Lecture 26
December 1, 2023

Anonymity Itself

- Some purposes for anonymity
  - Removes personalities from debate, or with appropriate choice of pseudonym, shape course of debate by implication
  - Prevent retaliation
  - Protect privacy
- Are these benefits or drawbacks?
  - Depends on society, and who is involved

Pseudonyms

- Names of authors of documents used to imply something about the document
- Example: *U.S. Federalist Papers*
  - These argued for the states adopting the U.S. Constitution
  - Real authors were Alexander Hamilton, James Madison, John Jay, all Federalists who wanted the Constitution adopted
  - But using alias “Publius” hid their names
    - Debate could focus on content of the *Federalist Papers*, not the authors or their personalities
    - Roman Publius seen as a model governor, implying the *Papers* represented responsible political philosophy, legislation

Whistleblowers

- Criticism of powerholders often fall into disfavor; powerholders retaliate, but anonymity protects these critics
  - Example: Anonymous sources spoke to Woodward and Bernstein, during U.S. Watergate scandal in 1970s; one important source, called “Deep Throat”, provided guidance that helped uncover a pattern of activity leading to impeachment articles against President Nixon and his resignation
    - “Deep Throat” later revealed as an assistant director of Federal Bureau of Investigation; had this been known, he would have been fired and might have been prosecuted
  - Example: Galileo openly held Copernican theory of the earth circling the sun; brought before the Inquisition and forced to recant

Privacy

- Anonymity protects privacy by obstructing amalgamation of individual records
- Important, because amalgamation poses 3 risks:
  - Incorrect conclusions from misinterpreted data
  - Harm from erroneous information
  - Not being let alone
- Also hinders monitoring to deter or prevent crime
- Conclusion: anonymity can be used for good or ill
  - Right to remain anonymous entails responsibility to use that right wisely

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

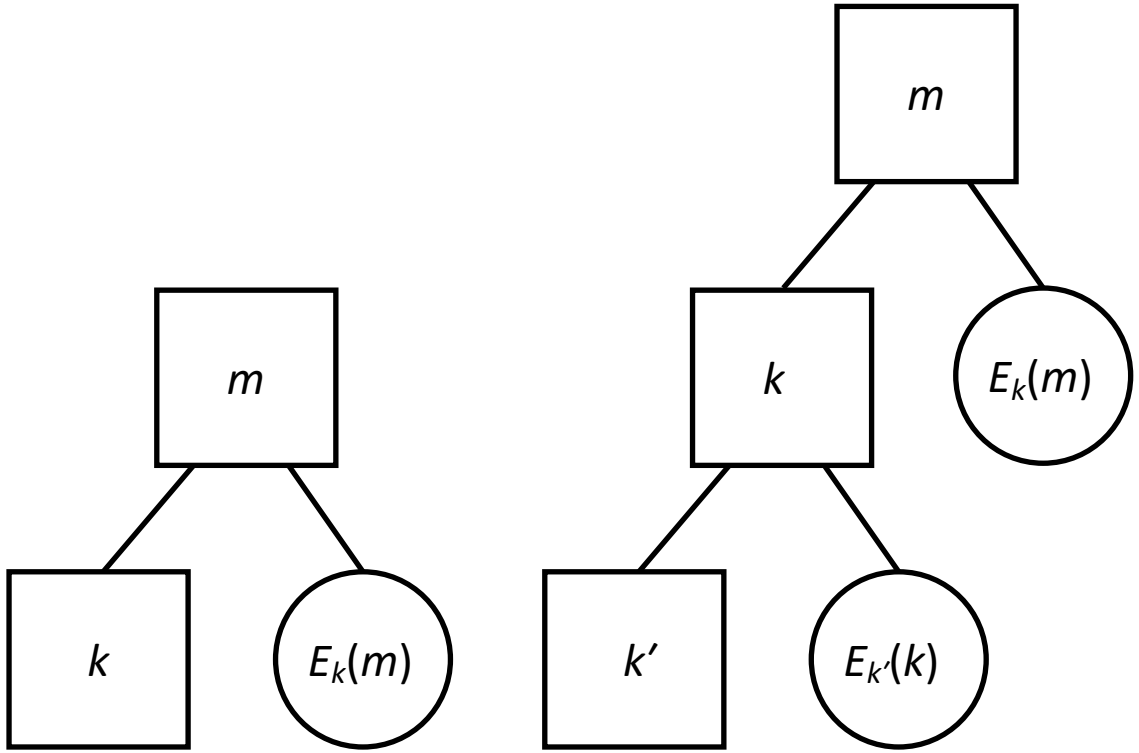
- Represent the goals and subgoals as a sequence of hierarchical nodes in a tree
  - Goal is the root
  - Interior nodes are subgoals
  - Leaves are starting points

