Lecture 22
November 18, 2024

Execution-Based Mechanisms

- Detect and stop flows of information that violate policy
  - Done at run time, not compile time
- Obvious approach: check explicit flows
  - Problem: assume for security, $\underline{x} \leq \underline{y}$

```
if x = 1 then y := a;
```

  - When $x \neq 1$, $\underline{x}$ = High, $\underline{y}$ = Low, $\underline{a}$ = Low, appears okay—but implicit flow violates condition!

Fenton's Data Mark Machine

- Each variable has an associated class
- Program counter (PC) has one too
- Idea: branches are assignments to PC, so you can treat implicit flows as explicit flows
- Stack-based machine, so everything done in terms of pushing onto and popping from a program stack

Instruction Description

- *skip*: instruction not executed
- *push(x, x)*: push variable *x* and its security class *x* onto program stack
- *pop(x, x)* : pop top value and security class from program stack, assign them to variable *x* and its security class *x* respectively

Instructions

- $x := x + 1$ (increment)
  - Same as:
    if $\underline{PC} \leq \underline{x}$ **then** $x := x + 1$ **else** *skip*
- **if** $x = 0$ **then goto** n **else** $x := x - 1$ (branch and save PC on stack)
  - Same as:

```
if x = 0 then begin
  push(PC, PC); PC := lub{PC, x}; PC := n;
 end else if PC ≤ x then
  x := x - 1
else
  skip;
```

More Instructions

- **if'** $x = 0$ **then goto** n **else** $x := x - 1$ (branch without saving PC on stack)
  - Same as:

```
if x = 0 then
  if x ≤ PC then PC := n else skip
else
  if PC ≤ x then x := x - 1 else skip
```

More Instructions

- **`return`** (go to just after last *if*)
  - Same as:
    `pop``(`*`PC`*`, `*`PC`*`);`
- **`halt`** (stop)
  - Same as:
    `if` *`program stack empty`* **`then`** *`halt`*
  - Note stack empty to prevent user obtaining information from it after halting

Example Program

```
1   if x = 0 then goto 4 else x := x - 1
2   if z = 0 then goto 6 else z := z - 1
3   halt
4   z := z + 1
5   return
6   y := y + 1
7   return
```

Initially $x = 0$ or $x = 1$, $y = 0$, $z = 0$

Program copies value of x to y

Example Execution: Initial Setting

x	*y*	*z*	*PC*	*PC*	*stack*	*check*
1	0	0	1	Low	—	

Example Execution: Step 1

x	*y*	*z*	*PC*	*PC*	*stack*	*check*
1	0	0	1	Low	—	
0	0	0	2	Low	—	Low ≤ *x*

```
if x = 0 then goto 4 else x := x - 1
```

```
if x = 0 then begin
   push(PC, PC); PC := lub{PC, x}; PC := n;
end else if PC ≤ x then
   x := x - 1
else
   skip;
```

Example Execution: Step 2

x	*y*	*z*	*PC*	*PC*	*stack*	*check*
1	0	0	1	Low	—	
0	0	0	2	Low	—	Low ≤ *x*
0	0	0	6	*z*	(3, Low)	*PC* ≤ *y*

```
if z = 0 then goto 6 else z := z - 1
```

```
if z = 0 then begin
   push(PC, PC); PC := lub{PC, z}; PC := n;
end else if PC ≤ z then
   z := z - 1
else
   skip;
```

Example Execution: Step 3

x	*y*	*z*	*PC*	*PC*	*stack*	*check*
1	0	0	1	Low	—	
0	0	0	2	Low	—	Low ≤ *x*
0	0	0	6	*z*	(3, Low)	*PC* ≤ *y*
0	1	0	7	*z*	(3, Low)	

`y := y + 1`

if *PC* ≤ *y* **then** `y := y + 1` **else** *skip*

Example Execution: Step 4

x	*y*	*z*	*PC*	*PC*	*stack*	*check*
1	0	0	1	Low	—	
0	0	0	2	Low	—	Low ≤ *x*
0	0	0	6	*z*	(3, Low)	*PC* ≤ *y*
0	1	0	7	*z*	(3, Low)	

`return`

`pop``(`*PC*`,` *PC*`);`

Example Execution: Step 5

x	*y*	*z*	*PC*	$\underline{PC}$	*stack*	*check*
1	0	0	1	Low	—	
0	0	0	2	Low	—	Low ≤ $\underline{x}$
0	0	0	6	$\underline{z}$	(3, Low)	$\underline{PC}$ ≤ $\underline{y}$
0	1	0	7	$\underline{z}$	(3, Low)	
0	1	0	3	Low	—	
halt						

if *program stack empty* **then** *halt*

Handling Errors

- Ignore statement that causes error, but continue execution
  - If aborted or a visible exception taken, user could deduce information
  - Means errors cannot be reported unless user has clearance at least equal to that of the information causing the error

