

CS590U

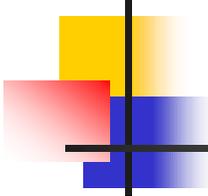
# **Access Control: Theory and Practice**

Lecture 6 (January 26)

Information Flow, Confinement &  
Covert Channels

# A Lattice Model of Secure Information Flow

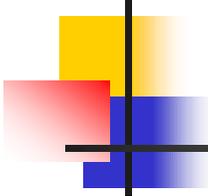
Dorothy Denning  
CACM 1976



# Information Flow Model

---

- An information flow model is defined by  $FM = \langle SC, \oplus, \rightarrow \rangle$ 
  - where  $SC$  is a finite set of security classes
  - $\oplus$  is the class-combining operator
    - is an associative and commutative binary operator
    - $A \oplus B$  denotes the security class of information that includes information both of  $a$  and of  $b$
  - $\rightarrow$  is a binary operation that specifies from which class information can flow into which class

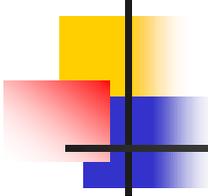


# When Information Flows

---

## Examples

- $y := x$
- $z := x; \quad y := z$
- $z := x + y;$
- $z := x \oplus y;$
- if  $x = 1$  then  $y := 1$
- if  $(x = 1)$  and  $(y = 1)$  then  $z := 1$
- if  $x = 1$  then if  $y = 1$  then  $z := 1$



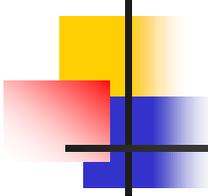
# Why Lattice?

---

- If the following holds, then  $\langle SC, \rightarrow \rangle$  is a lattice
  - $\langle SC, \rightarrow \rangle$  is a poset
  - SC is finite
  - SC has a lower bound L such that  $L \rightarrow A$  for all  $A \in SC$
  - $\oplus$  is the least upper bound operator

# A Note on the Confinement Problem

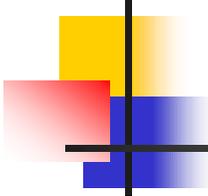
Butler Lampson  
CACM October 1973



# The Confinement Problem

---

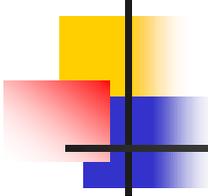
- Confine a program's execution so that it cannot transmit information to any other program except its caller.
- Motivation:
  - a customer uses a service program and wants to ensure that the inputs are not leaked by the service program



# Ways to leak information

---

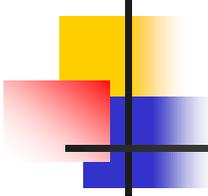
0. The service has memory and can be called by its owner
1. The service writes to a permanent file that can be read by its owner
2. The service writes to a temporary file that can be read by its owner
3. The service sends a message to the owner's process using interprocess communication



# Ways to leak information

---

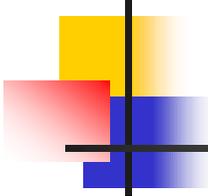
4. Information may be encoded in the bill rendered for the service, or payment for resources used by the service program
5. Using file lock as a shared boolean variable
6. By varying its ratio of computing to input/output or its paging rate, the service can transmit information to a concurrently running process



# Confinement rules (from the paper)

---

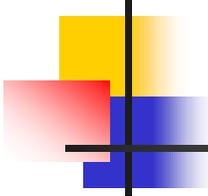
- A confined program must be memoryless, i.e., it must not be able to preserve information within itself from one call to another
- **Total isolation:** A confined program shall make no calls on any other program
  - sufficient to ensure confinement
  - quite impractical as even system calls may be dangerous and thus need to be forbidden



## Less Restrictive Case

---

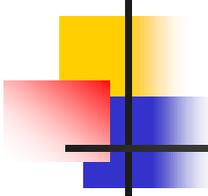
- Trusted programs: programs trusted not to leak data or help any confined program that calls them leak data
- **Transitivity:** if a confined program calls another program which is not trusted, then the called program must also be confined.
- It is difficult to write a trustworthy operating system, as some information path are subtle and obscure.



# Writing a Trustworthy Program

---

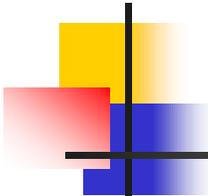
- A trustworthy program must guard against any possible leakage of data.
- In an operating system, the number of possible channels is large, but finite.
- It is necessary to enumerate all of them and to block each one.



# Three Categories of Channels

---

- Storage: write/read files
- Legitimate: bill for the service program
- Covert: CPU/memory usage
- The following simple principle is sufficient to block all legitimate & covert channels:
  - Masking: A program is confined must allow its caller to determine all its inputs into legitimate and covert channels. We say that the channels are masked by the caller.