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



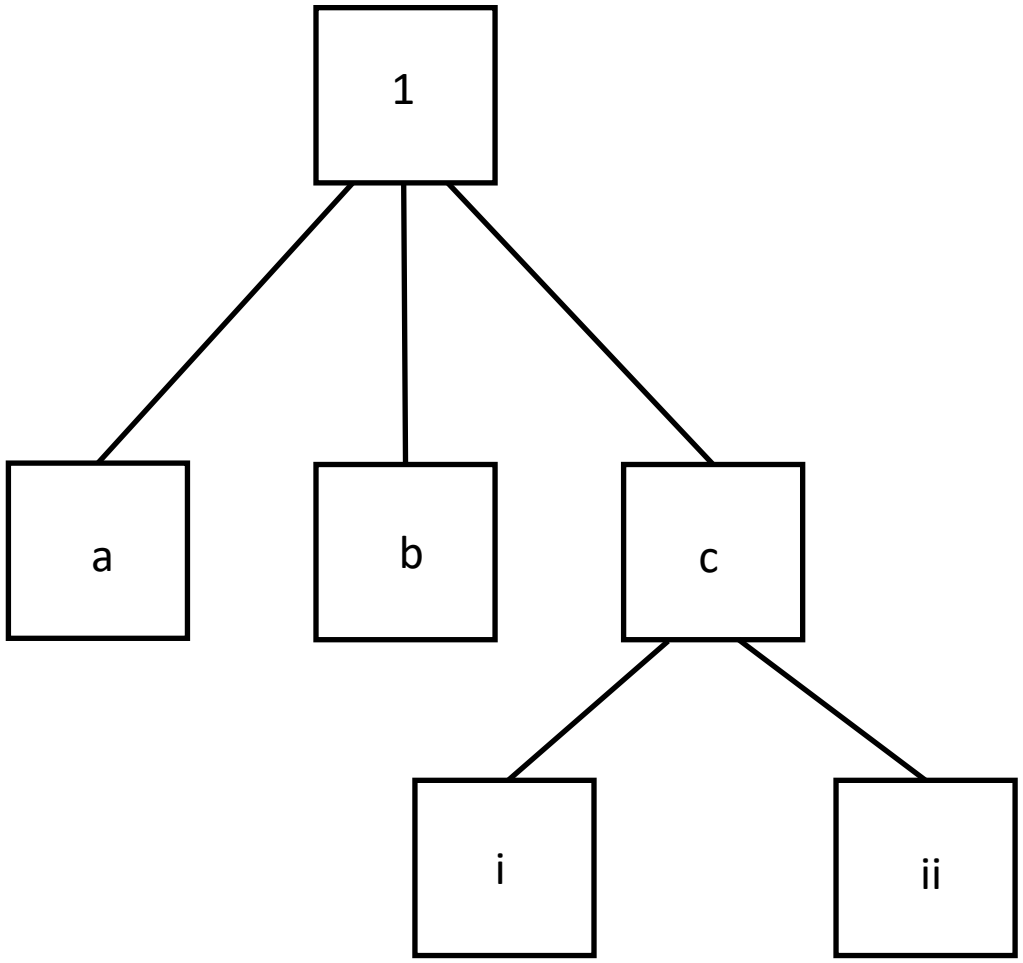
Schneier's Attack Trees

- 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

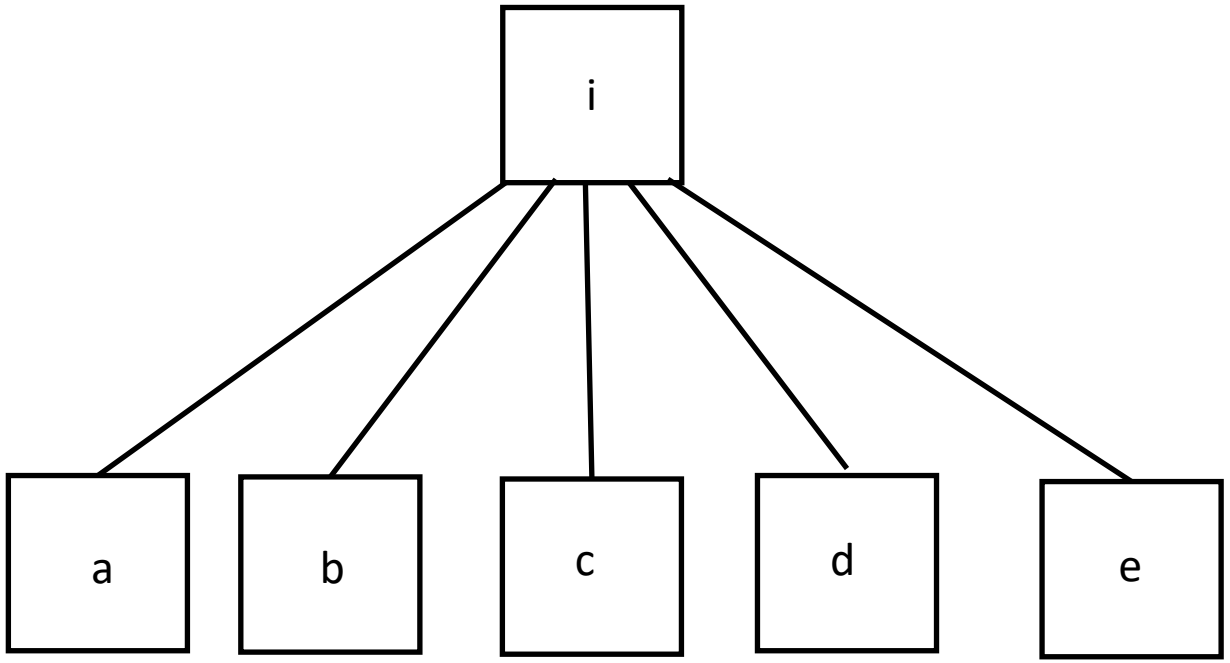
- Sage wants to read message Skyler sends to Caroline
- Five ways:
  1. Read message before Skyler encrypts it
  2. Read message after Caroline decrypts it
  3. Break encryption used to encrypt message
  4. Determine symmetric key used to encrypt message
