Protection in the Hydra Operating System

Ellis Cohen and David Jefferson
Carnegie-Mellon University

Presented By : Farhan Saleem Khan

Overview

- Introduction
- Protection Philosophy adopted by Hydra
- Objects, Capabilities and LNE's
- Protection Problems
  - Mutual Suspicion
  - Modification
  - Limiting Propagation of Capabilities
- Conclusion

Introduction – Protection Problem

- Sharing of information : possibility of malicious and accidental disclosure of information.
- Solution : Restrict the behavior of possible computation.

- **Definition**: A protection problem is simply a description of some class of restricted behavior.

- How: System provides *mechanisms* for restricting the access on certain information.

Hydra

- Protection must be integral part of operating system.

- **Hydra**
- Set of protection mechanisms
- Flexible enough to support wide range of security policies.
- Capability based mechanism.

Protection Philosophy

- Not specific security policy implemented.
- Provides a set of mechanisms with which a large set of policies can be constructed.
- Five philosophical principles.

1) Information can be divided into distinct objects for purpose of protection.

- Group information together into a uniform data structure called as ***object***.
- Provides protection at object level.
- Example:
  - Group information in the form of **file objects** and set protections at individual file-level.

Protection Philosophy (cont ...)

2) Objects are distinguished by type

- Each object is of a particular type, which remains constant.
- Provides certain objects and their respective procedure.
  - Procedure, Process, Semaphore etc ...
- Also provides mechanism for creating new objects and their respective operations that can manipulate them.

3) Access to objects is controlled by capabilities

- A ***capability*** contains a large number of access rights which determine how the object named by the capability can be accessed.
- May be passed from one user to another.
- Objects do not have single owner.
- All holders of capabilities for an object share control of it in proportion to their rights.

Protection Philosophy (cont ...)

4) Each program should execute with the smallest set of access rights necessary

- Protection domain is that set of capabilities which may be exercised by an executing procedure.
- It changes with each procedure call.
- Procedures have access to "own" objects, inaccessible to users with only execution right.
- Each call to a procedure executes in a new environment determined by
  - Procedures "owns"
  - Capabilities passed as arguments by the caller.

5) Knowledge about the representation and implementation of operations for each type of object should be hidden in module called subsystems.

- Each type of objects and their associated procedures comprises a ***subsystem***.
- Users can't access objects directly.
- Objects can be manipulated through procedures of that subsystem.

Objects

- A data structure representing an instance of a resource, either virtual or physical.
- Three-tuple : < unique-name , type , representation >
  - Unique-name distinguishes it from all other objects.
  - Type defines the nature of the resource
    - DEVICE, DIRECTORY, PROCESS, SEMAPHORE etc ...
  - Representation = ???

Capabilities

- Contains
  - name of the particular object and
  - set of access rights which determine how the object can be accessed.
- ***C-list***, a linearly numbered list of capabilities (associated with every executing program).
- The representations of capabilities and rights are manipulated only by the hydra Operating System ➔ impossible to forge a capability or gain access to an object without having capability for it.

Rights

- A capability contains names of an object and set of rights to that object.
- Rights list is implemented as a bit vector of 24.
  - 16 generic rights
  - 8 auxiliary rights
- Each bit : presence or absence of a right (*operation*).
- Not all generic rights applicable for all Types.
  - CALLRTS (call-right): makes sense for objects of type *Procedure*, but not for other objects ...
  - WRITERTS (write-rights): makes sense for *File* type, not for *Process*, *Semaphore* etc ...

