Lecture 15: Covert Channels, Assurance

- Covert channels
  - Detection
  - Mitigation
- The pump
- Why assurance?
- Trust and assurance
- Life cycle and assurance

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

Measuring Capacity

- Intuitively, difference between unmodulated, modulated channel
  - Normal uncertainty in channel is 8 bits
  - Attacker modulates channel to send information, reducing uncertainty to 5 bits
  - Covert channel capacity is 3 bits
    - Modulation in effect fixes those bits

Formally

- Inputs:
  - A input from Alice (sender)
  - V input from everyone else
  - X output of channel
- Capacity measures uncertainty in X given A
- In other terms: maximize

$$I(A; X) = H(X) - H(X \mid A)$$

with respect to A

Example (continued)

- If A, V independent, $p=p(A=0)$, $q=p(V=0)$:
  - $p(A=0, V=0) = pq$
  - $p(A=1, V=0) = (1-p)q$
  - $p(A=0, V=1) = p(1-q)$
  - $p(A=1, V=1) = (1-p)(1-q)$
- So
  - $p(X=0) = p(A=0, V=0)+p(A=1, V=1) = pq + (1-p)(1-q)$
  - $p(X=1) = p(A=0, V=1)+p(A=1, V=0) = (1-p)\mathrm{q} + p(1-q)$

More Example

- Also:
  - $p(X=0|A=0) = q$
  - $p(X=0|A=1) = 1-q$
  - $p(X=1|A=0) = 1-q$
  - $\mathrm{p}(X=1|A=1) = q$
- So you can compute:
  - $H(X) = -[(1-p)q + p(1-q)] \lg [(1-p)q + p(1-q)]$
  - $H(X|A) = -q \lg q - (1-q) \lg (1-q)$
  - $I(A;X) = H(X) - H(X|A)$

$I(A;X)$

$$I(A; X) = -[pq + (1-p)(1-q)] \lg [pq + (1-p)(1-q)] - [(1-p)q + p(1-q)] \lg [(1-p)q + p(1-q)] + q \lg q + (1-q) \lg (1-q)$$

- Maximum when $p = 0.5$; then

  $I(A;X) = 1 + q \lg q + (1-q) \lg (1-q) = 1-H(V)$

- So, if V constant, $q = 0$, and $I(A;X) = 1$
- Also, if $q = p = 0.5$, $I(A;X) = 0$

Analyzing Capacity

- Assume a noisy channel
- Examine covert channel in MLS database that uses replication to ensure availability
  - 2-phase commit protocol ensures atomicity
  - *Coordinator* process manages global execution
  - *Participant* processes do everything else

How It Works

- Coordinator sends message to each participant asking whether to abort or commit transaction
  - If any says "abort", coordinator stops
- Coordinator gathers replies
  - If all say "commit", sends commit messages back to participants
  - If any says "abort", sends abort messages back to participants
  - Each participant that sent commit waits for reply; on receipt, acts accordingly

Exceptions

- Protocol times out, causing party to act as if transaction aborted, when:
  - Coordinator doesn’t receive reply from participant
  - Participant who sends a commit doesn’t receive reply from coordinator

Covert Channel Here

- Two types of components
  - One at *Low* security level, other at *High*
- Low component begins 2-phase commit
  - Both *High*, *Low* components must cooperate in the 2-phase commit protocol
- *High* sends information to *Low* by selectively aborting transactions
  - Can send abort messages
  - Can just not do anything

Note

- If transaction *always* succeeded except when *High* component sending information, channel not noisy
  - Capacity would be 1 bit per trial
  - But channel noisy as transactions may abort for reasons *other* than the sending of information

Analysis

- X random variable: what *High* user wants to send
  - Assume abort is 1, commit is 0
  - $p = p(X = 0)$ probability *High* sends 0
- A random variable: what *Low* receives
  - For noiseless channel $X = A$
- $n + 2$ users
  - Sender, receiver, n others
  - q probability of transaction aborting at any of these n users

Basic Probabilities

- Probabilities of receiving given sending
  - $p(A=0 \mid X=0) = (1-q)^n$
  - $p(A=1 \mid X=0) = 1-(1-q)^n$
  - $p(A=0 \mid X=1) = 0$
  - $p(A=1 \mid X=1) = 1$
- So probabilities of receiving values:
  - $p(A=0) = p(1-q)^n$
  - $p(A=1) = 1-p(1-q)^n$

More Probabilities

- Given sending, what is receiving?
  - $p(X=0 \mid A=0) = 1$
  - $p(X=1 \mid A=0) = 0$
  - $p(X=0 \mid A=1) = p[1-(1-q)^n] / [1-p(1-q)^n]$
  - $p(X=1 \mid A=1) = (1-p) / [1-p(1-q)^n]$

