

Chapter 16 CONFINEMENT PROBLEM

- When a program executes, it interacts with its environment.
- The security policy allows some interactions and disallows others.
- The confinement problem deals with prevention of processes from taking disallowed actions.
- Systems isolate processes in two ways
 - In the first, the process is presented with an environment that appears to be a computer running only that process or those processes to be isolated.
 - The first type of environment prevents the process from accessing the underlying computer systems and any processes or resources that are not part of that environment.
 - In the second, an environment is provided in which the process actions are analyzed if they leak information.
 - The second type of environment does not emulate a computer. It just alters the interface between the existing computer and the process(es).
- Covert channels
 - Use shared resources as paths of communication, requiring sharing of space or sharing of time.
 - Detection of covert channels
 - Mitigation of covert channels

Chapter 16 CONFINEMENT PROBLEM

16.1 The Confinement Problem

- Consider a client and a server.
 - When the client issues a request to the server, the client sends the server some data.
 - The server uses the data to perform some function and either returns a result or no result.
 - Access control affects the function of the server in two ways:
 - The server must ensure that the resources it accesses on behalf of the client include only those resources the client is authorized to access.
 - This requirement represents the goal of the service provider.
 - The goal is to prevent the client from sending messages that cause the server to perform any function that the client is not authorized to perform.
 - The server must ensure that it does not reveal the client's data to any other entity not authorized to see the client's data.
 - This requirement represents the goal of the service user.
 - The goal is to prevent the server from transmitting confidential information to the service provider.
 - In either case, the server must be confined to accessing only a specific set of resources.

Chapter 16 CONFINEMENT PROBLEM

- **EXAMPLE:**
 - A server balances bank accounts for subscribers.
 - Each subscriber uses a client to transmit data about the account to the server (e.g., the current balance).
 - The server returns a list of information (e.g., outstanding checks).
 - Subscribers pay a fee for each use.
 - Server security issues
 - The service provider requires that the server correctly record who used the service.
 - The service provider also wants the server not to transmit any unauthorized information to the client (e.g., billing records).
 - So, the server must be confined to operating only on the data it is sent.
 - Client security issues
 - The server must correctly log the user's invocation so that the user is not charged incorrectly.
 - The server must not record or transmit data to the service provider because the subscriber's data is confidential to the subscriber.
 - So, the server must be confined to keeping the data to itself and to sending the results only to the subscriber.
 - Lampson calls this the confinement problem.

Chapter 16 CONFINEMENT PROBLEM

- **DEFINITION:** The confinement problem is the problem of preventing a server from leaking information that the user of the service considers confidential.
 - If a process does not store information, it cannot leak it.
 - Total isolation (idea by Lampson): A process that cannot be observed and cannot communicate with other processes cannot leak information.
 - In practice, achieving total isolation is difficult.
 - However, the processes to be confined usually share resources (e.g., CPUs, networks, and disk storage with other, unconfined processes).
 - Therefore, the unconfined processes can transmit information over the shared resources.
- **DEFINITION:** A covert channel is a path of communication that was not designed to be used for communication.
- **EXAMPLE:** Consider two processes p and q .
 - Process p is confined and it cannot communicate with process q .
 - Processes p and q 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 or 1 , and $send$ and waits until $send$ is recreated before repeating to read another bit
 - The protocol continues until p creates a file called end . At this point, the communication ceases.

Chapter 16 CONFINEMENT PROBLEM

- **DEFINITION:** The rule of transitive confinement states that if a confined process invokes a second process, the second process must be as confined as the caller.
 - Assume that process p is confined to prevent leakage.
 - If it invokes a second process q , then q must be similarly confined.
 - Otherwise, q could leak the information that p passes.
- **Confinement is a mechanism for enforcing the principle of least privilege.**
 - The principle of least privilege states that a subject should be given only those privileges that it needs in order to complete its tasks.
 - There are several problems that need to be addressed:
 - A properly confined process cannot transmit data to a second process unless the transmission is needed to complete their task.
 - The confined process needs access to the data to be transmitted.
 - So, the confinement must be on the transmission, not on the data access.
 - To complicate matters, the process may have to transmit some information to the second process.
 - In this case, the confinement mechanism must distinguish between transmission of authorized data and transmission of unauthorized data.

Chapter 16 CONFINEMENT PROBLEM

16.2 Isolation

- **Virtual machines**
 - Emulate computer
 - Process cannot access underlying computer system, anything not part of that computer system
- **Sandboxing**
 - Does not emulate computer
 - Alters interface between computer and process
- **Virtual machines**
 - A virtual machine (VM) is a program that simulates the HW of a (possibly abstract) computer system.
 - A virtual machine uses a special OS called a virtual machine monitor to provide a virtual machine on which conventional OS's can run.
 - The primary advantage of a VM is that existing OS's do not need to be modified.
 - Each VM is one subject; VMM knows nothing about processes running on each VM.
 - VMM mediates all interactions of VM with resources and other VM's.
 - VMM can apply security checks to its subjects, and those controls apply to the processes that those subjects are running.
 - Therefore, rule of transitive confinement is satisfied.

