC
- **B2 - Structured Protection**
  - More testing, review, and validation
  - OS not just one big program (least priv. within OS)
  - Requires covert channel analysis
- **B3 - Security Domains**
  - More stringent design, w. small ref monitor
  - Audit required to detect imminent violations
  - requires security kernel + 1 or more levels *within* the OS

Division A

- **A1 – Verified Design**
  - Design must be formally verified
  - Formal model of protection system
  - Proof of its consistency
  - Formal top-level specification
  - Demonstration that the specification matches the model
  - Implementation shown informally to match specification

Limitations of Orange book

- **How to deal with floppy disks?**
- **How to deal with networking?**
- **Takes too long to certify a system**
  - People don't want to run n-year-old software
- **Doesn't fit non-military models very well**
- **What if you want high assurance & DAC?**

DEC VMM security kernel

- **Goal: Build production-quality A1 system**
- **One approach: Build emulator for VMS OS**
  - As hard as building the VMS operating system itself
  - Won't evolve with VMS
  - Want to run applications written for other OSes like Ultrix
- **Alternative: Build VM monitor that emulates hardware**
  - Hardware interface simpler than OS, and evolves less
  - Can then run multiple OSes on top of VMM

Virtual machines

- **General idea – Exploit hardware protection:**
  - VMM runs in most privileged hardware mode
  - OS runs on top of VMM in less privileged mode
  - VMM catches and fixes anything the OS does that would:
    - Detect that it is not running in the highest security level
    - Access a raw hardware device
    - Violate the security of the VMM
- **Vax protection: user, supervisor, executive, kernel**
  - First two used by user code, last two by OS

Building a VMM for the Vax

- **Problem: VAX not fully "virtualizable"**
  - Some instructions sensitive but not privileged (don't trap)
  - Page tables can be modified without trapping to VMM
  - VMS uses all 4 protection rings, would need 5th for VMM
- **Solve first two by modifying microcode**
  - Extra bit in PSL indicates VM status
  - Fake PSL, VMPSL, contains emulated PSL
  - Causes sensitive instructions to trap
- **Solve third with "ring compression"**
  - Run both kernel and executive code in the executive
  - VMS kernel happens to trust executive anyway

Support for I/O

- **Device drivers in the OSes will no longer work**
  - VMM guards access to hardware
  - Could fix up requests by emulating existing hardware
  - Would be expensive, so require new device drivers
- **VOL/F11F layers in software emulate disk devices**
  - Implemented out of contiguously allocated files in a simplified file system
  - *Exchangeable volumes* use same format as regular OS
  - *Security kernel volumes* can contain mixed-label data

Security architecture

- **Subjects: Users and VMs**
  - Secure attention key lets user communicate with TCB
- **Objects: Devices, memory, disk and tape volumes**
- **Security levels (access classes)**
  - 8-bit security and integrity levels
  - 64-bits each of secrecy/integrity category-set

Invoking the VMM

- **VMs can make two calls into VMM:**
  - OPERATE – mount/unmount volumes, etc.
  - SET_ACL – change ACL on an object
- **Users can perform many more operations**
  - Connect/disconnect from virtual machines
  - Invoke privileges (e.g., change password, downgrade, . . .)
- **Problem: Don't want complicated parser in TCB**
  - But users want features like shell history, etc.
  - Solution: User types security commands to untrusted OS
  - VMM requires user to press secure attention key
  - VMM then confirms arguments actually passed to it

Software engineering

- **Highly-layered design (see p. 9)**
  - Lower layers prohibited from calling up (except event counts)
  - Aggressive sanity checking across abstraction layers
  - All freed memory set to 1s
- **Formal methods, as required by orange book**
- **Extensive design reviews**
- **High-security development environment**
  - System developed on itself
  - Locked cage inside locked room

Was it worth it?

- **System was almost 10 years in the making**
  - For about 50,000 lines of code (see Fig. 3)
- **Never became a product**
- **Does it sound like something you would want to use?**
  - Not even a graphical user interface
  - Maybe in 1981 when the project started

LOMAC

- **MAC not widely accepted outside military**
- **LOMAC's goal is to make MAC more palatable**
  - Stands for Low water Mark Access Control
- **Concentrates on Integrity**
  - More important goal for many settings
  - E.g., don't want viruses tampering with all your files
  - Also don't have to worry as much about covert channels
- **Comes with reasonable default**
  - Idea is to be minimally obtrusive

LOMAC overview

- **Subjects are *jobs* (essentially processes)**
  - Each subject has an integrity number (e.g., 1, 2)
  - Higher numbers mean more integrity
  - Subjects can be reclassified on observing low-integrity data
- **Objects file, pipes, etc.**
  - Objects have fixed integrity level; cannot change
- **Security: Low-integrity subjects cannot write to high integrity objects**
- **New objects have level of the creator**

LOMAC defaults

- **By default two levels, 1 and 2**
- **Level 2 (high-integrity) contains:**
  - All the Linux files intact from software distribution
  - The console and trusted terminals
- **Level 1 (low-integrity) contains**
  - Network devices, untrusted terminals, etc.
- **Idea: Suppose work compromises your web server**
  - Worm comes from network $\rightarrow$ level 1
  - Won't be able to muck with system files

The self-revocation problem

- **Want to integrate with Unix unobtrusively**
- **Problem: Application expectations**
  - Kernel access checks usually done at file open time
  - Legacy applications don't pre-declare they will observe low-integrity data
  - An application can "taint" itself unexpectedly, revoking its own permission to access an object it created
- **Example:** `ps | grep user`
  - Pipe created before ps reads low-integrity data
  - Ps becomes tainted, can no longer write to grep

Solution

- **Don't consider pipes to be real objects**
- **Join multiple processes together in a "job"**
  - Pipe ties processes together in job
  - Any processes tied to job when they read or write to pipe
  - So will lower integrity of both ps and grep
- **Similar idea applies to shared memory and IPC**

Stretch break