Entropies

- $H(X) = -p \lg p - (1-p) \lg (1-p)$
- $H(X \mid A) = -p[1-(1-q)^n] \lg p$
  $- p[1-(1-q)^n] \lg [1-(1-q)^n]$
  $+ [1-p(1-q)^n] \lg [1-p(1-q)^n]$
  $- (1-p) \lg (1-p)$
- $\mathrm{I(A;X)} = -p(1-q)^n \lg p$
  $+ p[1-(1-q)^n] \lg [1-(1-q)^n]$
  $- [1-p(1-q)^n] \lg [1-p(1-q)^n]$

Capacity

- Maximize this with respect to p (probability that *High* sends 0)
  - Notation: $m = (1-q)^n$, $M = (1-m)^{(1-m)}$
  - Maximum when $p = M / (Mm+1)$
- Capacity is:

$$I(A;X) = \frac{Mm \lg p + M(1-m) \lg (1-m) + \lg (Mm+1)}{(Mm+1)}$$

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

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

As V and A are independent,

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

$= p(A{=}a)(\Sigma_V p(B{=}b \mid V{=}v)p(V{=}v))$

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

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 $x(w)$
  - $x(w) = \mathit{length}(w)$ **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 $p(X{=}x) = 0.25+0.25 = 0.5$

$p(X{=}x \mid A{=}a) = p(X{=}(a+x) \bmod 2) = 0.5$

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

Mitigation of Covert Channels

- Problem: these work by varying use of shared resources
- One solution
  - Require processes to say what resources they need before running
  - Provide access to them in a way that no other process can access them
- Cumbersome
  - Includes running (CPU covert channel)
  - Resources stay allocated for lifetime of process

Alternate Approach

- Obscure amount of resources being used
  - Receiver cannot distinguish between what the sender is using and what is added
- How? Two ways:
  - Devote uniform resources to each process
  - Inject randomness into allocation, use of resources

Uniformity

- Variation of isolation
  - Process can't tell if second process using resource
- Example: KVM/370 covert channel via CPU usage
  - Give each VM a time slice of fixed duration
  - Do not allow VM to surrender its CPU time
    - Can no longer send 0 or 1 by modulating CPU usage

Randomness

- Make noise dominate channel
  - Does not close it, but makes it useless
- Example: MLS database
  - Probability of transaction being aborted by user other than sender, receiver approaches 1
    - $q \to 1$
  - $I(A; X) \to 0$
  - How to do this: resolve conflicts by aborting increases q, or have participants abort transactions randomly

Problem: Loss of Efficiency

- Fixed allocation, constraining use
  - Wastes resources
- Increasing probability of aborts
  - Some transactions that will normally commit now fail, requiring more retries
- Policy: is the inefficiency preferable to the covert channel?

Example

- Goal: limit covert timing channels on VAX/VMM
- "Fuzzy time" reduces accuracy of system clocks by generating random clock ticks
  - Random interrupts take any desired distribution
  - System clock updates only after each timer interrupt
  - Kernel rounds time to nearest 0.1 sec before giving it to VM
    - Means it cannot be more accurate than timing of interrupts

Example

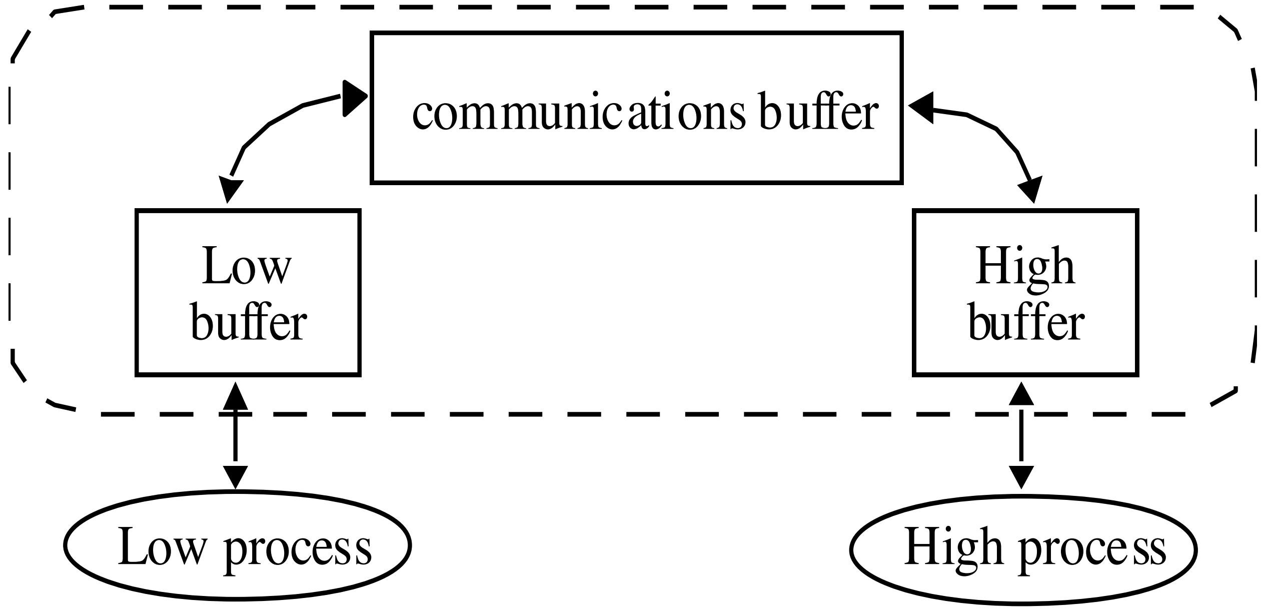
- I/O operations have random delays
- Kernel distinguishes 2 kinds of time:
  - *Event time* (when I/O event occurs)
  - *Notification time* (when VM told I/O event occurred)
    - Random delay between these prevents VM from figuring out when event actually occurred)
    - Delay can be randomly distributed as desired (in security kernel, it's 1–19ms)
  - Added enough noise to make covert timing channels hard to exploit

