ECS 289M Lecture 19

May 15, 2006

Variable Classes

- Up to now, classes fixed
  - Check relationships on assignment, etc.
- Consider variable classes
  - Fenton's Data Mark Machine does this for *PC*
  - On assignment of form $y := f(x_1, \ldots, x_n)$, $\underline{y}$ changed to lub{ $\underline{x}_1, \ldots, \underline{x}_n$ }
  - Need to consider implicit flows, also

Example Program

```
(* Copy value from x to y
 * Initially, x is 0 or 1 *)
proc copy(x: int class { x };
                  var y: int class { y })
var z: int class variable { Low };
begin
  y := 0;
  z := 0;
  if x = 0 then z := 1;
  if z = 0 then y := 1;
end;
```

- $\underline{z}$ changes when z assigned to
- Assume $\underline{y} < \underline{x}$

Analysis of Example

- $x = 0$
  - `z := 0` sets $\underline{z}$ to Low
  - `if x = 0 then z := 1` sets z to 1 and $\underline{z}$ to $\underline{x}$
  - So on exit, $y = 0$
- $x = 1$
  - `z := 0` sets $\underline{z}$ to Low
  - `if z = 0 then y := 1` sets y to 1 and checks that lub{Low, $\underline{z}$} ≤ $\underline{y}$
  - So on exit, $y = 1$
- Information flowed from $\underline{x}$ to $\underline{y}$ even though $\underline{y} < \underline{x}$

Handling This (1)

- Fenton's Data Mark Machine detects implicit flows violating certification rules

Handling This (2)

- Raise class of variables assigned to in conditionals even when branch not taken
- Also, verify information flow requirements even when branch not taken
- Example:
  - In `if` *x* `= 0 then` *z* `:= 1`, $\underline{z}$ raised to $\underline{x}$ whether or not $x = 0$
  - Certification check in next statement, that $\underline{z} \leq \underline{y}$, fails, as $\underline{z} = \underline{x}$ from previous statement, and $\underline{y} \leq \underline{x}$

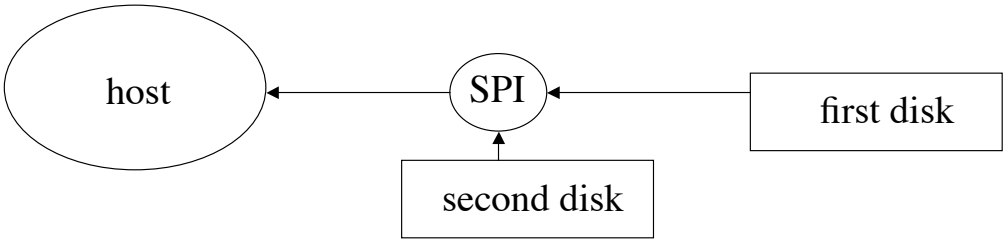
Handling This (3)

- Change classes only when explicit flows occur, but *all* flows (implicit as well as explicit) force certification checks
- Example
  - When $x = 0$, first "if" sets $\underline{z}$ to Low then checks $\underline{x} \leq \underline{z}$
  - When x = 1, first "if" checks that $\underline{x} \leq \underline{z}$
  - This holds if and only if $\underline{x}$ = Low
    - Not possible as $\underline{y} < \underline{x}$ = Low and there is no such class

Example Information Flow Control Systems

- Use access controls of various types to inhibit information flows
- Security Pipeline Interface
  - Analyzes data moving from host to destination
- Secure Network Server Mail Guard
  - Controls flow of data between networks that have different security classifications

