Information Flow & Covert Channels

Objectives

- Understand information flow principles
- Understand how information flows can be identified
- Understand the purpose of modeling information access
- Understand covert channels and how to prevent them

Information Flow

- Information Flow: transmission of information from one "place" to another. Absolute or probabilistic.
- How does this relate to confidentiality policy?
  - Confidentiality: What subjects *can* see what objects. So, confidentiality specifies *what* is allowed.
  - Flow: Controls what subjects *actually* see. So, information flow describes *how* policy is enforced.

Recognizing Information Flows

- Compiler-based
  - Verifies that information flows throughout a program are authorized. Determines if a program *could* violate a flow policy.
- Execution-based
  - Prevents information flows that violate policy.
- Both analyze code
- Execution-based typically requires tracking the security level of the PC as the program executes.

Compiler Mechanisms

- Declaration approach
  - x: *integer* **class** { *A,B* }
  - Specifies what security classes of information are allowed in x
- Function parameter: class = argument
- Function result: class = ∪ parameter classes
  - Unless function verified stricter
- Rules for statements
  - Assignment: LHS must be able to receive all classes in RHS
  - Conditional/iterator: then/else must be able to contain if part
- *Verify a program is secure becomes type checking!*

Examples

- Assignments:
  - x = w+y+z;
  - lub{w,y,z} ≤ x
- Compound Statements:

```
begin
  x = y+z;
  a = b+c -x
end
```

lub{y,z} ≤ x and lub{b,c,x} ≤ a

Compiler-Based Mechanisms

```
int sum (int x class{x}) {
   int out class{x, out};
   out = out + x;
}
```

What is required for this to be a secure flow?

$x \leq out$ and $out \leq out$

Compiler-Based Mechanisms

- Iterative statements - Information can flow from the *absence* of execution.

```
while f(x1, x2, …, xn) do
    S;
```

- Which direction are the flows?
  - from var's in the conditional stmt thru assignments to variables in S
- For iterative statements to be secure:

1. Statement terminates
2. S is secure
3. lub {x_1, x_2, …, x_n } ≤ glb {target of an assignment of S}

Execution Mechanisms

- Problem with compiler-based mechanisms
  - *May be too strict*
  - Valid executions not allowed
- Solution: run-time checking
- Difficulty: *implicit* flows
  - **if** x=1 **then** y:=0;
  - When x:=2, does information flow to y?
- Solution: Data mark machine
  - Tag variables
  - *Tag Program Counter*
  - Any branching statement affects PC security level
    - Affect ends when “non-branched” execution resumes

Data Mark: Example

- Statement involving only variables *x*
  - If PC ≤ *x* then *statement*
- Conditional involving *x*:
  - Push PC, PC = *lub*(PC,*x*), execute inside
  - When done with conditional statement, Pop PC
- Call: Push PC
- Return: Pop PC
- Halt
  - if *stack empty* then *halt execution*

Covert Channels

- Covert channels are found in everyday life
- Name some!

Covert Channels

- A path of communication that was not designed to be used for communication
- An information flow that is not controlled by a security mechanism
- Can occur by allowing low-level subjects to see names, results of comparisons, etc. of high-level objects
- Difficult to find, difficult to control, critical to success

