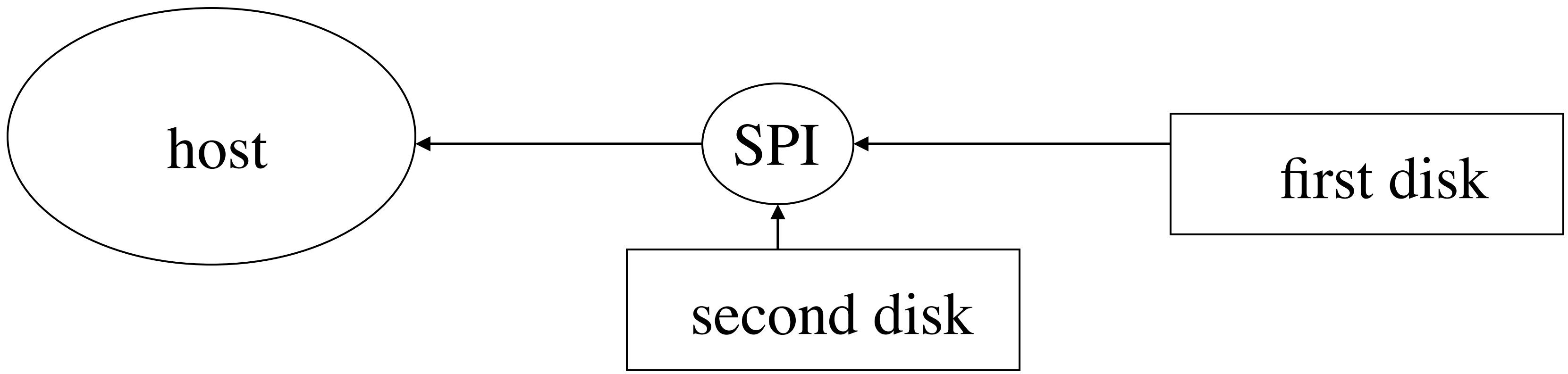
Lecture 14: Flow & Confinement

- Examples of information flow applications
- The confinement problem
- Isolation: virtual machines, sandboxes
- Covert channels
  - Detection
  - Mitigation
- The pump

Examples

- 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

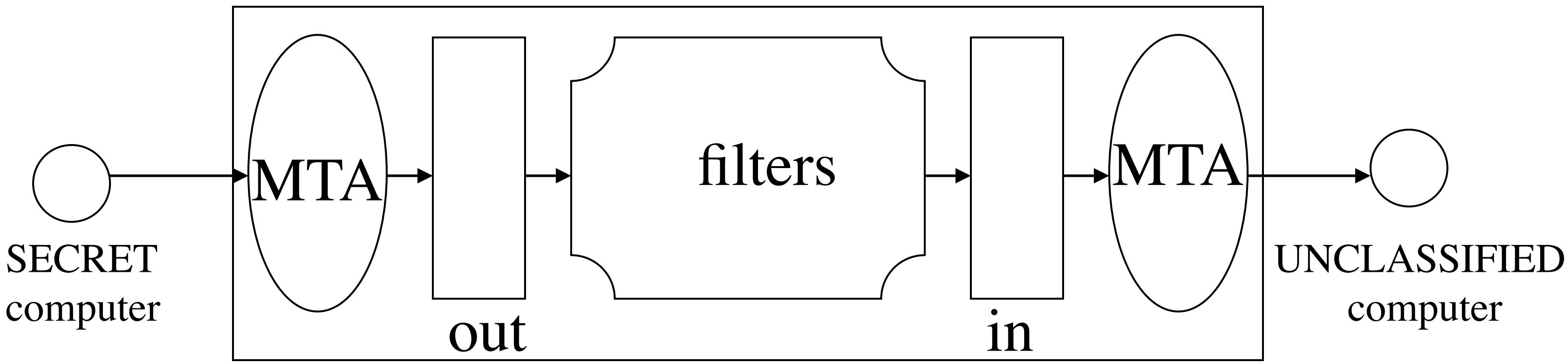


- SPI analyzes data going to, from host
  - No access to host main memory
  - Host has no control over SPI

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 discrepancy, 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

- What is the problem?
- Isolation: virtual machines, sandboxes
- Detecting covert channels

