ECS 289M Lecture 22

May 22, 2006

Information Flow Analysis

- Recall compiler-based information flow analysis
  - Exception depends upon value of variable
    - Covert channel, as exception (or lack of it) communicates information about value
  - Synchronization, IPC operations
    - One process sends message or blocks on receive; other process can detect this

Source Code Analysis

- Covert channels arise when processes can view or alter kernel variables
  - So identify variables that processes can refer to directly or view, alter indirectly

Step 1

- Identify kernel functions, processes
  - Processes are those that function at highest level of privilege, perform actions for ordinary users
- Not administrative processes, functions
  - Administrators don't need to leak anything; they have privileges to do it directly

Step 2

- Identify kernel variables accessible to user processes; processes must:
  - Control *how* variable is altered
  - Detect that variable has been altered
- Specific criteria:
  - Value of variable obtained from system call
  - Calling process can detect two or more different states of that variable

Example

```
x := f(a, b);
if x = 0 then
   x := x + 10;
return x;
```

```
y := f(a, b);
if y = 0 then
      z := 1;
else
      z := 0;
return z;
```

x directly visible as returned directly

y indirectly visible as not returned directly, but its value can be deduced from *z*, which is returned

Caveats

- Find *all* data flows through kernel
  - Need to detect all data and functional dependencies
- Record or structure
  - Consider each of its elements
- Array of structures
  - Consider each element of each structure, and array as a whole
- Pointers must be included
  - When point to variables in question

Step 3

- Analyze variables looking for covert channels
  - Method similar to that of deriving SRM
  - Results in terms of operations that alter, view variables
    - Only alter or only view: ignore operation
- Covert channel may be associated with many variables
- Variable may be associated with many covert channels

Application

- Analyze Secure Xenix kernel
- Found two variables involved in covert channels
- 4 classes of generic channels identified
  - One exploitable only when system failed
  - One could not be eliminated without changing semantics of regular Xenix
- Concluded that informal analysis would not make all associations of variables, system calls

Use of SRMM

- Examined Secure Xenix top-level specification
- SRM method failed to spot several covert channels
  - Not surprising, as the TLS did not specify data structures in which covert channels were found

Covert Flow Trees

- Idea: model flow of information through shared resource with tree
- Tree-structured representation of sequence of operations that move information from one process to another
- 5 types of nodes: goal symbols, operation symbol, failure symbol, and symbol, or symbol

Goal Symbols

- Specify states that must exist for information to flow
  - *Modification goal*: reached when attribute modified
  - *Recognition goal*: reached when attribute modification is detected
  - *Direct recognition goal*: reached when subject can detect change of attribute by direct reference or calling function that returns it

Goal Symbols

– *Inferred recognition goal*: reached when subject can detect change of attribute without direct reference or calling function that returns it
– *Inferred-via goal*: reached when information passed from one attribute to other attributes using specified system call
– *Recognize-new-state goal*: reached when attribute modified when information passed using the variable is specified by *inferred-via goal*

Other Symbols

- *Operation symbol*
  – Represents primitive operation
- *Failure symbol*
  – Information cannot be sent along this path
- *And symbol*
  – Reached when for all children (1) child is an operation; and (2) if child is a goal, it is reached
- *Or symbol*
  – Reached when for any children (1) child is an operation; or (2) if child is a goal, it is reached

Example

- Files have 3 attributed
  - *locked* true when file locked
  - *opened* true when file opened
  - *inuse* set containing PIDs of processes that have file open
- Functions
  - *read_access*(*p*, *f*) true if process *p* can read file *f*
  - *empty*(*s*) true if *s* has no elements
  - *random* returns an argument chosen at random

Operations

```
(* lock file if not locked and not opened; otherwise return 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;
```

Operations

```
(* 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 the process ID to the 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
```


Step 1

	Lockfile	*Unlockfile*	*Filelocked*	*Openfile*	*Fileopened*
reference	locked, inuse	locked	locked	locked, inuse	inuse
modify	locked	locked	∅	inuse	∅
return	∅	∅	locked	∅	inuse

Step 2

- Goal: locate covert storage channel that uses some attribute
- Do this by constructing covert flow tree
  – Type of goal controls construction

Goals

- Topmost goal: attribute be modified, modification recognized
  – one child (*and*) with two children (modification goal and recognition goal)
- Modification goal: operation modifies attribute
  – one child (*or*) with one child per operation (operation)
- Recognition goal: subject recognize, infer change in attribute
  – one child (*or*) with two children (direct recognition goal, indirect recognition goal)