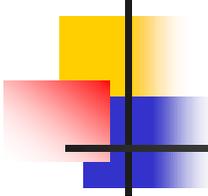
# On Blocking Covert Channels

---

- Enforcement: The supervisor must ensure that a confined program's input to covert channels conforms to the caller's specifications.
  - this may require slowing the program down, generating spurious disk references, or whatever, but it is conceptually straightforward
  - The cost of enforcement may be high. A cheaper alternative is to bound the capacity of the covert channels.

# A Comment on the Confinement Problem

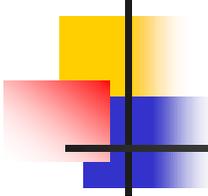
Steven B. Lipner  
SOSP 1975



# Key observations

---

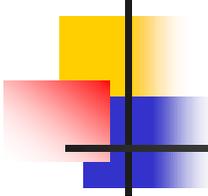
- The confinement problem is similar in objective to MAC security
  - the common objective is to stop information flow
- Supposedly, \*-property solves confinement problem for storage channels
  - Identifying all objects is difficult, but can be done



# Closing “Covert Channels” is the most difficult

---

- To close “timing channels”
  - each subject must be constrained to see a virtual time depending only on its activities
  - seems to solve the covert channel problem
  - unclear whether this is possible, because each user also has sense of time outside the system

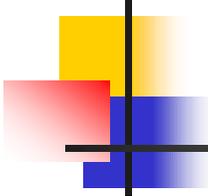


## Conclusion of this paper

---

- While the storage and legitimate channels of Lampson can be closed with a minimal impact on system efficiency, closing the covert channel seems to impose a direct and unreasonable performance penalty.
- Closing the covert channels seems at a minimum very difficult, and may very well be impossible in a system where physical resources are shared.

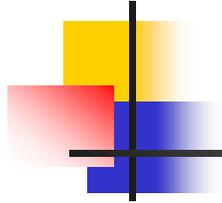
# Other Discussions on Covert Channels



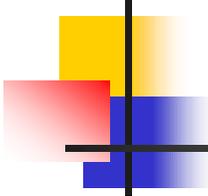
# Covert Channels in MLS

---

- Covert storage channels: In BLP, if a file is considered to be an object, a low subject may be able to see file names of high, which can encode information.
  - low users can write high files; thus it reasonable to know names of high files
- Covert timing channels



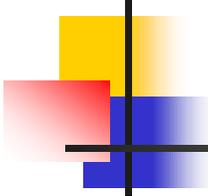
- Covert channels are often noisy
- However, information theory and coding theory can be used to encode and decode information through noisy channels
- Military requires cryptographic components be implemented in hardware
  - to avoid trojan horse leaking keys through covert channels



# The Resource Matrix Approach

---

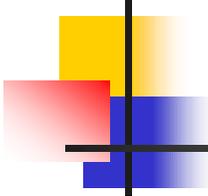
- An approach to systematically identify covert channels
- Kemmerer: “Shared Resource Matrix Methodology: An Approach to Identifying Storage and Timing Channels”, ACM TOCS.
  - Conference version in Oakland 1982.



# Intuition

---

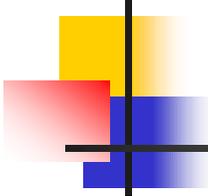
- Finding all resources that are shared between high and low users
  - covert channels rely on sharing of some resource that can be used in an unexpected way to transfer information



# The Matrix

---

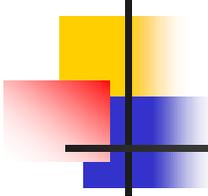
- Each system resource has a row
- Each lowest-level system operation that can be performed on resources is a column
- Each cell contains a subset of  $\{R, M\}$ 
  - R means referencing the resource
  - M means modifying the resource



# Criteria for Identifying Covert Channels

---

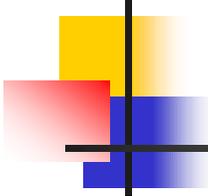
- E.g., a storage channel exists when a high user can change an attribute of a shared resource and a low user can detect the change
- E.g., the criteria for a timing channel includes a shared common attribute, a shared time reference, and a means for modulating changes to this attribute.



# Polyinstantiation

---

- Suppose that a High user creates a file named agents, when a Low user tries to create the same file, it would fail, thus leaking information
  - may be solved using naming conventions
- The problem gets more difficult in databases:  
Suppose that a High user allocate classified cargo to a ship, then a Low user may think the ship is empty and tries to allocate other cargos
  - one approach is to use a cover story



# End of Lecture 6

---

- Next lecture
  - Integrity, Biba, Clark-Wilson