Variable Classes

- Up to now, classes fixed
  - Check relationships on assignment, etc.
- Consider variable classes
  - Fenton's Data Mark Machine does this for $\underline{PC}$
  - 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: integer class { x };
                  var y: integer class { y })
var z: integer 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 $\underline{y} < \underline{x}$ (that is, $\underline{x}$ strictly dominates $\underline{y}$; they are not equal)

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 $\underline{y} < \underline{x}$

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 $\underline{x} \leq \underline{z}$
  - This holds if and only if $\underline{x}$ = Low
    - Not possible as $\underline{y} < \underline{x}$ = Low by assumption and there is no class that Low strictly dominates

Integrity Mechanisms

- The above also works with Biba, as it is mathematical dual of Bell-LaPadula
- All constraints are simply duals of confidentiality-based ones presented above

Example 1

For information flow of assignment statement:

$$y := f(x_1, \ldots, x_n)$$

the relation glb{ $\underline{x}_1$, …, $\underline{x}_n$ } ≥ $\underline{y}$ must hold

- Why? Because information flows from x_1, …, x_n to y, and under Biba, information must flow from a higher (or equal) class to a lower one

Example 2

For information flow of conditional statement:

if $f(x_1, \ldots, x_n)$ **then** S_1**; else** S_2**; end;**

then the following must hold:

- S_1, S_2 must satisfy integrity constraints
- glb{ $\underline{x}_1$, …, $\underline{x}_n$ } ≥ lub{$\underline{y}$ | y target of assignment in S_1, S_2 }

Example Information Flow Control Systems

- Privacy and Android Cell Phones
  - Analyzes data being sent from the phone
- Firewalls

Privacy and Android Cell Phones

- Many commercial apps use advertising libraries to monitor clicks, fetch ads, display them
  - So they send information, ostensibly to help tailor advertising to you
- Many apps ask to have full access to phone, data
  - This is because of complexity of permission structure of Android system
- Ads displayed with privileges of app
  - And if they use Javascript, that executes with those privileges
  - So if it has full access privilege, it can send contact lists, other information to others
- Information flow problem as information is flowing from phone to external party

Analyzing Android Flows

- Android based on Linux
  - App executables in bytecode format (Dalvik executables, or DEX) and run in Dalvik VM
  - Apps event driven
  - Apps use system libraries to do many of their functions
  - Binder subsystem controls interprocess communication
- Analysis uses 2 security levels, *untainted* and *tainted*
  - No categories, and *tainted* < *untainted*

TaintDroid: Checking Information Flows

- All objects tagged *tainted* or *untainted*
  - Interpreters, Binder augmented to handle tags
- Android native libraries trusted
  - Those communicating externally are *taint sinks*
- When untrusted app invokes a taint sink library, taint tag of data is recorded
- Taint tags assigned to external variables, library return values
  - These are assigned based on knowledge of what native code does
- Files have single taint tag, updated when file is written
- Database queries retrieve information, so tag determined by database query responder

TaintDroid: Checking Information Flows

- Information from phone sensor may be sensitive; if so, *tainted*
  - TaintDroid determines this from characteristics of information
- Experiment 1 (2010): selected 30 popular apps out of a set of 358 that required permission to access Internet, phone location, camera, or microphone; also could access cell phone information
  - 105 network connections accessed *tainted* data
  - 2 sent phone identification information to a server
  - 9 sent device identifiers to third parties, and 2 didn't tell user
  - 15 sent location information to third parties, none told user
  - No false positives

TaintDroid: Checking Information Flows

- Experiment 2 (2012): revisited 18 out of the 30 apps (others did not run on current version of Android)
  - 3 still sent location information to third parties
  - 8 sent device identification information to third parties without consent
    - 3 of these did so in 2010 experiment
    - 5 were new
  - 2 new flows that could reveal *tainted* data
  - No false positives

Firewalls

- Host that mediates access to a network
  - Allows, disallows accesses based on configuration and type of access
- Example: block Conficker worm
  - Conficker connects to botnet, which can use system for many purposes
    - Spreads through a vulnerability in a particular network service
  - Firewall analyze packets using that service remotely, and look for Conficker and its variants
    - If found, packets discarded, and other actions may be taken
  - Conficker also generates list of host names, tried to contact botnets at those hosts
    - As set of domains known, firewall can also block outbound traffic to those hosts

Filtering Firewalls

- Access control based on attributes of packets and packet headers
  - Such as destination address, port numbers, options, etc.
  - Also called a *packet filtering firewall*
  - Does not control access based on content
  - Examples: routers, other infrastructure systems

Proxy

- Intermediate agent or server acting on behalf of endpoint without allowing a direct connection between the two endpoints
  - So each endpoint talks to proxy, thinking it is talking to other endpoint
  - Proxy decides whether to forward messages, and whether to alter them