Goals

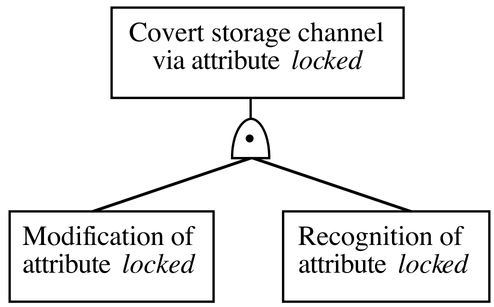
- Direct recognition goal: operation accesses attribute
  - one child (*or*) with one child per operation (operation); if none, return failure
- Inferred recognition goal: modification inferred on basis of one or more other attributes
  - one child (*or*) with one child inferred-via per operation that references some attribute and modifies some attribute
- Inferred-via goal: value of attribute be inferred via operation and recognition of new state of attribute resulting from that operation
  - one child (*and*) with two children (operation for operation used to draw inference, recognize-new-state goal)

Goals

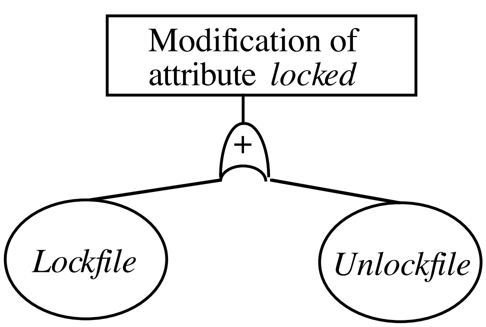
- Recognize-new-state goal: value of attribute be inferred via operation and recognition of new state of attribute resulting from that operation
  - Latter requires recognition goal for attribute
  - one child (*or*) with one recognition goal symbol child for each attribute enabling inference of modification of attribute in question
- Construction ends when all paths terminate in either operation symbol or failure symbol

Example: Tree for *locked*

- Goal state "Covert storage channel via attribute *locked*"
  - *and* node is child
  - Modification goal is "modification of *locked*"
  - Recognition goal is "recognition of *locked*"



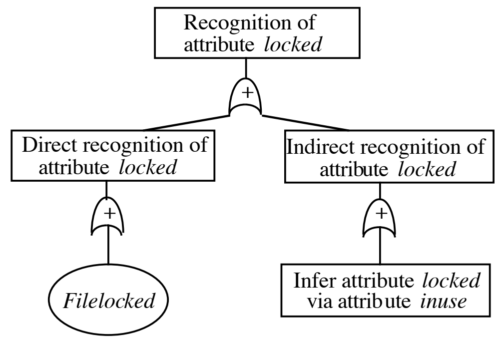
Example: Modification goal



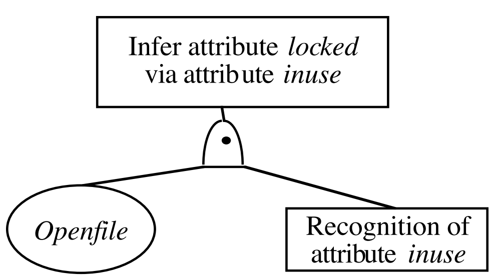
- Functions *Lockfile*, *Unlockfile* modify *locked* attribute
  - They make up the children

Example: Recognition Goal

- Direct branch: *Filelocked* returns value of *locked*
- Indirect branch: does any function modify some attribute other than *locked* after referencing *locked*
  - Attribute *inuse*



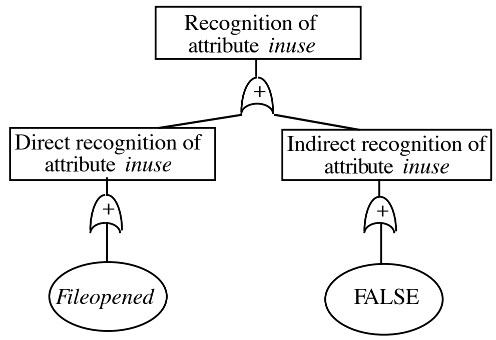
Example: Inferred Attribute

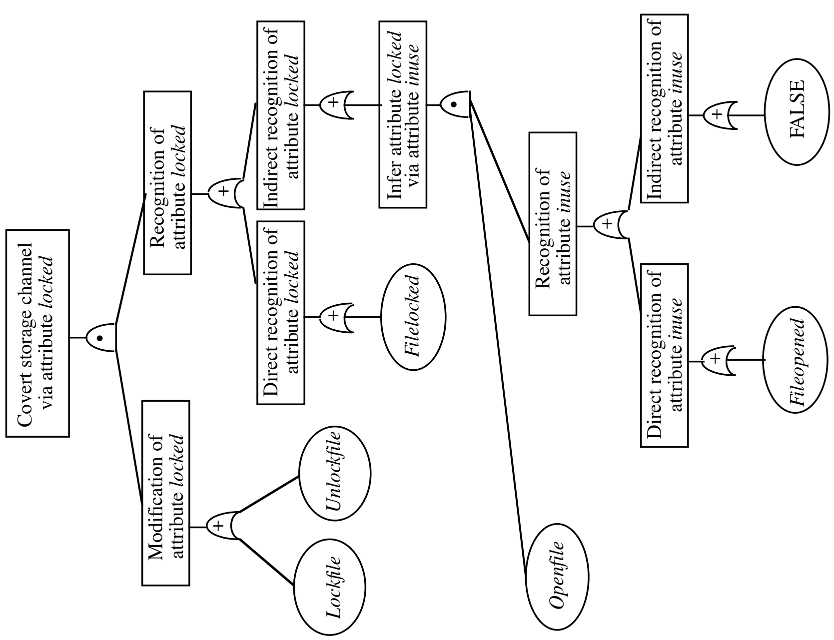


- *Openfile* uses *locked* to modify *inuse*
  - *and* node with recognition of attribute *inuse*
- Requires recognizing modification of *inuse*