Improvement

- Modify scheduler to run processes in increasing order of security level
  - Now we're worried about "reads up", so …
- Countermeasures needed only when transition from *dominating* VM to *dominated* VM
  - Add random intervals between quanta for these transitions

The Pump

- Tool for controlling communications path between *High* and *Low*



Details

- Communications buffer of length *n*
  - Means it can hold up to *n* messages
- Messages numbered
- Pump ACKs each message as it is moved from *High* (*Low*) buffer to communications buffer
- If pump crashes, communications buffer preserves messages
  - Processes using pump can recover from crash

Covert Channel

- Low fills communications buffer
  - Send messages to pump until no ACK
  - If *High* wants to send 1, it accepts 1 message from pump; if *High* wants to send 0, it does not
  - If *Low* gets ACK, message moved from *Low* buffer to communications buffer ⇒ *High* sent 1
  - If *Low* doesn't get ACK, no message moved ⇒ *High* sent 0
- Meaning: if *High* can control rate at which pump passes messages to it, a covert timing channel

Performance vs. Capacity

- Assume *Low* process, pump can process messages more quickly than *High* process
- L_i random variable: time from *Low* sending message to pump to *Low* receiving ACK
- H_i random variable: average time for *High* to ACK each of last *n* messages

Case1: $E(L_i) > H_i$

- *High* can process messages more quickly than *Low* can get ACKs
- Contradicts above assumption
  - Pump must be delaying ACKs
  - *Low* waits for ACK whether or not communications buffer is full
- Covert channel closed
- Not optimal
  - Process may wait to send message even when there is room

Case 2: $E(L_i) < H_i$

- *Low* sending messages faster than *High* can remove them
- Covert channel open
- Optimal performance

Case 3: $E(L_i) = H_i$

- Pump, processes handle messages at same rate
- Covert channel open
  - Bandwidth decreased from optimal case (can’t send messages over covert channel as fast)
- Performance not optimal

Adding Noise

- Shown: adding noise to approximate case 3
  - Covert channel capacity reduced to 1/*nr* where *r* time from *Low* sending message to pump to *Low* receiving ACK when communications buffer not full
  - Conclusion: use of pump substantially reduces capacity of covert channel between *High*, *Low* processes when compared to direct connection

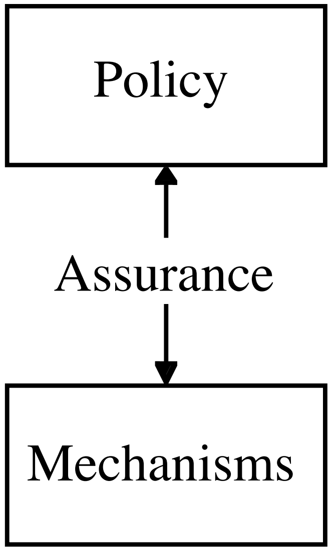
Assurance

- Trust
- Problems from lack of assurance
- Types of assurance
- Life cycle and assurance
- Waterfall life cycle model
- Other life cycle models

Trust

- *Trustworthy* entity has sufficient credible evidence leading one to believe that the system will meet a set of requirements
- *Trust* is a measure of trustworthiness relying on the evidence
- *Assurance* is confidence that an entity meets its security requirements based on evidence provided by applying assurance techniques

Relationships



Statement of requirements that explicitly defines the security expectations of the mechanism(s)

Provides justification that the mechanism meets policy through assurance evidence and approvals based on evidence

Executable entities that are designed and implemented to meet the requirements of the policy

Problem Sources

1. Requirements definitions, omissions, and mistakes
2. System design flaws
3. Hardware implementation flaws, such as wiring and chip flaws
4. Software implementation errors, program bugs, and compiler bugs
5. System use and operation errors and inadvertent mistakes
6. Willful system misuse
7. Hardware, communication, or other equipment malfunction
8. Environmental problems, natural causes, and acts of God
9. Evolution, maintenance, faulty upgrades, and decommissions