Chapter 16 CONFINEMENT PROBLEM

- **EXAMPLE: KVM/370**
 - Security-enhanced version of IBM VM/370 VMM
 - Provided virtual machines for its users.
 - One of its goals was to prevent communication between VM's of different security classes.
 - Like VM/370, it provided VM's with minidisks; some VM's could share some areas of disk.
 - Unlike VM/370, it used a security policy to mediate access to shared areas of the disk to limit communications between systems.
- **EXAMPLE: DEC VAX VMM**
 - A VMM developed by DEC for the DEC VAX.
 - The monitor is a security kernel that can run Ultrix OS or VMS OS.
 - The VMM runs on the native VAX HW and is invoked whenever the virtual machine executes a privileged instruction.
 - The VAX has 4 levels of privilege: user, supervisor, executive, and kernel.
 - The VMM subjects are users and VM's.
 - The VMM has a basic, flat file system for its own use and partitions the remaining disk space among the virtual machines.
 - Each VM has its own set of file systems.
 - Each subject and object has a multilevel security and integrity label.
 - The security and integrity levels form an access class.
 - Two entities have the same access class iff their security and integrity levels are the same, and one entity dominates another iff both the security and integrity classes dominate.

Chapter 16 CONFINEMENT PROBLEM

- **Sandboxes**
 - A playground sandbox provides a safe environment for children to stay in.
 - If the children leave the sandbox without supervision, they may do things they are not supposed to do.
 - The computer sandbox is similar!
 - It provides a safe environment for programs to execute in.
 - If the programs leave the sandbox, they may do things they are not supposed to do.
 - Hence, both types of sandboxes restrict the actions of their occupants.
- **DEFINITION:** A sandbox is an environment in which the actions of a process are restricted according to a security policy.
- **Systems may enforce restrictions in two ways:**
 - In the first enforcement method, the sandbox can limit the execution environment as needed.
 - This is usually done by adding extra security checking mechanisms to the libraries or kernel. The program itself is not modified.
 - The JAVA virtual machine is a sandbox because its security manager limits access of downloaded programs to system resources as dictated by a security policy.
 - In the second enforcement method, the program (or process) is modified to be executed.
 - Dynamic debuggers and some profilers use this technique by adding breakpoints to the code. When the trap occurs, the state of the running process is analyzed.

Chapter 16 CONFINEMENT PROBLEM

16.3 Covert Channels

- **DEFINITION:**

- A covert storage channel uses an attribute of the shared resource.
- A covert timing channel uses a temporal or ordering relationship among accesses to a shared resource.
- **EXAMPLE: File Manipulation**
 - The example about two processes p and q is a covert storage channel.
 - The shared resource is the directory and the names of the files in that directory.
 - The processes communicate by altering characteristics (file names and file existence) of the shared resource.
- **EXAMPLE: Real-Time Clock**
 - KVM/370 had covert timing channel
 - Two VM's can establish a covert channel based on the CPU quantum that each VM receives.
 - VM1 wants to send 1 bit to VM2.
 - » To send 0 bit: VM1 relinquishes CPU as soon as it gets CPU
 - » To send 1 bit: VM1 uses CPU for full quantum
 - » VM2 determines which bit is sent by seeing how quickly it gets CPU
 - » The shared resource is the CPU
 - » The processes communicate by using a real-time clock to measure the intervals between accesses to a shared resource.
 - Hence, this is a covert timing channel.

Chapter 16 CONFINEMENT PROBLEM

- A covert timing channel is usually defined in terms of a real-time clock or a timer.
- However, temporal relationship sometimes use neither.
- An ordering of events implies a time-based relationship that involves neither a real-time clock not a timer.
- **EXAMPLE: Ordering of Events**
 - Two VM's
 - Share cylinders 100–200 on a disk
 - The disk uses a SCAN algorithm to schedule disk accesses
 - One VM has security class High, and the other has class Low
 - A process on High VM wants to send information to a process on Low VM
 - The process on the Low machine issues a read request for data on cylinder 150. When the request completes, it relinquishes the CPU.
 - Now we know where the disk head is.
 - The process on the High machine wants to send a bit
 - To send 1 bit, High seeks to cylinder 140 and relinquish CPU
 - To send 0 bit, High seeks to cylinder 160 and relinquish CPU
 - The process on the Low machine issues requests for tracks 139 and 161
 - Seek to 139 first indicates a 1 bit
 - Seek to 161 first indicates a 0 bit
 - The process on the Low machine uses the timer to determine how long it takes for its requests to complete.
 - This is an example of covert timing channel because it uses ordering relationship among accesses to transmit information.