Local Name Space (LNE)

- C-list not an attribute of executing program alone.
- Any object may contain a C-list
- Two important effects.
  - Executing program may be represented as a TYPE of OBJECT.
    - This type is called ***Local Name Space***.
  - New objects can be defined in terms of existing objects.
    - For Example: File Directory which contains both a list of files and a semaphore which provides mutual exclusion on the file directory.
- An object also contains a ***Data-part***, a block of storage holding relevant information.
  - For Example: The data-part might be used to hold the string names of the files.
- C-list + Data-part ➔ representation of object

Generic Operations

- All objects have common underlying structure
  - C-list and Data-part
- Hydra provides a type-independent, generic operations for manipulations of these objects.
- Kernel calls or k-calls (trap to the kernel).
- Getdata
  - Provides access to the data-part of object.
  - Arguments:
    - Path to a capability for the object whose data-part is to be read.
    - Parameters to specify what part to be read (offset and length)
    - Destination address in caller address space
- Putdata
  - To write into data-part
- Addata
  - Append data onto the end of the data-part

Manipulation of data-part

Generic Operations (cont ...)

- Similar operations for manipulating C-list.
- Load
  - Reading a C-list entry
- Store
  - Writing on a C-list entry
- Append
  - Append a capability to a C-list
- Delete
  - Removes a capability from the C-list
  - Capability is replaced with capability of a special type NULL.
- Copy
  - For copying data-part and C-list of an object into another object.

Manipulation of C-list

Generic Operations (cont ...)

- Generic operations implemented indivisibly.
  - Two processes can't operate on the same object simultaneously.
  - This is also insured in multiprocessor environment.
- Mutual exclusion of composite operations requires some sort of synchronization, such as semaphores.

Procedures

- An object serving as an abstraction of ordinary procedures
- May take capabilities as arguments.
- May return capability as result.

Call (cProc , return-Slot , p_1 , $mask_1$, … , p_n , $mask_n$)

  - cProc : capability of the **procedure** to be called
  - return-Slot : Slot in the C-list where cProc can return a capability
  - p_1 : capability to argument 1
  - $Mask_1$: mask to restrict the rights in the capability passed

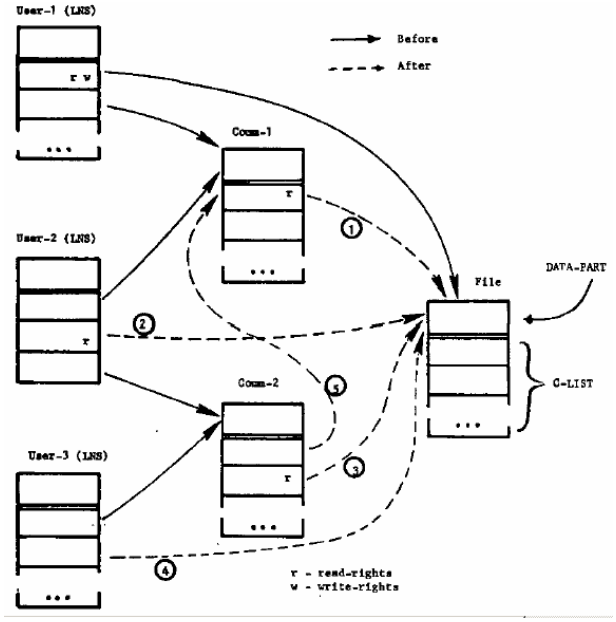
Sharing

- Object / capability structure ➔ permits sharing

- **Sharing of information**
- User-1 and User-2 both have capability for some object Comm-1
  - User-1 stores some information in Comm-1 data-part
  - User-2 retrieves it

- **Sharing of rights**
- User-1 and User-2 both have capability for some object Comm-1
- User-1 has rights to read and write some specific file and wishes to grant User-2 read access only.
  - User-1 stores file capability in Comm-1 allowing only read access
  - User-2 accesses the file using capability stored Comm-1

Sharing (cont ...)