Covert Channels

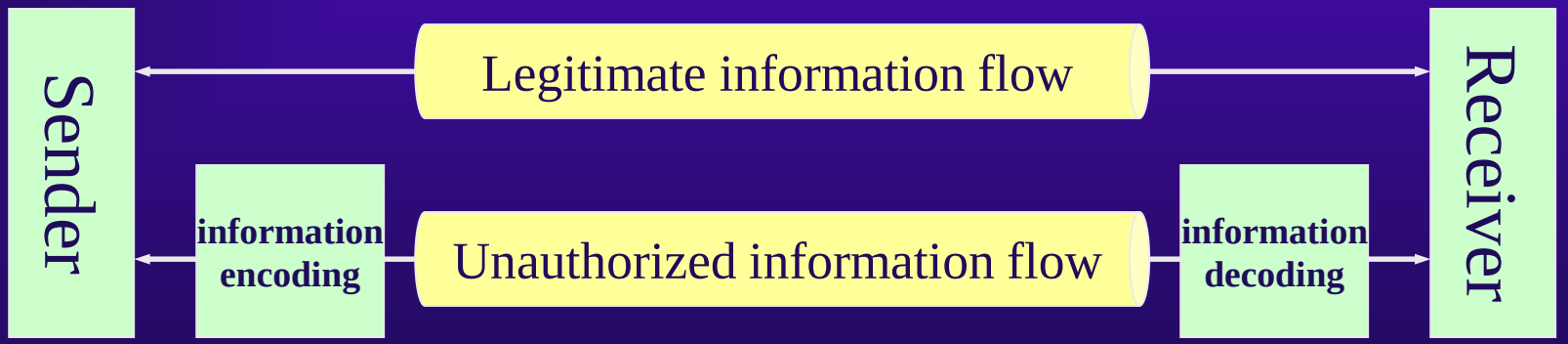
- Program that leaks confidential information *intentionally* via *secret channels*.
- Not that hard to leak a small amount of data
  - A 64 bit shared key is quite small!
- Example channels
  - Adjust the formatting of output: use the “\t” character for “1” and 8 spaces for “0”
  - Vary timing behavior based on key

Definition of convert channel

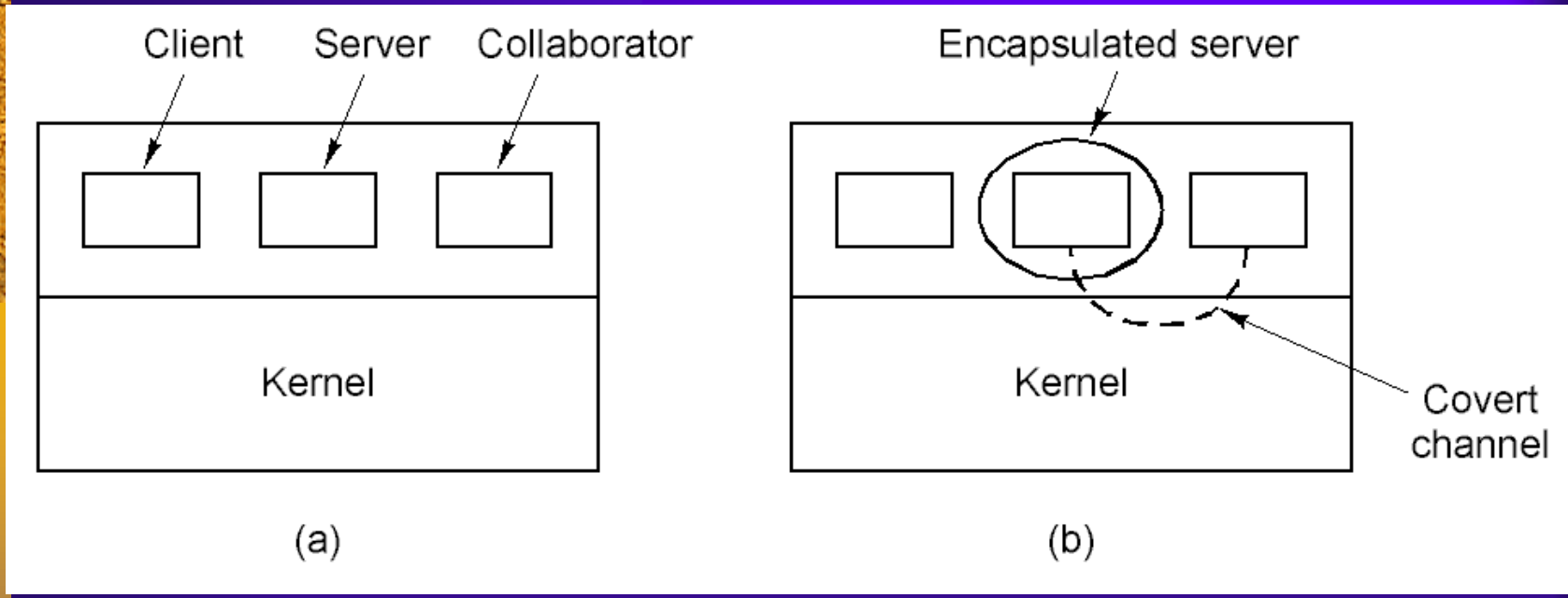
- **Definition 1** : A communication channel is covert if it is neither designed nor intended to transfer information at all
- **Definition 2** : A communication channel is covert if it is based on transmission by storage into variables that describe resource states
- **Definition 3** : Those channels that are a result of resource allocation policies and resource management implementation
- **Definition 4** : Those that use entities not normally viewed as data objects to transfer information from one subject to another
- **Definition 5** : Given a non-discretionary security policy model M and its interpretation I(M) in an operating system, any potential communication between two subjects I(S1) and I(S2) of I(M) is covert if and only if any communication between the corresponding subjects S1 and S2 of the model M is illegal in M.