Proxy Firewall

- Access control done with proxies
  - Usually bases access control on content as well as source, destination addresses, etc.
  - Also called an *applications level* or *application level firewall*
  - Example: virus checking in electronic mail
    - Incoming mail goes to proxy firewall
    - Proxy firewall receives mail, scans it
    - If no virus, mail forwarded to destination
    - If virus, mail rejected or disinfected before forwarding

Example

- Want to scan incoming email for malware
- Firewall acts as recipient, gets packets making up message and reassembles the message
  - It then scans the message for malware
  - If none, message forwarded
  - If some found, mail is discarded (or some other appropriate action)
- As email reassembled at firewall by a mail agent acting on behalf of mail agent at destination, it's a proxy firewall (application layer firewall)

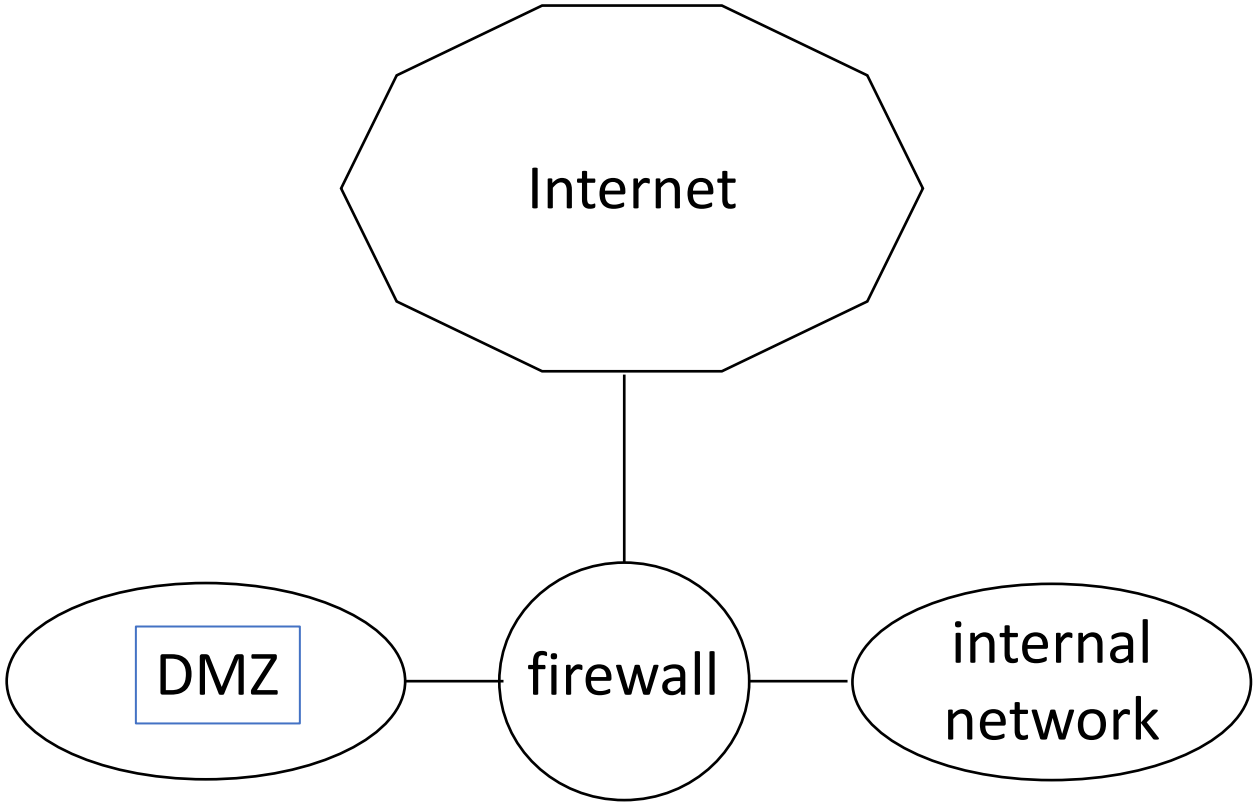
Stateful Firewall

- Keeps track of the state of each connection
- Similar to a proxy firewall
  - No proxies involved, but this can examine contents of connections
  - Analyzes each packet, keeps track of state
  - When state indicates an attack, connection blocked or some other appropriate action taken