Chapter 16 CONFINEMENT PROBLEM

- **DEFINITION:**
 - A noiseless covert channel is a covert channel that uses a resource available to the sender and receiver only.
 - A noisy covert channel is a covert channel that uses a resource available also to subjects other than the sender and receiver.
- The difference between these two types of channels lies in the need to filter out extraneous information.
 - Any information that the receiver obtains from a noiseless channel comes from the sender.
 - In a noisy channel, the sender's information is mixed with meaningless information (or noise) from other entities using the resource.
- The key properties of covert channels are existence and bandwidth.
 - Existence informs us that there is a channel along which information can be transmitted.
 - Bandwidth informs us how rapidly information can be sent.
 - Covert channel analysis establishes both properties.
 - The channels can then be eliminated or their bandwidth can be reduced.

Chapter 16 CONFINEMENT PROBLEM

- **Detection of covert channels**
 - Covert channels require sharing.
 - The manner of sharing controls which subjects can send, which subjects can receive information using that shared resource.
 - Porras and Kemmerer have developed an approach:
 - They model the flow of information through shared resources with a tree.
 - That paths of flow are identified in this structure.
 - The analyst is able to determine whether is flow is legitimate or covert.
- **A covert tree is a tree-structured representation of the sequence of operations that move information from one process to another.**
- **It is consisted of 5 types of nodes:**
 - Goal symbols
 - An operation symbol
 - A failure symbol
 - An and symbol
 - An or symbol
- **Constructing the tree is a 3 step process.**

Chapter 16 CONFINEMENT PROBLEM

- **Goal Symbol Tree Nodes**
 - A **modification** goal is reached when an attribute is modified.
 - A **recognition** goal is reached when a modification of an attribute is detected.
 - A **direct recognition** goal is reached when a subject can detect the modification of an attribute by referencing it directly or calling a function that returns it.
 - An **inferred recognition** goal is reached when a subject can detect the modification of an attribute without referencing it directly and without calling a function that references the attribute directly.
 - An **inferred-via** goal is reached when information is passed from one attribute to other attributes via a specified primitive operation (e.g., system call).
 - A **recognized-new-state** goal is reached when an attribute that was modified when information was passed using it is specified by an inferred-via goal.

Chapter 16 CONFINEMENT PROBLEM

- **Other Tree Nodes**
 - An operation symbol represents a primitive operation.
 - These symbols may vary among systems if they have different primitive operations.
 - A failure symbol indicates that information cannot be sent along the path on which it lies.
 - An and symbol is a goal that is reached when both of the following hold for all children:
 - The child is an operation
 - If the child is a goal, then the goal is reached.
 - An or symbol reached when either of the following holds for any children:
 - The child is an operation.
 - If the child is a goal, then the goal is reached.
- To make the steps concrete, a simple set of operations are presented.
- These operations will help us understand if they can create covert channel.

Chapter 16 CONFINEMENT PROBLEM

- **EXAMPLE: Consider a file system.**
 - Each file has 3 attributes:
 - The boolean attribute locked is true when the file is locked
 - The boolean attribute, isopen, is true when the file is opened
 - The third attribute inuse is a set that contains the process ID of each process that has the file open
 - Functions:
 - The function *read_access(p,f)* is true if process *p* has read rights over file *f*
 - The function *empty(s)* is true if set *s* has no members
 - The function *random* returns one of its arguments chosen at random

Chapter 16 CONFINEMENT PROBLEM

Four operations

(* lock the file if it is not locked and not opened; otherwise indicate it is locked by returning false *)

```
procedure Lockfile(f: file): boolean;  
begin  
    if not f.locked and empty(f.inuse) then  
        f.locked := true;
```

```
end;
```

(* unlock the file *)

```
procedure Unlockfile(f: file);  
begin
```

```
    if f.locked then  
        f.locked := false;
```

```
end;
```

(* say whether the file is locked *)

```
function Filelocked(f: file): boolean;
```

```
begin
```

```
    Filelocked := f.locked;
```

```
end;
```

(* open the file if it isn't locked and the process has the right to read the file *)

```
procedure Openfile(f: file);  
begin
```

```
    if not f.locked and  
        read_access(process_id, f) then
```

```
        (* add process ID to inuse set *)
```

```
        f.inuse = f.inuse + process_id;
```

```
end;
```

(* if the process can read the file, say if the file is open, otherwise return a value at random *)

```
function Fileopened(f: file): boolean;
```

```
begin
```

```
    if not read_access(process_id, f) then  
        Fileopened := random(true, false);
```

```
    else
```

```
        Fileopened := not isempty(f.inuse);
```

```
end
```

Chapter 16 CONFINEMENT PROBLEM

The first step in constructing a covert tree: Determine what attributes (if any) the primitive operations reference, modify and return.