Covert Channels Result From

- Transfer unauthorized information
- Do not violate access control and other security mechanisms
- Available almost anytime
- Result from following conditions
  - Design oversight during system or network implementation
  - Incorrect implementation or operation of the access control mechanism
  - Existence of a shared resource between the sender and the receiver
  - The ability to implant and hide a Trojan horse



Covert Channels

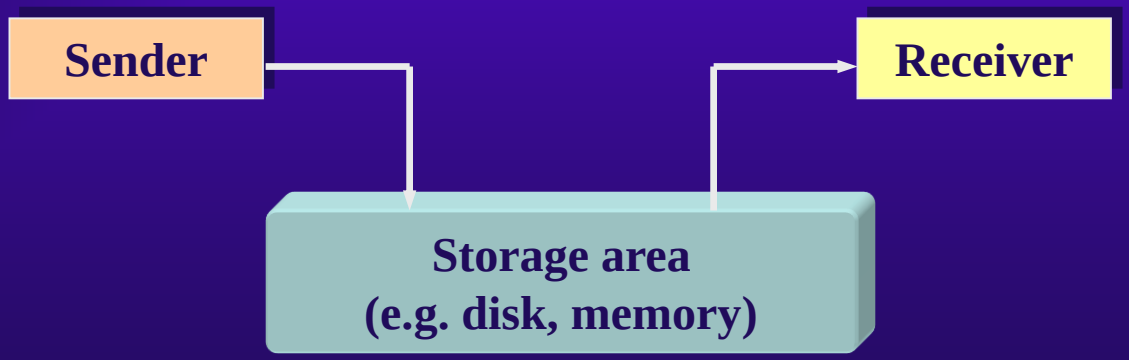


client, server and collaborator processes

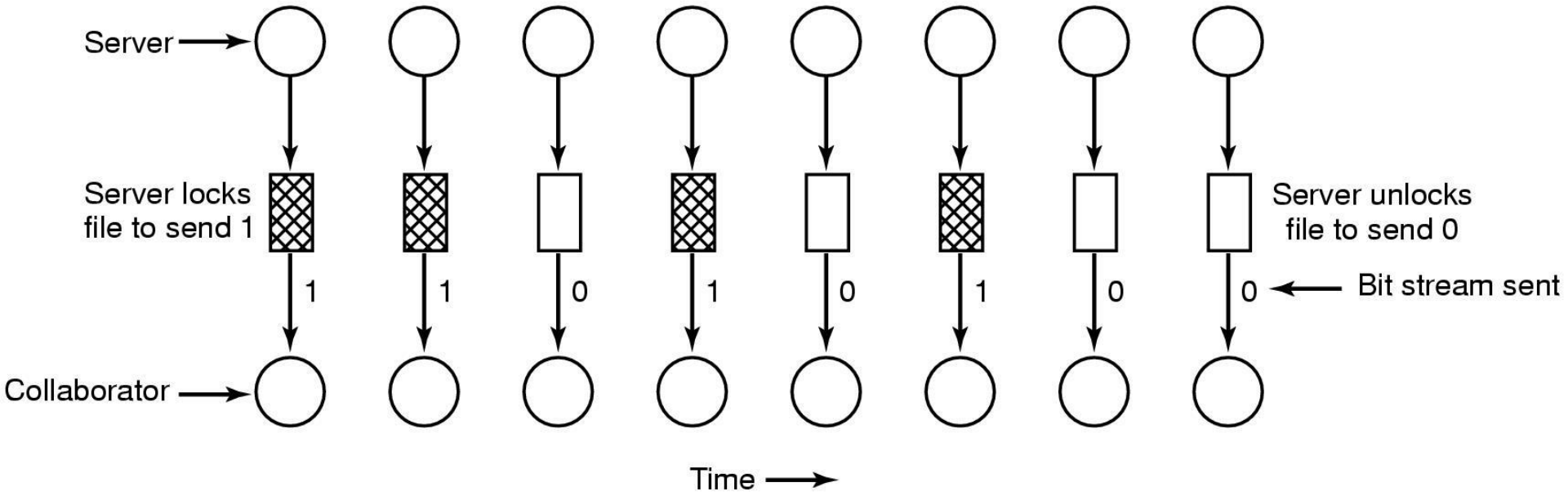
encapsulated server can still leak to collaborator via covert channels

Covert storage channel

- Involves the direct or indirect writing to storage location by one process and direct or indirect reading of the storage by another process.
- Example storage mechanisms
  - Disk space
  - Print spacing
  - File naming