Protection Problems

- Mutual Suspicion
- Modification
- Limiting Propagation of Capabilities

Mutual Suspicion

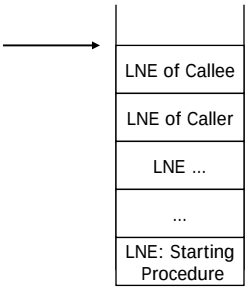
- Caller
  - One user (caller) calls a utility procedure (callee) belonging to another user or kernel.
  - Risk of malicious access to caller data by the callee procedure. (delete files, etc ...)
- Callee
  - Utility procedure or kernel procedure (callee) is called by many users (callers).
  - It manipulates certain private files or data structures.
  - Callee needs some guarantee that, callers never access these sensitive data structures.
- These two problems are together called ***Mutual Suspicion***.

- ***Problem Definition***
  - *"The Caller of a hydra procedure needs a guarantee that the callee is not granted access to any of his objects, excepts those for which capabilities are explicitly passed as parameters.*
  - *The callee needs a guarantee that the caller can't gain access to private data of callee, except when the callee explicitly allows it".*

Mutual Suspicion (cont ...)

- Protection for Callee
  - Every procedure can only operate objects whose capabilities are present in it C-list.
  - Procedure may have "*own*" capabilities. So keep sensitive and private data in own capabilities and doesn't return it back to caller.

- Protection for Caller
  - A procedure can't access capabilities in LNEes deeper in the process stack, the only capabilities accessed by callee are its "own" or those passed by caller explicitly.