Network Organization: DMZ

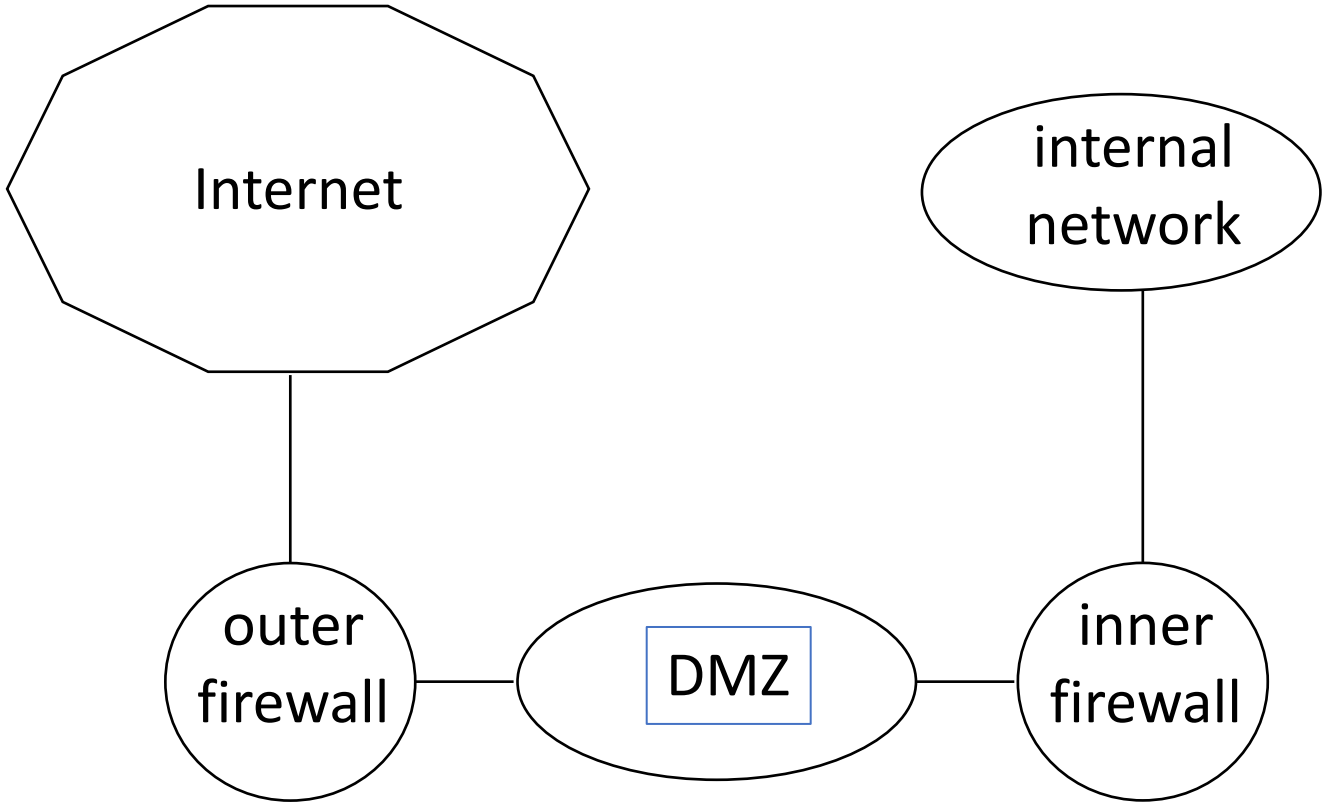
- DMZ is portion of network separating a purely internal network from external network
- Usually put systems that need to connect to the Internet here
- Firewall separates DMZ from purely internal network
- Firewall controls what information is allowed to flow through it
  - Control is bidirectional; it control flow in both directions

One Setup of DMZ



One dual-homed firewall that routes messages to internal network or DMZ as appropriate

Another Setup of DMZ



Two firewalls, one (outer firewall) connected to the Internet, the other (inner firewall) connected to internal network, and the DMZ is between the firewalls

Attacks

- *Attack*: a sequence of actions creating a violation of a security policy
  - *Multistage attack*: attack requiring several steps to achieve its goal
- *Goal of the attack*: what the attacker hopes to achieve
- *Target of the attack*: entity that the attacker wishes to affect
- Example: burglar stealing someone's jewelry
  - *Attack*: what she does to steal the jewelry; probably *multistage* (break window, find jewelry box, break it open, take jewelry, get out of house)
  - *Goal of the attack*: steal the jewelry
  - *Target of the attack*: the jewelry, also the owner of the jewelry

Representing Attacks

- Can be done at many levels of abstraction
- As you go deeper, some steps become more detailed and break down into multiple steps themselves
- *Subgoal*: the goal of each step to move the attacker closer to the goal of the attack

Example: Penetration of Corporate Computer System

- Goal: gain access to corporate computer system
- Procedure was to try to get people to reveal account information, change passwords to something the attackers knew
  - Target: newly-hired employees who hadn't had computer security awareness briefing
  - Subgoal 1: find those people
  - Subgoal 2: get them to reveal account info, change passwords

Focus on Subgoal 1

- For subgoal 1, needed to find list of these people
  - Subgoal 1-1: learn about company's organization
- Procedure was to get annual report (public), telephone directory (not public)
  - Subgoal 1-2: acquire the telephone directory (this required 2 numbers)
  - Subgoal 1-3: get the two numbers (only available to employees)
  - Subgoal 1-4: impersonate employees
- Had corporate controls blocked attackers from achieving subgoal, they would need to find other ways of doing it