Attributes and Operations

	Lockfile	Unlockfile	Filelocked	Openfile	Fileopened
reference	<i>locked, inuse</i>	<i>locked</i>	<i>locked</i>	<i>locked, inuse</i>	<i>inuse</i>
modify	<i>locked</i>	∅	∅	<i>inuse</i>	∅
return	∅	∅	<i>locked</i>	∅	<i>inuse</i>

∅ means no attribute affected in specified manner.

Chapter 16 CONFINEMENT PROBLEM

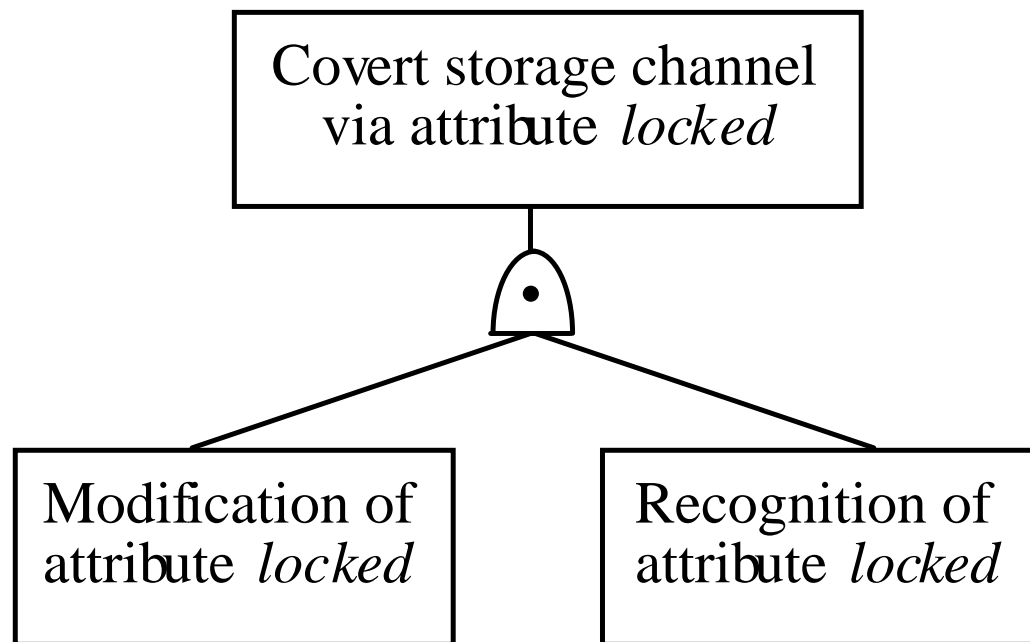
- The second step in constructing a covert tree begins with the goal of locating a covert storage channel that uses some attribute.
- The type of goal controls the construction:
 - The topmost goal requires that the attribute be modified and that the modification be recognized.
 - It has one child (an and symbol), which in turn has 2 children (a modification goal symbol and a recognition goal symbol).
 - A modification goal requires some primitive operation to modify the attribute.
 - It has one or child, which has one child operation symbol per operation for all operations that modify the attribute.
 - A recognition goal requires that a subject either directly recognize or infer an change in an attribute.
 - It has an or symbol as its child.
 - The or symbol has 2 children, one a direct recognition goal symbol and the other an inferred recognition goal symbol.
 - A direct recognition goal requires that an operation access the attribute.
 - Like the modification goal, it has one or child, and the child in turn has one child operation symbol for each operation that returns the attribute.
 - If no operation returns the attribute, a failure symbol is attached
 - An inferred recognition goal requires that the modification be inferred on the basis of one or more other attributes.
 - It has one child (an or symbol), which has one child inferred-via symbol for each operation that references an attribute and that modifies some attribute (possibly the same one that was referenced).
 - An inferred-via goal requires that the value of the attribute be inferred via some operation and a recognition of the new state of the attribute resulting from that operation.
 - It has one child ((an and symbol), which has 2 children (an operation symbol representing the primitive operation used to draw the inference and a recognize-new-state goal symbol).
 - A recognized-new-state goal requires that the value of the attribute be inferred via some operation and a recognition of the new state of the attribute resulting from that operation.
 - The latter requires a recognition goal for the attribute.
 - The child node of the recognized-new-state goal symbol is an or symbol, and for each attribute enabling the inference of the modification of the attribute in question, the or symbol has a recognition goal symbol child.