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

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 1: KVM/370

- KVM/370 is security-enhanced version of VM/370 VMM
  - Goal: prevent communications between VMs of different security classes
  - Like VM/370, provides VMs with minidisks, sharing some portions of those disks
  - Unlike VM/370, mediates access to shared areas to limit communication in accordance with security policy

Example 2: VAX/VMM

- Can run either VMS or Ultrix
- 4 privilege levels for VM system
  - VM user, VM supervisor, VM executive, VM kernel (both physical executive)
- VMM runs in physical kernel mode
  - Only it can access certain resources
- VMM subjects: users and VMs

Example 2

- VMM has flat file system for itself
  - Rest of disk partitioned among VMs
  - VMs can use any file system structure
    - Each VM has its own set of file systems
  - Subjects, objects have security, integrity classes
    - Called *access classes*
  - VMM has sophisticated auditing mechanism

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

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

Modifying Programs

- Add breakpoints or special instructions to source, binary code
  - On trap or execution of special instructions, analyze state of process
- Variant: *software fault isolation*
  - Add instructions checking memory accesses, other security issues
  - Any attempt to violate policy causes trap

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

Configuration File

```
# 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

Sandboxes, VMs, and TCB

- Sandboxes, VMs part of trusted computing bases
  - Failure: less protection than security officers, users believe
  - "False sense of security"
- Must ensure confinement mechanism correctly implements desired security policy

Covert Channels

- Shared resources as communication paths
- *Covert storage channel* uses attribute of shared resource
  - Disk space, message size, etc.
- *Covert timing channel* uses temporal or ordering relationship among accesses to shared resource
  - Regulating CPU usage, order of reads on disk

Example Storage Channel

- 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

Example Timing Channel

- System has two VMs
  - Sending machine *S*, receiving machine *R*
- To send:
  - For 0, *S* immediately relinquishes CPU
    - For example, run a process that instantly blocks
  - For 1, *S* uses full quantum
    - For example, run a CPU-intensive process
- *R* measures how quickly it gets CPU
  - Uses real-time clock to measure intervals between access to shared resource (CPU)

Example Covert Channel

- Uses ordering of events; does not use clock
- Two VMs sharing disk cylinders 100 to 200
  - SCAN algorithm schedules disk accesses
  - One VM is *High* (*H*), other is *Low* (*L*)
- Idea: *L* will issue requests for blocks on cylinders 139 and 161 to be read
  - If read as 139, then 161, it's a 1 bit
  - If read as 161, then 139, it's a 0 bit

How It Works

- *L* issues read for data on cylinder 150
  - Relinquishes CPU when done; arm now at 150
- *H* runs, issues read for data on cylinder 140
  - Relinquishes CPU when done; arm now at 140
- *L* runs, issues read for data on cylinders 139 and 161
  - Due to SCAN, reads 139 first, then 161
  - This corresponds to a 1
- To send a 0, *H* would have issued read for data on cylinder 160

Analysis

- Timing or storage?
  - Usual definition ⇒ storage (no timer, clock)
- Modify example to include timer
  - L uses this to determine how long requests take to complete
  - Time to seek to 139 < time to seek to 161 ⇒ 1; otherwise, 0
- Channel works same way
  - Suggests it's a timing channel; hence our definition

Noisy vs. Noiseless

- Noiseless: covert channel uses resource available only to sender, receiver
- Noisy: covert channel uses resource available to others as well as to sender, receiver
  - Idea is that others can contribute extraneous information that receiver must filter out to "read" sender's communication

Key Properties

- *Existence*: the covert channel can be used to send/receive information
- *Bandwidth*: the rate at which information can be sent along the channel
- Goal of analysis: establish these properties for each channel
  - If you can eliminate the channel, great!
  - If not, reduce bandwidth as much as possible

Step #1: Detection

- Manner in which resource is shared controls who can send, receive using that resource