Security Pipeline Interface

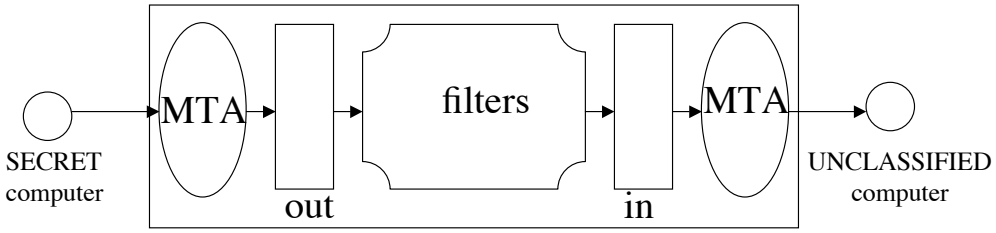


- SPI analyzes data going to, from host
  - No access to host main memory
  - Host has no control over SPI

Use

- Store files on first disk
- Store corresponding crypto checksums on second disk
- Host requests file from first disk
  - SPI retrieves file, computes crypto checksum
  - SPI retrieves file's crypto checksum from second disk
  - If a match, file is fine and forwarded to host
  - If discrepency, file is compromised and host notified
- Integrity information flow restricted here
  - Corrupt file can be seen but will not be trusted

Secure Network Server Mail Guard (SNSMG)



- Filters analyze outgoing messages
  - Check authorization of sender
  - Sanitize message if needed (words and viruses, etc.)
- Uses type checking to enforce this
  - Incoming, outgoing messages of different type
  - Only appropriate type can be moved in or out

Confinement

- The confinement problem
- Isolating entities
  - Virtual machines
  - Sandboxes
- Covert channels
  - Detecting them
  - Analyzing them
  - Mitigating them

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

Lipner's Notes

- All processes can obtain rough idea of time
  - Read system clock or wall clock time
  - Determine number of instructions executed
- All processes can manipulate time
  - Wait some interval of wall clock time
  - Execute a set number of instructions, then block

Kocher's Attack

- This computes $x = a^z \bmod n$, where $z = z_0 \ldots z_{k-1}$

```
x := 1; atmp := a;
for i := 0 to k–1 do begin
  if z_i = 1 then
      x := (x * atmp) mod n;
  atmp := (atmp * atmp) mod n;
end
result := x;
```

- Length of run time related to number of 1 bits in z

Isolation

- Present process with environment that appears to be a computer running only those processes being isolated
  - Process cannot access underlying computer system, any process(es) or resource(s) not part of that environment
  - A *virtual machine*
- Run process in environment that analyzes actions to determine if they leak information
  - Alters the interface between process(es) and computer

Virtual Machine

- Program that simulates hardware of a machine
  - Machine may be an existing, physical one or an abstract one
- Why?
  - Existing OSes do not need to be modified
    - Run under VMM, which enforces security policy
    - Effectively, VMM is a security kernel

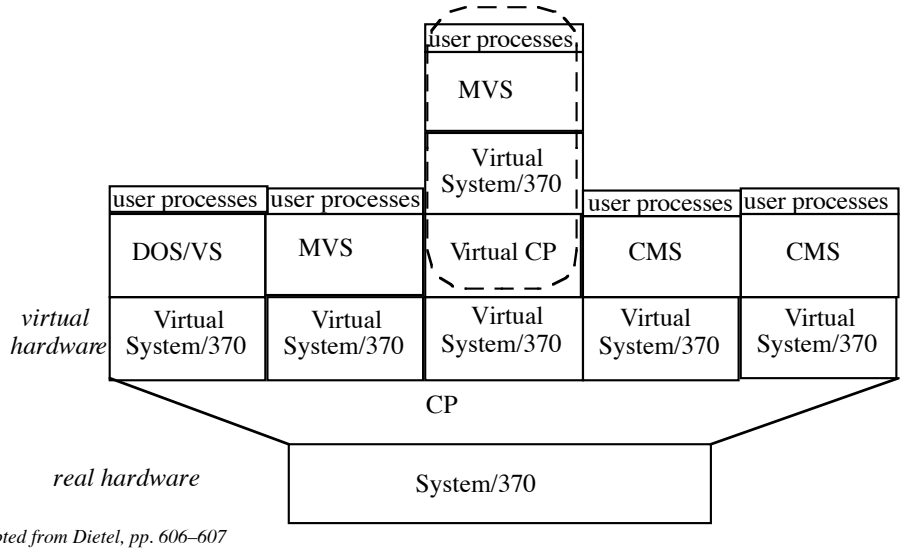
Review of How VMs Work

- Virtual Machine Structure
- Virtual Machine Monitor
  - Privilege
  - Physical Resources
  - Paging