Attack Trees

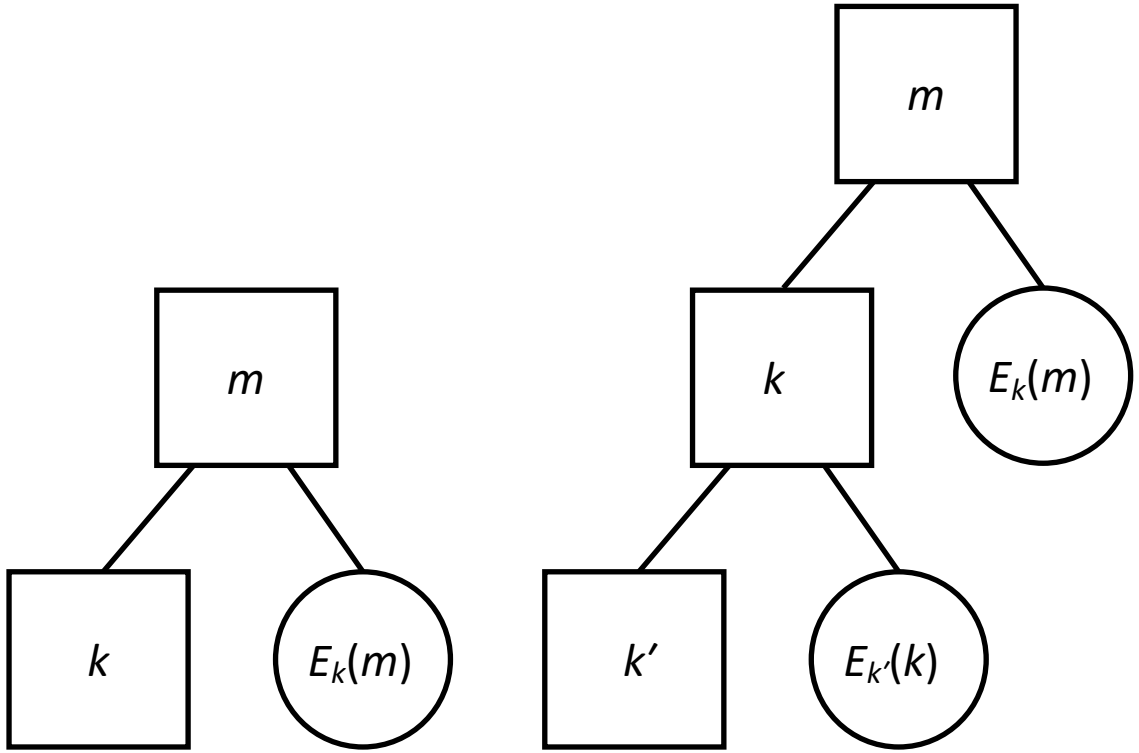
- Represent the goals and subgoals as a sequence of hierarchical nodes in a tree
  - Goal is the root

Security Flaws in Cryptographic Key Management Schemes

- Goal: develop package to allow attackers to ask what data is needed to determine encryption key
- System has only 2 functions, $c = E_k(m)$ and $m = D_k(c)$
- Attack ("search") tree has the required information represented as root node, other nodes represent subgoals
- 2 types of nodes
  - Required: represents information necessary for parent; *satisfied* when that information becomes available
  - Available: represents known information
- As tree constructed, find leaf nodes that are required (using breadth-first search), construct additional layer

Example

- Assume Sage knows $E_k(m)$, $E_{k'}(k)$, k'
  - Nodes for these are available nodes
- Goal: determine m
  - Node representing m is required node
- Tree construction:
  - To get m, use k to decrypt $E_k(m)$ (left tree)
  - To get k, determine if it is encrypted and if so, try to decrypt it (right tree)
- Now all leaves are available nodes