  5. Obtain Caroline's private key
- Focus on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

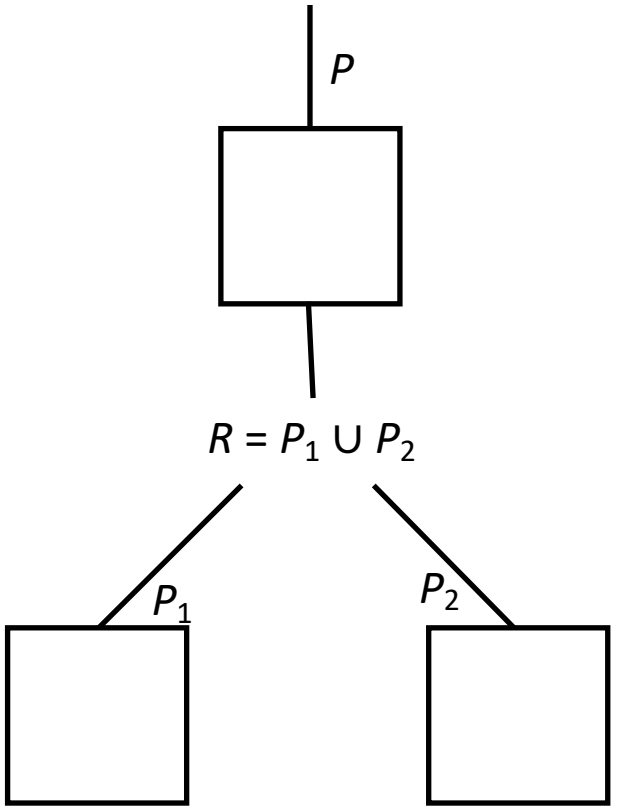
1. Read a message that Skyler is sending to Caroline. (OR)
    1.1. Read the message before Skyler encrypts it.
    1.2. Read the message after Caroline decrypts it. (OR)
        1.2.1. Monitor Caroline's outgoing mail.
        1.2.2. Add a "Reply-To" field to the header (or change the address in the 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