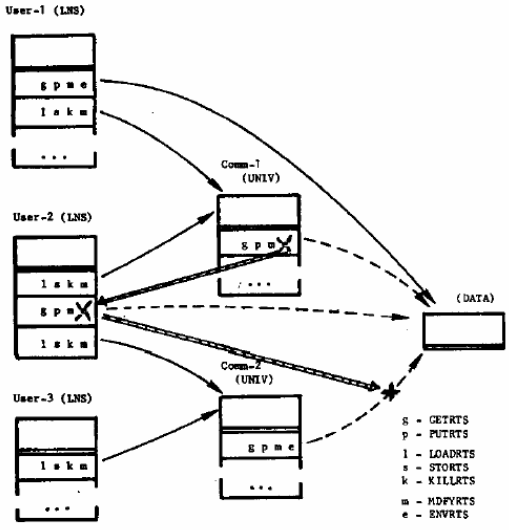


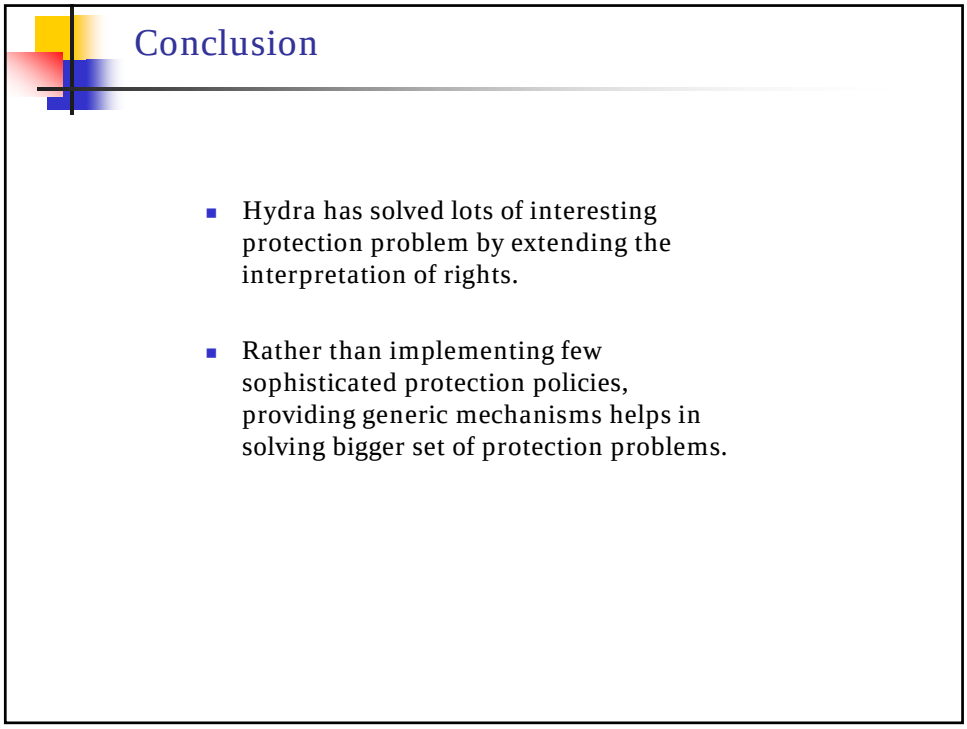
Modification

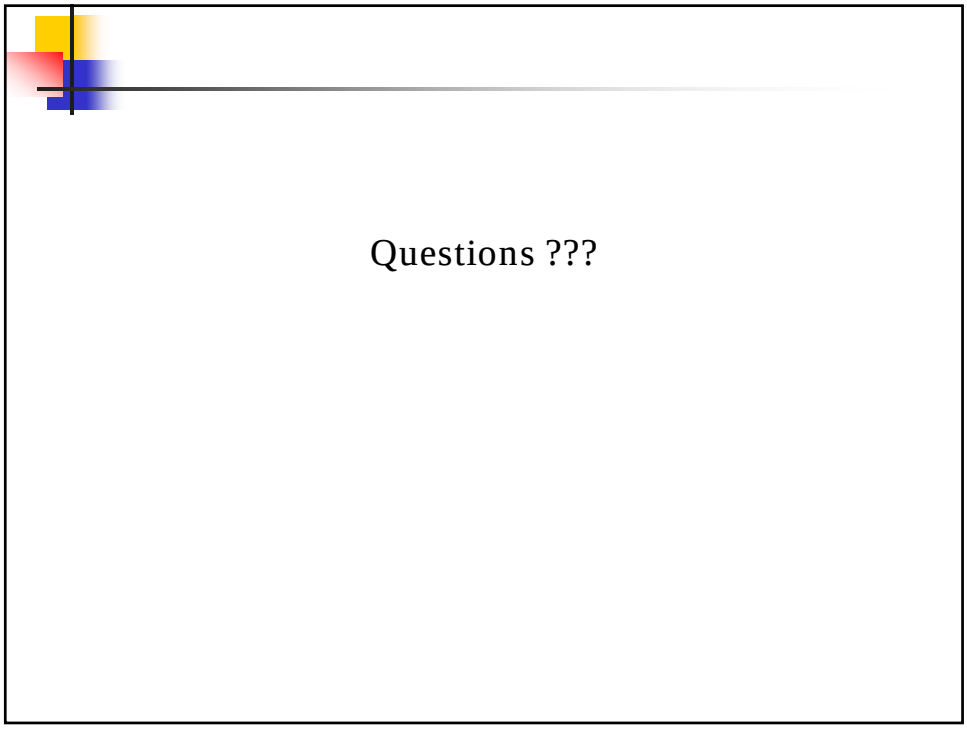
- *"User often want guarantee that an object passed as an argument to a procedure will not be modified as a result of the call."*
- Hydra solves this problem by using MDFYRTS.
  - Each Hydra k-call that modifies an object requires a capability but MDFYRTS as well.
  - Thus to store a capability in an object, one must have a capability for the object with both STORTS and MDFYRTS.
  - To solve modification problem, hydra enforces that MDFYRTS can never be gained through amplification.
  - So user passes a capability to callee procedure restricting MDFYRTS, there is no way that the procedure can modify the data-file.

Limited Propagation of Capabilities

- *"A user wishes to allow another user to access an object but wants to guarantee that the other user can't share access with a third user."*
- *Hydra solves this problem by capability right ENVRTS.*
  - *A capability may only be stored in an object if the capability contains ENVRTS.*




Conclusion
Hydra has solved lots of interesting protection problem by extending the interpretation of rights.
Rather than implementing few sophisticated protection policies, providing generic mechanisms helps in solving bigger set of protection problems.


Questions ???