Chapter 16 CONFINEMENT PROBLEM

- Tree construction ends when all paths through the tree terminate in either an operation symbol or a failure symbol.
- Because the construction is recursive, the analyst may encounter a loop in the tree construction.
- If this happens, a parameter called repeat defines the # of times that the path may be traversed.
 - This places an upper bound on the size of the tree.
- **EXAMPLE:**
 - A covert flow tree for the attribute locked.
 - The goal state is “covert storage channel via attribute locked.”
 - The and node has two children.
 - “modification of attribute locked”
 - “recognition of attribute locked”

Chapter 16 CONFINEMENT PROBLEM

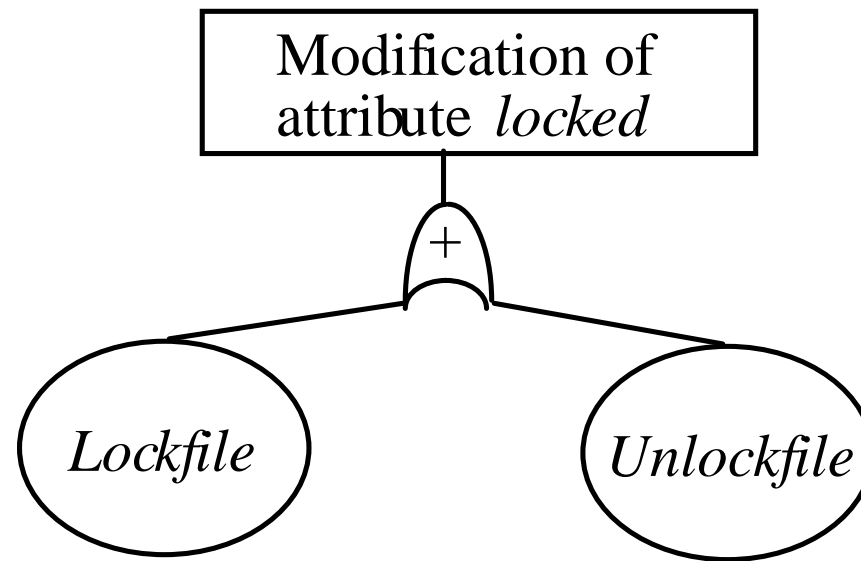
First Step