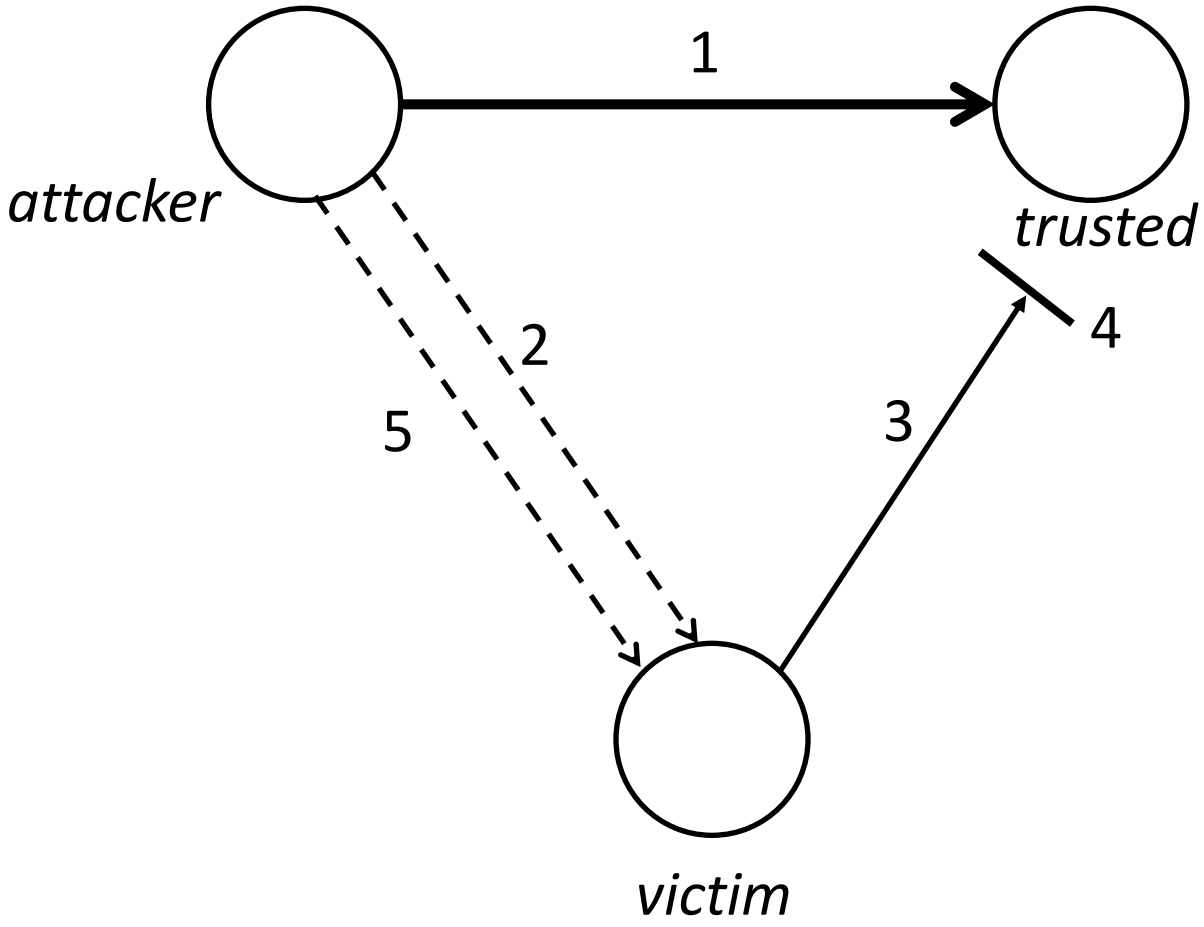
- *Concept* is a set *R* of capabilities and a mapping from *R* to another set *P* of capabilities that are provided
  - Description of subgoal of attack
- Interpretation: attacker has a set of *required* capabilities *R* to reach subgoal; it then acquires a set *P* of *provided* capabilities