  - Noninterference
  - Shared Resource Matrix Methodology
  - Information flow analysis
  - Covert flow trees

Noninterference

- View "read", "write" as instances of information transfer
- Then two processes can communicate if information can be transferred between them, even in the absence of a direct communication path
  - A covert channel
  - Also sounds like interference …

Example: SAT

- Secure Ada Target, multilevel security policy
- Approach:
  - $\pi(i, l)$ removes all instructions issued by subjects dominated by level l from instruction stream i
  - $A(i, \sigma)$ state resulting from execution of i on state σ
  - $\sigma.v(s)$ describes subject s's view of state σ
- System is noninterference-secure iff for all instruction sequences i, subjects s with security level $l(s)$, states σ,

$$A(\pi(i, l(s)), \sigma).v(s) = A(i, \sigma).v(s)$$

Theorem

- Version of the Unwinding Theorem
- Let Σ be set of system states. A specification is noninterference-secure if, for each subject *s* at security level $l(s)$, there exists an equivalence relation $\equiv: \Sigma \times \Sigma$ such that
  - for $\sigma_1, \sigma_2 \in \Sigma$, when $\sigma_1 \equiv \sigma_2$, $\sigma_1.v(s) = \sigma_2.v(s)$
  - for $\sigma_1, \sigma_2 \in \Sigma$ and any instruction *i*, when $\sigma_1 \equiv \sigma_2$, $A(i, \sigma_1) \equiv A(i, \sigma_2)$
  - for $\sigma \in \Sigma$ and instruction stream *i*, if $\pi(i, l(s))$ is empty, $A(\pi(i, l(s)), \sigma).v(s) = \sigma.v(s)$

Intuition

- System is noninterference-secure if:
  - Equivalent states have the same view for each subject
  - View remains unchanged if any instruction is executed
  - Instructions from higher-level subjects do not affect the state from the viewpoint of the lower-level subjects

Analysis of SAT

- Focus on object creation instruction and readable object set
- In these specifications:
  - s subject with security level $l(s)$
  - o object with security level $l(o)$, type $\tau(o)$
  - σ current state
  - Set of existing objects listed in a global object table $T(\sigma)$

Specification 1

- *object_create*:

$$[\ \sigma' = object\_create(s,o,l(o),\tau(o),\sigma) \wedge \sigma' \neq \sigma\ ]$$
$$\Leftrightarrow$$
$$[\ o \notin T(\sigma) \wedge \mathrm{l}(s) \leq l(o)\ ]$$

- The create succeeds if, and only if, the object does not yet exist and the clearance of the object will dominate the clearance of its creator
  - In accord with the "writes up okay" idea

Specification 2

- readable object set: set of existing objects that subject could read
  - *can_read*(s, o, σ) true if in state σ, o is of a type that s can read (ignoring permissions)
- $o \notin readable(s, \sigma) \Leftrightarrow [\ o \notin T(\sigma) \lor$

$$\neg(l(o) \leq l(s)) \lor \neg(can\_read(s, o, \sigma))]$$

- Can't read a nonexistent object, one with a security level that the subject's security level does not dominate, or object of the wrong type

Specification 3

- SAT enforces tranquility
  - Adding object to readable set means creating new object
- Add to readable set:

  $[o \notin readable(s, \sigma) \wedge o \in readable(s, \sigma')] \Leftrightarrow [\sigma' = object\_create(s, o, l(o), \tau(o), \sigma) \wedge o \notin T(\sigma) \wedge l(s') \le l(o) \le l(s) \wedge can\_read(s, o, \sigma')]$
- Says object must be created, levels and discretionary access controls set properly

Check for Covert Channels

- σ_1, σ_2 the same except:
  - o exists only in latter
  - $\neg(l(o) \le l(s))$
- Specification 2:
  - $o \notin readable(s, \sigma_1)$ { o doesn't exist in σ_1}
  - $o \notin readable(s, \sigma_2)$ { $\neg(l(o) \le l(s))$ }
- Thus $\sigma_1 \equiv \sigma_2$
  - Condition 1 of theorem holds

Continue Analysis

- s' issues command to create o with:
  - $l(o) = l(s)$
  - of type with $can\_read(s, o, \sigma_1')$
    - σ_1' state after $object\_create(s', o, l(o), \tau(o), \sigma_1)$
- Specification 1
  - σ_1' differs from σ_1 with o in $T(\sigma_1)$
- New entry satisfies:
  - $can\_read(s, o, \sigma_1')$
  - $l(s') \le l(o) \le l(s)$, where s' created o

Continue Analysis

- *o* exists in σ_2 so:

$$\sigma_2' = object\_create(s', o, \sigma_2) = \sigma_2$$

- But this means

$$\neg [\ A(object\_create(s', o, l(o), \tau(o), \sigma_2), \sigma_2) \equiv A(object\_create(s', o, l(o), \tau(o), \sigma_1), \sigma_1)\ ]$$

  - Because create fails in σ_2 but succeeds in σ_1
- So condition 2 of theorem fails