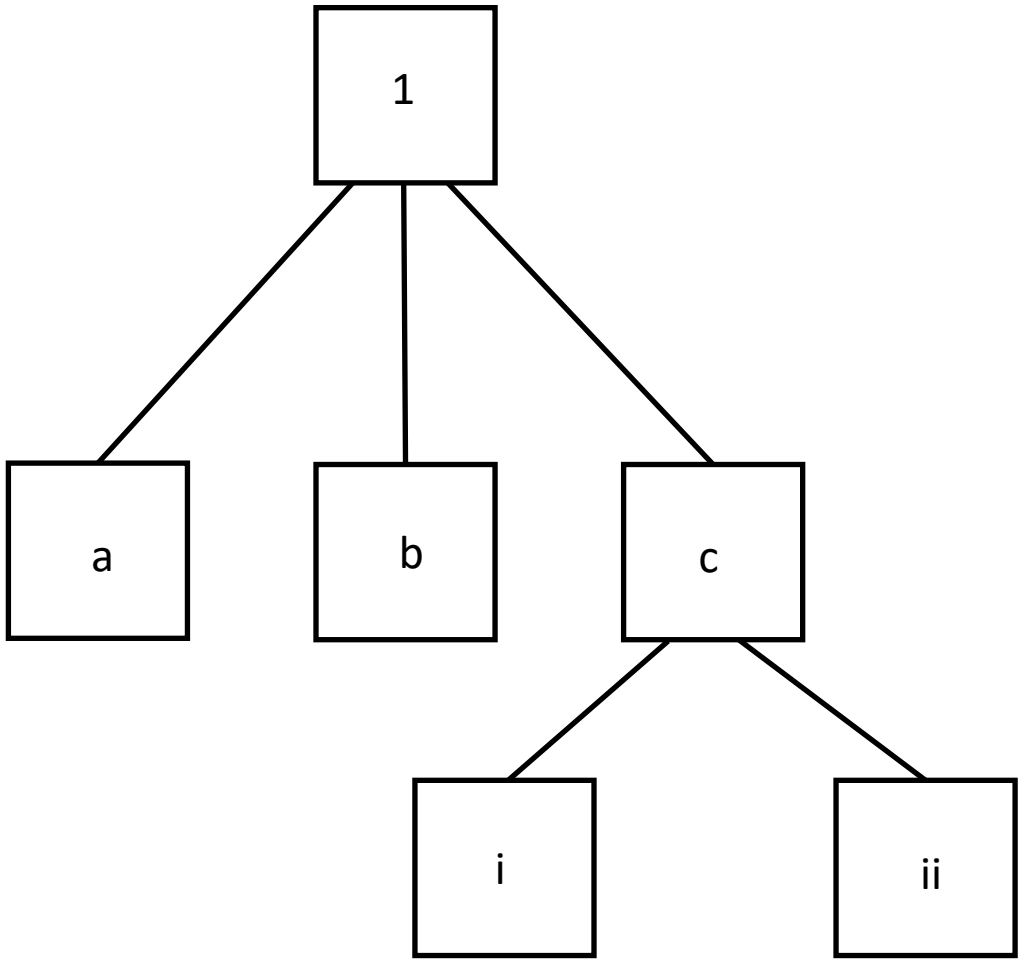
Schneier's Attack Trees

- Two types of nodes
  - *And* nodes require all children to be satisfied before it is satisfied
  - *Or* nodes require at least 1 of its children to be satisfied before it is satisfied
  - *Weight* of node indicates some relevant characteristic, like difficulty of satisfying node
    - Weights of interior nodes depend upon weights of child nodes
    - Weights of leaf nodes assigned externally
- Goal represented as root node of set of tree
- Determine the steps needed to satisfy the goal
  - These become children of the root
- Repeat that step for each child
  - Stop when leaf nodes are at appropriate level of abstraction

Example: Reading PGP-Encrypted Message

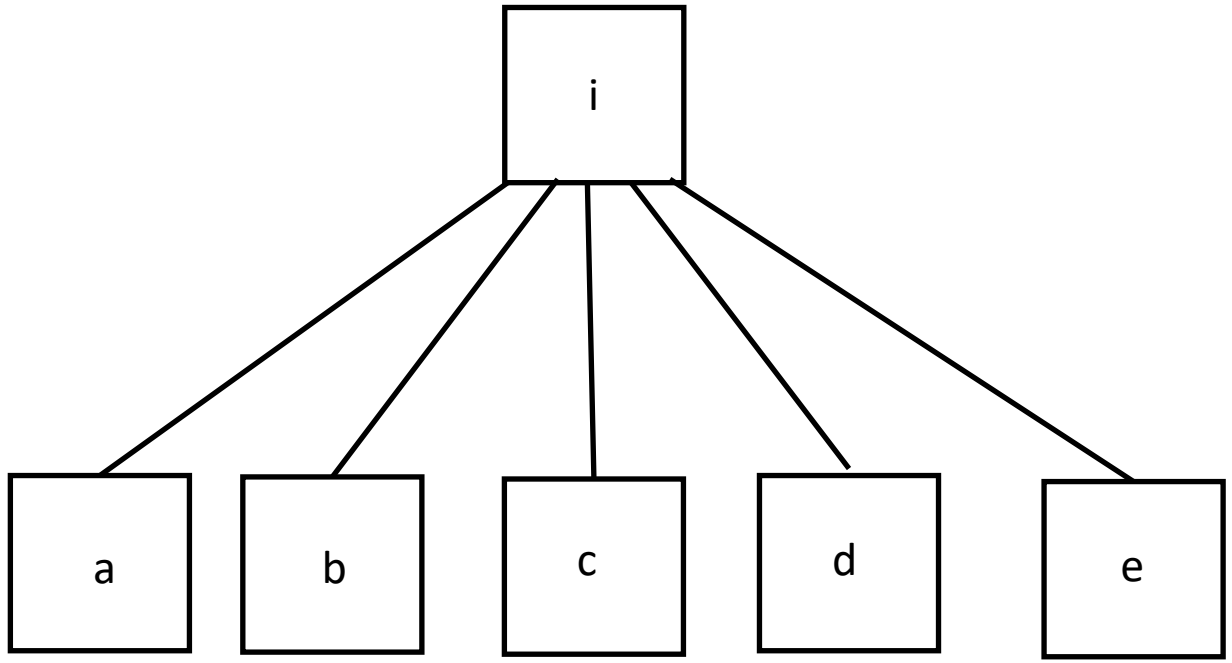
- Sage wants to read message Skyler sends to Caroline
- Five ways:
  1. Read message before Skyler encrypts it
  2. Read message after Caroline decrypts it
  3. Break encryption used to encrypt message
  4. Determine symmetric key used to encrypt message
  5. Obtain Caroline's private key
- Focus on 2, read message after Caroline decrypts it

