

CS 518

Hydra: Non-hierarchical Protection via a User Extensible Type System

Randy Wang
Fall 2002
Princeton University

Background

- Experimental machine and experimental OS
- Two key concepts: (micro-) kernel and capability
- Micro-kernel
 - Minimal kernel in supervisor mode
 - Many OS services at user level
- Push these ideas to the very extreme

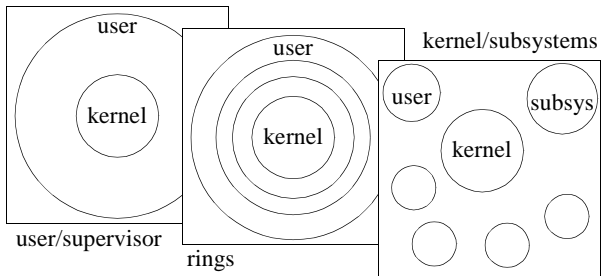
Hydra Key Ideas

- Non-hierarchical protection
- User-extensible type system
- Capability-based protection
- “Sealing/unsealing” for crossing protection boundaries

Outline

- ~~Introduction~~
- **Detailed mechanism**
- An example
- Postscript

Non-hierarchical System



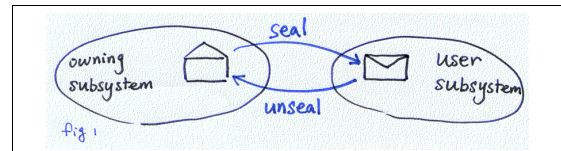
- Hierarchical systems
 - Inner rings have strictly more power
 - Doesn't scale to many components
- Non-hierarchical systems
 - Subsystem: **protected** collection of code and data
 - No one subsystem is intrinsically more powerful than the other

CS518

4

Randy Wang

Non-hierarchical Protection



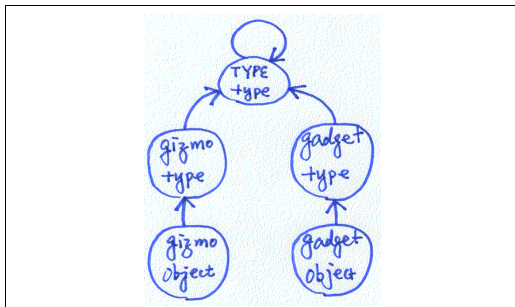
- The “owning” subsystem
 - creates,
 - “seals”, and
 - hands out an object
- “Other” subsystems
 - can't manipulate the object
 - can only hand it back to the owning system
- The “owning” subsystem
 - receives,
 - unseals, and
 - manipulates the object

CS518

5

Randy Wang

Extensible Type System



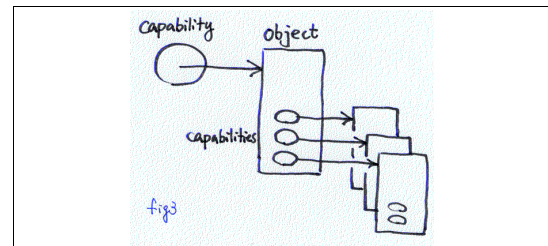
- Objects
 - name
 - type, and
 - representation
- Three-level type hierarchy
 - objects
 - type objects
 - the TYPE type object

CS518

6

Randy Wang

Capabilities and Instance Hierarchy



- Capability
 - object name
 - privileges
 - no other means of referencing objects
- Instance hierarchy
 - Embed capabilities as you would pointers
 - Can have cycles (so need garbage collection)

CS518

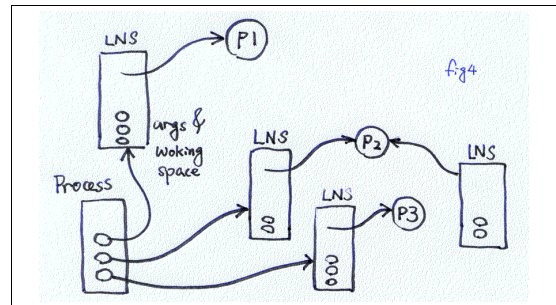
7

Randy Wang

Built-in Basic Types

- Page
- Procedure
 - Static info associated with procedure
- Local name space (LNS)
 - activation record
- Process
 - stack of LNS

Built-in Type Illustration



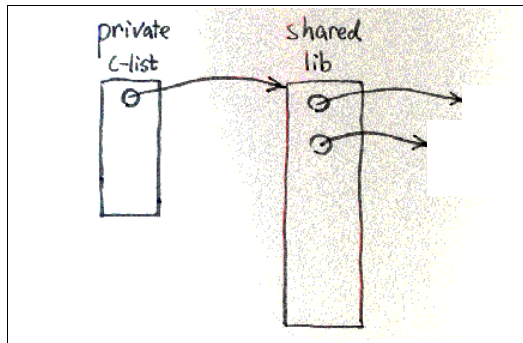
Rights Amplification

- The “Fifth Element”: templates
 - a tuple: (type, old privs, new privs)
 - one template for each argument in a procedure
- Protection boundary crossing
 - the kernel checks each argument
 - if type matches, and
 - old privs are sufficient
 - amplify rights to allow new privs
- Template creation
 - from the type object
 - creating template is simply another privilege that needs to be carefully controlled