- This implies a covert channel as system is not noninterference-secure

Example Exploit

- To send 1:
  - High subject creates high object
  - Recipient tries to create same object but at low
    - Creation fails, but no indication given
  - Recipient gives different subject type permission to read, write object
    - Again fails, but no indication given
  - Subject writes 1 to object, reads it
    - Read returns nothing

Example Exploit

- To send 0:
  - High subject creates nothing
  - Recipient tries to create same object but at low
    - Creation succeeds as object does not exist
  - Recipient gives different subject type permission to read, write object
    - Again succeeds
  - Subject writes 1 to object, reads it
    - Read returns 1

Use

- Can analyze covert storage channels
  - Noninterference techniques reason in terms of security levels (attributes of objects)
- Covert timing channels much harder
  - You would have to make ordering an attribute of the objects in some way

SRMM

- Shared Resource Matrix Methodology
- Goal: identify shared channels, how they are shared
- Steps:
  - Identify all shared resources, their visible attributes [rows]
  - Determine operations that reference (read), modify (write) resource [columns]
  - Contents of matrix show how operation accesses the resource

Example

- Multilevel security model
- File attributes:
  - existence, owner, label, size
- File manipulation operations:
  - read, write, delete, create
  - create succeeds if file does not exist; gets creator as owner, creator's label
  - others require file exists, appropriate labels
- Subjects:
  - High, Low

Shared Resource Matrix

	read	**write**	**delete**	**create**
existence	R	R	R, M	R, M
owner			R	M
label	R	R	R	M
size	R	M	M	M

Covert Storage Channel

- Properties that must hold for covert storage channel:
  1. Sending, receiving processes have access to same *attribute* of shared object;
  2. Sender can modify that attribute;
  3. Receiver can reference that attribute; and
  4. Mechanism for starting processes, properly sequencing their accesses to resource

Example

- Consider attributes with both R, M in rows
- Let High be sender, Low receiver
- create operation both references, modifies existence attribute
  - Low can use this due to semantics of create
- Need to arrange for proper sequencing accesses to existence attribute of file (shared resource)

Use of Channel

- 3 files: *ready*, *done*, *1bit*
- Low creates *ready* at High level
- High checks that file exists
  - If so, to send 1, it creates *1bit*; to send 0, skip
  - Delete *ready*, create *done* at High level
- Low tries to create *done* at High level
  - On failure, High is done
  - Low tries to create *1bit* at level High
- Low deletes *done*, creates *ready* at High level

Covert Timing Channel

- Properties that must hold for covert timing channel:
  1. Sending, receiving processes have access to same *attribute* of shared object;
  2. Sender, receiver have access to a time reference (wall clock, timer, event ordering, …);
  3. Sender can control timing of detection of change to that attribute by receiver; and
  4. Mechanism for starting processes, properly sequencing their accesses to resource

Example

- Revisit variant of KVM/370 channel
  - Sender, receiver can access ordering of requests by disk arm scheduler (attribute)
  - Sender, receiver have access to the ordering of the requests (time reference)
  - High can control ordering of requests of Low process by issuing cylinder numbers to position arm appropriately (timing of detection of change)
  - So whether channel can be exploited depends on whether there is a mechanism to (1) start sender, receiver and (2) sequence requests as desired

Uses of SRM Methodology

- Applicable at many stages of software life cycle model
  - Flexbility is its strength
- Used to analyze Secure Ada Target
  - Participants manually constructed SRM from flow analysis of SAT model
  - Took transitive closure
  - Found 2 covert channels
    - One used assigned level attribute, another assigned type attribute

Summary

- Methodology comprehensive but incomplete
  - How to identify shared resources?
  - What operations access them and how?
- Incompleteness a benefit
  - Allows use at different stages of software engineering life cycle
- Incompleteness a problem
  - Makes use of methodology sensitive to particular stage of software development