Beginning the Tree



1. Read message after Caroline decrypts it
   a. Monitor Caroline's outgoing mail; or
   b. Add a "Reply-To:" header (or change an existing one); or
   c. Compromise Caroline's computer and read the decrypted message
      i. Compromise Caroline's computer; and
      ii. Read the decrypted message

Next Layer



i. Read message after Caroline decrypts it
   a. Copy decrypted message from memory; or
   b. Copy decrypted message from secondary storage; or
   c. Copy decrypted message from backup; or
   d. Monitor network to observe Caroline sending the plaintext message; or
   e. Use a Van Eyk device to monitor the display of the message on Caroline's screen as it is displayed there

Textual Representation

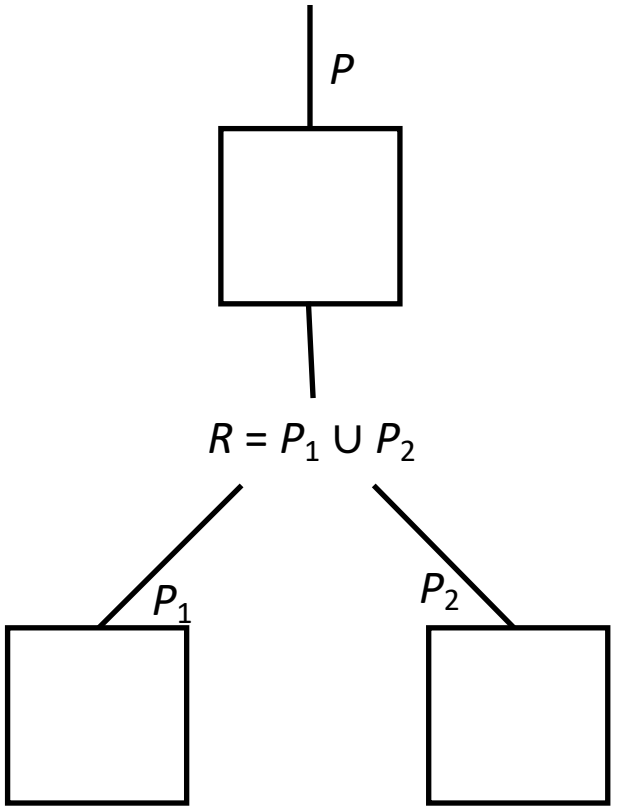
1. Read a message that Skyler is sending to Caroline. (OR)
    1.1. Read the message before Skyler encrypts it.
    1.2. Read the message after Caroline decrypts it. (OR)
        1.2.1. Monitor Caroline's outgoing mail.
        1.2.2. Add a "Reply-To" field to the header (or change the address in the existing "Reply-To" field).
        1.2.3. Compromise Caroline's computer and read the decrypted message. (AND)
            1.2.3.1. Compromise Caroline's computer. (OR)
                1.2.3.1.1. Copy decrypted message from memory.
                1.2.3.1.2. Copy decrypted message from secondary storage.
                1.2.3.1.3. Copy decrypted message from backup.
                1.2.3.1.4. Monitor network to observe Caroline sending the cleartext message.
                1.2.3.1.5. Use a Van Eck device to monitor the display of the message on Caroline's monitor as it is displayed.
            1.2.3.2. Read the decrypted message.
    1.3. Break the encryption used to encrypt the message.
    1.4. Determine the symmetric key used to encrypt the message.
    1.5. Obtain Caroline's private key.