- Put “and” node under goal
- Put children under “and” node

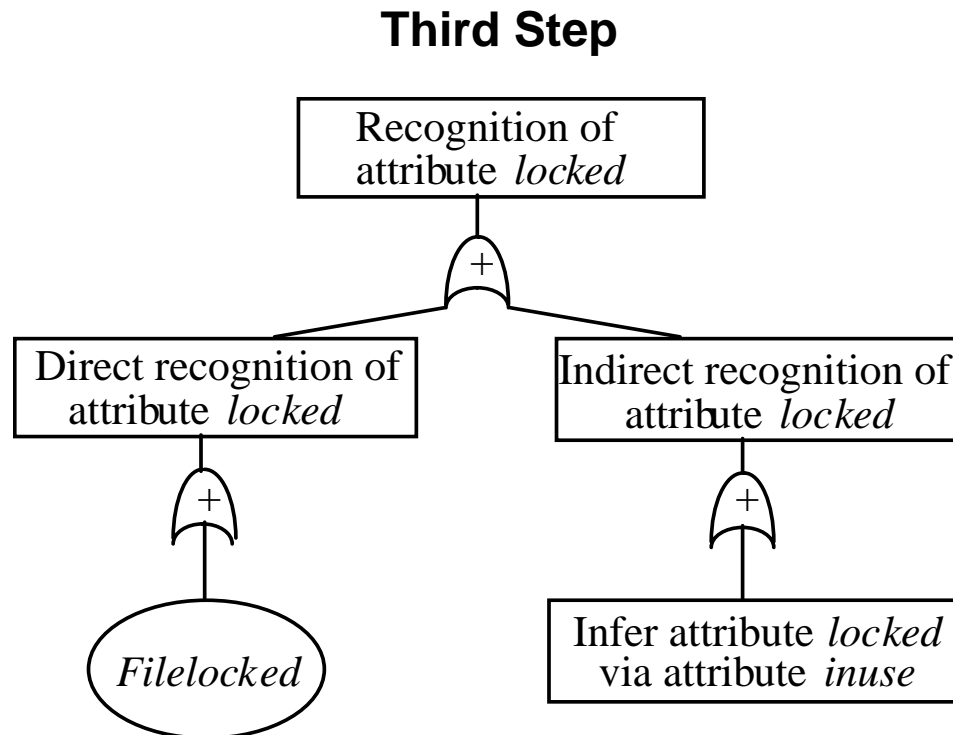
Chapter 16 CONFINEMENT PROBLEM

Second Step



Operations *Lockfile* and *Unlockfile* modify *locked*

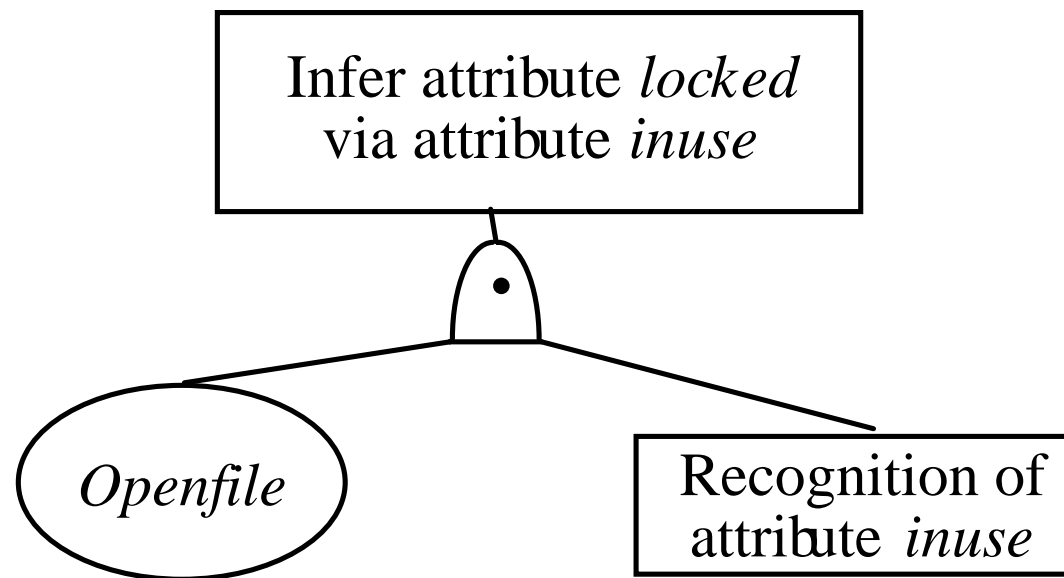
Chapter 16 CONFINEMENT PROBLEM



- “Recognition” had direct, inferred recognition children
- Direct recognition child: “and” node with *Filelocked* child
 - *Filelocked* returns value of *locked*
- Inferred recognition child: “or” node with “inferred-via” node
 - Infers *locked* from *inuse*

Chapter 16 CONFINEMENT PROBLEM

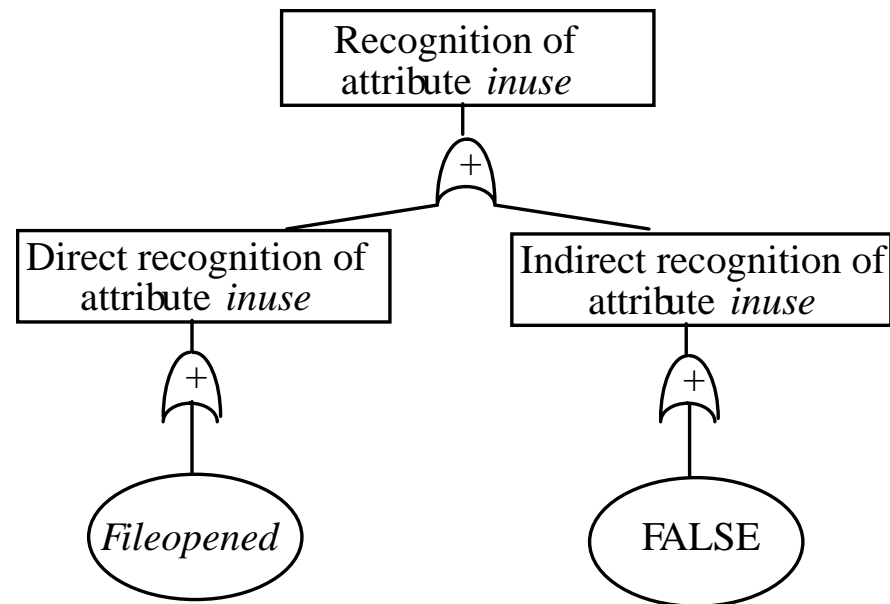
Fourth Step



- “Inferred-via” node requires *Openfile*
 - Change in attribute *inuse* represented by recognize-new-state goal