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

JIGSAW Language

- Implements requires/provides model
- Capabilities: sets of typed attributes and values
  - **`extern`** keyword means it is defined elsewhere
- Concepts: two sets of capabilities
  - Required capabilities in **`requires`** block
  - Provided capabilities in **`provides`** block
  - **`action`** block lists actions to take when a concept is active

Example: JIGSAW Representation of *rsh* Attack

```
capability nosend is
      true_src, src, dst: type Host; # attacker, trusted, victim
      using: type Service;           # service to be exploited
end.
```

Structure of a capability:

- *using* is command to be executed, exploiting a service (here, *rsh*)

Example: JIGSAW Representation of *rsh* Attack

```
concept rsh_connection_spoofing is
        requires
                TP: type Trusted_Partner;          #- trusted host
                SA: type Active_Service;           #- service (here, rshd)
                PPS: type Prevent_Packet_Send;
                FPS: type Forged_Packet_Send;
                extern SNP: type SeqNumProbe;
```

PPS: capability for *true_src* to block *src* host receiving packets from *dst*

FPS: capability for *true_src* to send forget packet to *dst*

SNP: capability for *true_src* to determine next sequence number of *dst*

Example: JIGSAW Representation of *rsh* Attack

```
with          #- These instantiate the capabilities
      TP.service is RSH,          #- service is RSH
      PPS.host is TP.trusted,           #- blocked host = trusted host
      FPD.dst.host is TP.trustor,       #- spoofed packets go to host
                                        #-  trusting TP
      FPS.src is [PPS.host, PPS.port], #- apparent source of forged
                                        #-  packets is blocked
      SNP.dst is [SA.host, SA.port],  #- probed host must be
      SA.port is TCP/RSH,               #-  running RSH on usual port
      SA.service is RSH,
      SNP.dst is FPS.dst                #- forged packets go to probed
      active(FPS) during active(PPS)  #-  host while DoS of trusted
                                        #-  host is active
```

Example: JIGSAW Representation of *rsh* Attack

To meet **`requires`** conditions, relationships in **`with`** block must hold:

- Trusted host must be running *rsh* servicve
- Attacker must be able to block trusted host from sending packets to victim
- Attacker must be able to send spoofed packets ostensibly from trusted host to victim
- Attacker must know sequence number of packet victim sends to trusted host
- When attack on victim is being carried out, attack on trusted host must also be active

Example: JIGSAW Representation of *rsh* Attack

```
requires
        PSC: type push_channel;
        REX: type remote_execution;
```

PSC: capability to send code, commands to *dst*

REX: capability to execute that code, commands on *dst*

Example: JIGSAW Representation of *rsh* Attack

```
with          #- These set the new capabilities
      PSC.src <- FPS.true_src,          #- capability to move code from
      PSC.dst <- FPS.dst,               #-  attacker to rsh server
      PSC.true_src <- FPS.true_src,     #-  (victim)
      PSC.using <- rsh;
      REX.src <- FPS.true_src,          #- capability to execute code,
      REX.dst <- FPS.dst,               #-  commands on rsh server
      REX.true_src <- FPS.true_src,     #-  (victim)
      REX.using <- rsh;
end;
action
      true -> report("rsh connection spoofing: " + TP.hostname)
end;
```

Example: JIGSAW Representation of *rsh* Attack

- When all conditions in **`requires`** block satisfied, concept *rsh_connection_spoofing* is realized
- Attacker gets capabilities defined in **`provides`** section
  - Here, *PSC* and *REX* capabilities
- Events in **`action`** block executed
  - Here, message is printed to alert observer an *rsh* spoofing attack under way