What Is It?

- *Virtual machine monitor* (VMM) virtualizes system resources
  - Runs directly on hardware
  - Provides interface to give each program running on it the illusion that it is the only process on the system and is running directly on hardware
  - Provides illusion of contiguous memory beginning at address 0, a CPU, and secondary storage to *each* program

Example: IBM VM/370



Adapted from Dietel, pp. 606–607

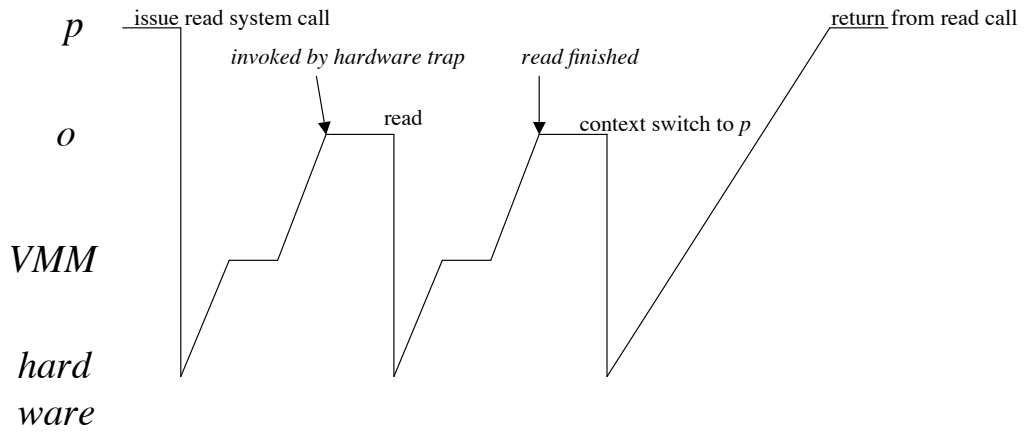
Privileged Instructions

1. VMM running operating system *o*, which is running process *p*
   - *p* tries to read—privileged operation traps to hardware
2. VMM invoked, determines trap occurred in *o*
   - VMM updates state of *o* to make it look like hardware invoked *o* directly, so *o* tries to read, causing trap
3. VMM does read
   - Updates *o* to make it seem like *o* did read
   - Transfers control to *o*

Privileged Instructions

4. *o* tries to switch context to *p*, causing trap
5. VMM updates virtual machine of *o* to make it appear *o* did context switch successfully
   - Transfers control to *o*, which (as *o* apparently did a context switch to *p*) has the effect of returning control to *p*

Privileged Instructions



Privilege and VMs

- *Sensitive instruction* discloses or alters state of processor privilege
- *Sensitive data structure* contains information about state of processor privilege

When Is VM Possible?

- Can virtualize an architecture when:
  1. All sensitive instructions cause traps when executed by processes at lower levels of privilege
  2. All references to sensitive data structures cause traps when executed by processes at lower levels of privilege

Example: VAX System

- 4 levels of privilege (user, supervisor, executive, kernel)
  - CHMK changes privilege to kernel level; sensitive instruction
    - Causes trap *except* when executed in kernel mode; meets rule 1
  - Page tables have copy of PSL, containing privilege level; sensitive data structure
    - If user level processes prevented from altering page tables, trying to do so will cause a trap; this meets rule 2