Chapter 16 CONFINEMENT PROBLEM

Fifth Step



- “Recognize-new-state” node
 - Direct recognition node: “or” child, *Fileopened* node beneath (recognizes change in *inuse* directly)
 - Inferred recognition node: “or” child, FALSE node beneath (nothing recognizes change in *inuse* indirectly)

Chapter 16 CONFINEMENT PROBLEM

Final Tree

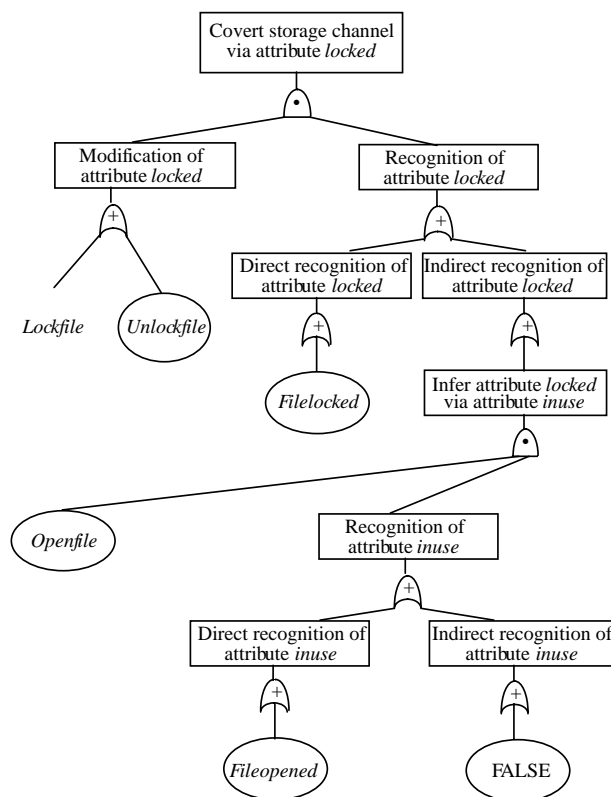
The analyst constructs 2 lists:

The first list contains sequences of operations that modify the attribute.

The second list sequences of operations that recognize modifications in the attribute.

A sequence from the first list followed by a sequence from the second list is a channel along which information can flow.

The analyst examines these channels to determine which are covert.



In the covert tree:

The first list has 2 sequences of operations that modify the attribute:

(*Lockfile*), (*Unlockfile*)

The second list also has 2 sequences of operations that recognize modifications to the attribute:

(*Filelocked*), (*Openfile, Fileopened*))

These sequences result in 4 channels of communication:

Lockfile, then *Filelocked*

Unlockfile, then *Filelocked*

Lockfile, then *Openfile*, then *Fileopened*

Unlockfile, then *Openfile*, then *Fileopened*

Chapter 16 CONFINEMENT PROBLEM

- If a High-level user transmits information to a Low-level user by locking and unlocking a file...
 - the first 2 channels (in combination) represent a direct covert storage channel.
 - The last 2 channels represent an indirect covert storage channel.
- To use the channel, the High-level process locks a file to send a 0 bit and unlocks a file to send a 1 bit.
- The Low-level process tries to open the locked file. It then uses *Fileopened* to see if it has opened the file.
- If the file is opened, the High-level process did not lock the file (a 0 bit).
- If the file is not opened, the High-level process did lock the file (a 1 bit).

Chapter 16 CONFINEMENT PROBLEM

- **Mitigation of covert channels**
 - Covert channels carry information by varying the use of shared resources.
 - One approach to eliminate all covert channels is to require processes to state what resources they need before execution and provide these resources in such a manner that only the process can access them.
 - This includes runtime, and when the stated runtime is reached, the process is terminated and the resources are released.
 - The resources remain allocated for the full even if the process terminates earlier.
 - Otherwise, a second process could infer information from the timing of the release of the resources.
 - This strategy effectively implements Lampson's idea of total isolation, but it is infeasible in practice.
 - An alternative approach is to obscure the amount of resources that a process uses.
 - A receiving process cannot determine what amount of resource usage is attributable to the sender and what amount is attributable to the obfuscation.
 - This can be done in 2 ways:
 - The resources devoted to each process can be made uniform.
 - » The system eliminates meaningful irregularities in resource allocation and use.
 - A system can inject randomness into the allocation and use of resources.
 - » The goal is to make the covert channel a noisy one and to have the noise dominate the channel.
 - » This does not close the covert channel but renders it useless.