Requires/Provides Model

- Generalization of attack trees
- Based on *capabilities*, semantic objects encapsulating semantically typed attributes
  - Represent information or a situation to advance an attack
- *Concept* is a set *C* of capabilities and a mapping from *C* to another set of capabilities that are provided
  - Description of subgoal of attack
  - Attacker has a set of *required* capabilities *R* to reach subgoal; it then acquires a set *P* of provided capabilities

Concept

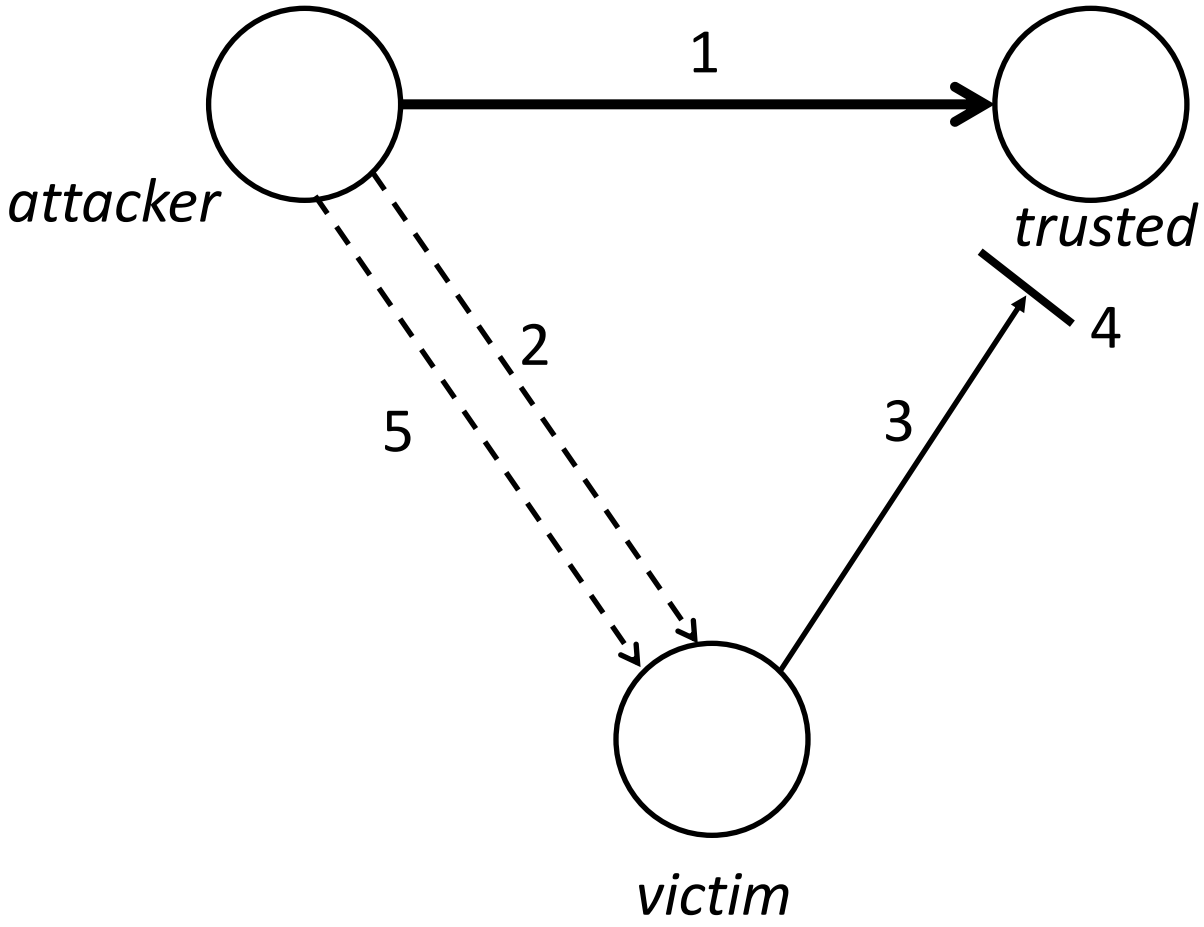
- *Concept* is a set *R* of capabilities and a mapping from *R* to another set *P* of capabilities that are provided
  - Description of subgoal of attack
- Interpretation: attacker has a set of *required* capabilities *R* to reach subgoal; it then acquires a set *P* of *provided* capabilities



Concept

- Captures *effect* of attack
  - How the attack works (ie, how capabilities are required) irrelevant to concept; that attacker has them is what matters
- Moves away from having to know every method of attack to get to a step
  - Concept embodies the step, so all model needs is required capabilities
- Can compose attacks based solely on effects and not methods of attack

Example: *rsh* Attack



1. *attacker* launches a DoS against *trusted*
2. *attacker* sends *victim* forged SYN, apparently from *trusted*
3. *victim* sends SYN/ACK to *trusted*
4. It never gets there due to DoS
5. *attacker* sends forged SYN/ACK to *trusted*, with command in data segment of packet
   - Need to know right sequence number
   - If so, causes command to be executed as though *trusted* requested it

Example: *rsh* Attack

- *Requires* capability: blocking of a connection between the *trusted* and *victim* hosts
  - Contains source address, destination address
  - Also time interval indicating when communication is blocked (ie, when the DoS attack is under way, and how long it lasts)
- *Provides* capability: execute command on *victim* host as if command were from *trusted* host
- *Concept*: spoof *trusted* host to *victim* host