Outline

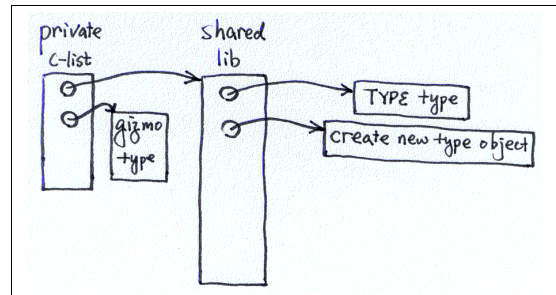
- Introduction
- Detailed mechanism
- **An example**
- Postscript

An Example (Part 1: log in)



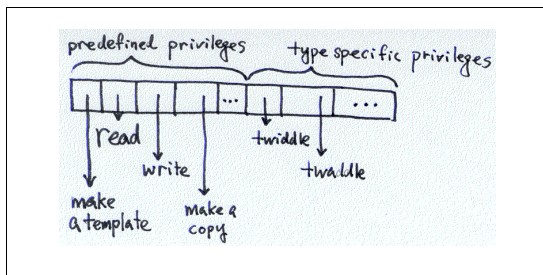
- c-list and the shared library

An Example (Part 2: create gizmo type)

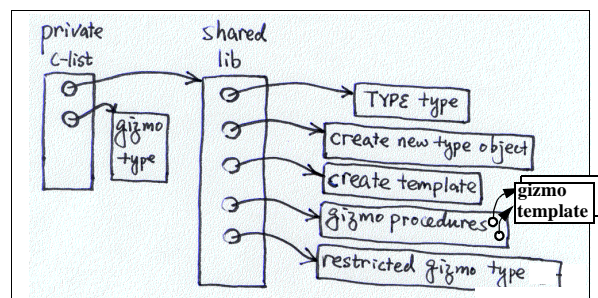


- create new type using TYPE type object
- store capability in private c-list

Gizmo Privileges



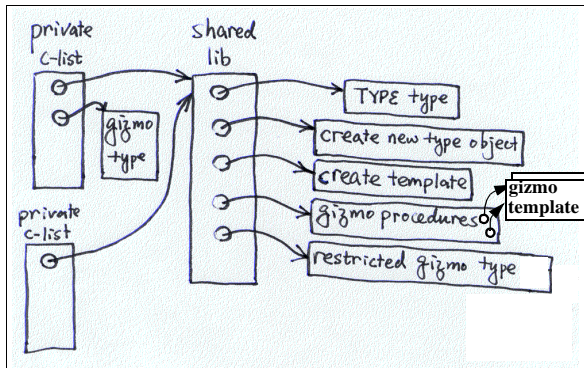
An Example (Part 3: package and publish gizmo)



- create gizmo templates
- package and publish procedures and template
- publish restricted copy of gizmo type

An Example

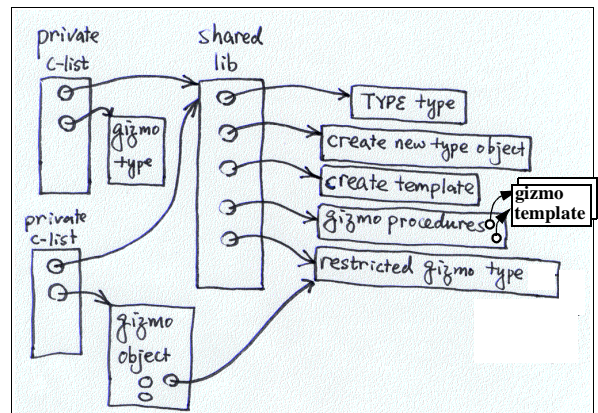
(Part 4: another user logs in)



- capability for shared library

An Example

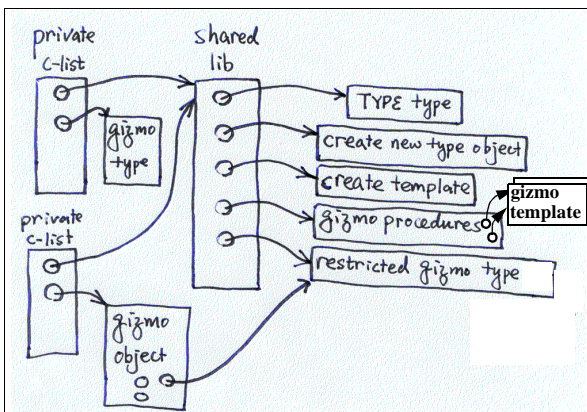
(Part 5: creates gizmo object)



- invokes create gizmo procedure

An Example

(Part 6: invokes twiddle procedure)



- rights amplification: template
 - type = gizmo
 - old priv = twiddle
 - new priv = read/write

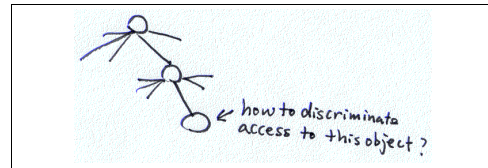
Outline

- Introduction
- Detailed mechanism
- An example
- **Postscript**

Hydra Key Ideas

- Non-hierarchical protection
- User-extensible type system
- Capability-based protection
- “Sealing/unsealing” for crossing protection boundaries

Criticisms



- Overhead of Hydra procedure calls
 - like a context switch
- Overhead of dealing with persistent objects
 - disk operations
 - garbage collection
- The “kernel” that wasn’t
 - goal: 4 KB
 - reality: 1/3 MB
- Sharing using capability can be awkward
 - hard to discriminate access
- Applications?

Postscript

- A “glorious failure”
- Persistent object stores keep re-inventing this idea