Example: Recognition

- Direct recognition of change: *Fileopened*
- Indirect recognition of change: nothing





Next Step

- First list: sequences of operations modifying attribute
- Second list: sequences of operations recognizing modifications in attribute
- Information can flow along channel of sequence from first list followed by sequence from second list

Example

- List 1 = ((*Lockfile*) , (*Unlockfile*))
- List 2 = ((*Filelocked*), (*Openfile* , *Fileopened*))
- So 4 channels (sequences):
  – *Lockfile*, *Filelocked*
  – *Unlockfile*, *Filelocked*
  – *Lockfile*, *Openfile*, *Fileopened*
  – *Unlockfile*, *Openfile*, *Fileopened*

Example Attack

- *High* process sending information to *Low* process by locking, unlocking file:
  - First two channels are direct covert storage channel
  - Last two indirect covert storage channels
    - *High* process locks file (1 bit) or unlocks file (0 bit)
    - *Low* process tries to open file
    - *Low* process uses *Fileopened* to see if it worked; if so, 0 bit; if not, 1 bit

Summary

- Compared to SRMM
  - Both based on examining shared resources for reference, modification
  - Covert flow trees identifies explicit sequences of operations that cause information flow; SRM identifies channels
- How it did:
  - Covert flow trees found sequences of operations for all SRM, noninterference channels on SAT, and 1 more channel/sequence the other methods missed

Analysis

- Goal: determine at what rate information can be transmitted over a covert channel
  - Measured in "capacity" (bits) per unit time or per number of trials
  - Assumes security policy considers covert channel a serious problem
  - May or may not be true; depends entirely on threat model and operational issues

Noninterference and Capacity

- Alice sends information to Bob
- Random variables:
  - W represents inputs to machine
  - A represents inputs from Alice
  - V represents inputs not from Alice
  - B represents all possible outputs to Bob
- $I(A;B)$ amount of information transmitted over covert channel

When Is Capacity 0?

Theorem: If A, V independent and A noninterfering with B, then $I(A;B) = 0$

Proof: Sufficient to show A, B independent, or

$$p(A=a,B=b) = p(A=a)p(B=b)$$

In general,

$$p(A=a,B=b) = \Sigma_V p(A=a,B=b,V=v)$$

A noninterfering with B: deleting that part of input making up a will not change output b.

Proof

So only need to consider values of B that could result from values of V; so

$$p(A=a,B=b) = \Sigma_V\, p(A=a,V=v)p(B=b|V=v)$$

As V and A are independent,

$$p(A=a,B=b) = \Sigma_V\, p(A=a,V=V)p(B=b|V=v)$$

$$= p(A=a)(\Sigma_V p(B=b|V=v)p(V=v))$$

$$= p(A=a)p(B=b)$$

Is Noninterference Needed?

- System has:
  - 1 state bit; initially 0
  - 3 inputs, I_A, I_B, I_C
  - 1 output O_X
- Each input bit flips state bit
  - Value of state output
- Let *w* be sequence of inputs corresponding to output *x*(*w*)
  - *x*(*w*) = *length*(*w*) **mod** 2

I_A and O_X

- I_A not noninterfering with O_X
  - Delete inputs from I_A, changes length of output and hence value of *x*(*w*)
- Let:
  - *W* represents length of input sequences
  - *A* represents length of components of input subsequence contributed by I_A
  - *V* represents length of components of input subsequence not contributed by I_A
    - *A*, *V* independent
  - *X* represents output state

Case 1

- If $V = 0$, then:

$$W = (A + V) \textbf{ mod } 2 = A \textbf{ mod } 2$$

- So W, I dependent
- So are A, X
- Hence $I(A; X) \neq 0$

Case 2

Let I_B, I_C produce inputs such that

$$p(V=0) = p(V=1) = 0.5$$

Then:

$$p(X=x) = p(V=x,A=0)+p(V=1-x,A=1)$$

By independence of A, I:

$$p(X=x) = p(V=x)p(A=0)+p(V=1-x)p(A=1)$$

So $p(X=x) = 0.25+0.25 = 0.5$

$p(X=x|A=a)=p(X=(a+x) \text{ mod } 2) = 0.5$

So A and X independent, giving $I(A;X) = 0$

Meaning

- Covert channel capacity will be 0 if:
  - Input noninterfering with output, or
  - Input sequence comes from independent sources *and* all possible values from at least 1 source equiprobable
    - In effect, distribution “hides” interference