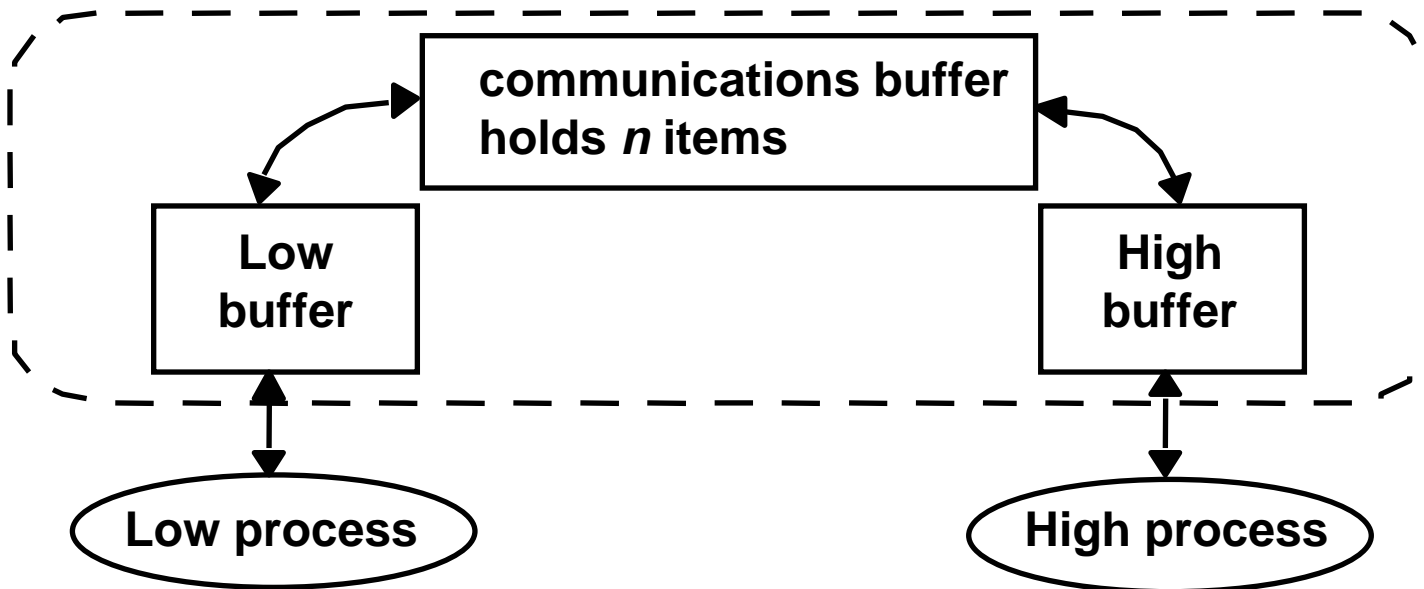
Chapter 16 CONFINEMENT PROBLEM

EXAMPLE:

The pump is a HW or SW tool for controlling a communication path between a High process and a Low process.

The messages are numbered and the communications buffer preserves messages if the pump crashes.

Under these assumptions, the processes can recover



Chapter 16 CONFINEMENT PROBLEM

- **Covert timing channel**
 - The High process can control the rate at which the pump passes messages to it.
 - The Low process fills the communications buffer by sending messages to the pump until it fails to receive an acknowledgment.
 - Low gets ACK for each message put into the buffer; no ACK for messages when the communications buffer is full.
 - At that point, the High and Low processes begin their trials.
 - Protocol:
 - At the beginning of each trial, if the High process wants to send a 1, it allows the pump to send it one of the queued messages.
 - If the High process wants to send a 0, it does not accept any message from the pump.
 - If the Low process gets an ACK, it means that a message has moved from the Low buffer to the communications buffer.
 - This can happen only if a space in the communications buffer opens.
 - This occurs when the High process reads a message.
 - Hence, if the Low process gets an ACK, the High process is signaling a 1.
 - By a similar argument, if the Low process does not get an ACK, the High process is signaling a 0.
 - Following the trial, if the Low process has received an ACK, it must send another message to the pump to enter the state required for the next trial.

Chapter 16 CONFINEMENT PROBLEM

- We assume that the Low process and the pump can process messages more quickly than the High process.
- Three cases arise:
 - Case 1:
 - The High process can process messages in less time than it takes for the Low process to get the acknowledgement.
 - This contradicts the assumption, so the pump must be artificially delaying ACKs.
 - The Low process will wait for an ACK regardless of whether the communications buffer is full or not.
 - Although this closes the covert channel, it is not optimal because the processes may wait even when they do not need to.
 - Case 2:
 - The Low process is sending messages into the pump faster than the High process can remove them.
 - Although it maximizes performance, it opens the covert channel.
 - Case 3:
 - The pump and the processes handle the messages at the same rate.
 - It balances security and performance by decreasing the bandwidth of the covert channel (w.r.t. time) and increases performance.
 - The covert channel is open but performance is not optimal.