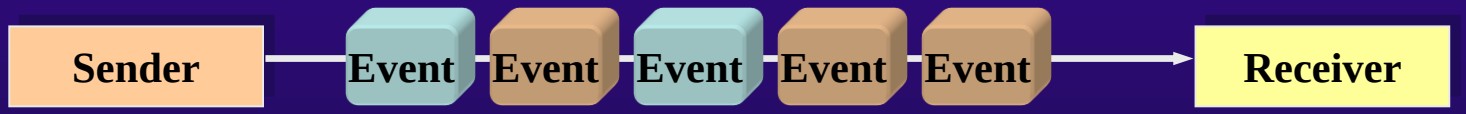
Covert Channels



A covert channel using file locking

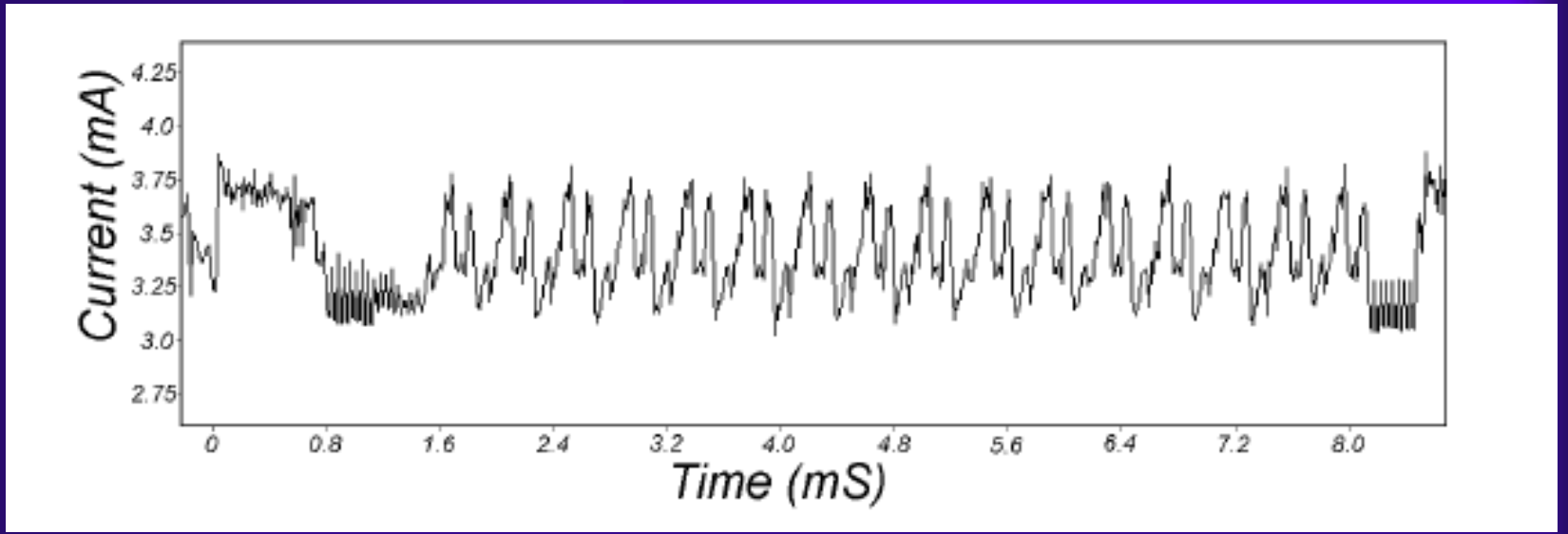
Covert timing channel

- Covert timing channel
  - Signals information to another by modulating its own use of system resource is such way that this manipulation affects the real response time observed by second process.
- Sequence of events
  - CPU utilization
  - Resource availability



Differential Power Analysis

- Read the value of a DES password off of a smartcard by watching power consumption!



- This figure shows simple power analysis of DES encryption. The 16 rounds are clearly visible.

Covert channel identification: Shared resource matrix (SRM) method

Four steps

1. Analyze all Trusted Computing Base primitive operations
2. Build a shared resource matrix
3. Perform a transitive closure on the entries of the SRM
4. Analyze each matrix column containing row entries with either ‘R’ or ‘M’

L : legal channel exists

N : one cannot gain useful information from channel

S : sending and receiving processes are the same

P : potential channel exists

primitives	shared global variables		
	mode	mode	file table
access	R		
chmod	RM		
write	RM	M	
link	RM		
...	...	...	...

R : Read

M : Modify

Covert channel identification:
Shared resource matrix (SRM) method

Conditions

1. Two or more process must have access to a common resource
2. At least One process must be able to alter the condition of the resource
3. The other process must be able to sense if the resource has been altered
4. There must be a mechanism for initiating and sequencing communications over this channel

Advantages

Can be applied to both formal and informal specifications
Does not differentiate between storage and timing channels
Does not require that security levels be assigned to internal TCP variables

Drawbacks

Individual TCB primitives cannot be proven secure in isolation
May identify potential channels that could otherwise be eliminated automatically by information flow analysis

Covert Channel Mitigation

- Can covert channels be eliminated?
  - Eliminate shared resource?
- *Severely* limit flexibility in using resource
  - Otherwise we get the halting problem
  - Example: Assign fixed time for use of resource
    - *Closes timing channel*
- Not always realistic
  - *Do we really need to close every channel?*

Covert Channel Analysis

- Solution: Accept covert channel
  - But analyze the capacity
    - *How many bits/second can be "leaked"*
- Allows cost/benefit tradeoff
  - Risk exists
  - Limits known
- Example: Assume data time-critical
  - Ship location classified until next commercial satellite flies overhead
  - Can covert channel transmit location before this?

Conclusion

- Have you ever used or even seen a language with security types?
- Why not?

- Under what circumstances would you worry about covert channels?