Process-specific Permissions

• Design the permissions of a process specific to its use

• How do we change the permissions of a process in an ACL system?
Confused Deputy Problem

• Imagine a multi-client server
  – Each client has a different set of objects that they can access

• In an ACL system, the server always has access to all the objects
  – What happens if a client tricks the server into accessing into another client’s objects?
  – Shouldn’t the server only have access to that client’s objects for its requests?
Capabilities

• A capability is the tuple (object, rights)
• A capability system implements access control by checking if the process has an appropriate capability
  – Simple, right?
  – This is a little like a ticket in the Kerberos system

• Q: Does this eliminate the need for authentication?
Capabilities

• A: Well, yes and no …

• Capabilities remove the overhead of managing per object rights, but add the overhead of managing capabilities

• Moreover, to get any real security, they have to be unforgeable
  – Hardware tags (to protect capabilities)
  – Protected address space/registers
  – Language based techniques
    • Enforce access restrictions on caps.
  – Cryptography
    • Make them unforgeable
Real OS Capabilities

- The OS kernel manages capabilities in the process table, out of reach of the process
- Capabilities added by user requests (that comply with policy)
User space capability?

• Well, what are the requirements?
  – Authenticity/integrity - do not want malicious process to forge capabilities

• Start with the data itself: [object, rights]
  – Object is typically encoded with identifier, or by some other tag (capabilities are sometimes known as tags)
  – Rights are often fixed (read, modify, write, execute, etc.)

• Now, do what you with any other data (assume the kernel has a secret key k)

\[ E(k, [O_i, r_1, r_2, \ldots r_n]) \]

• What’s wrong with this construction (I got it from the website of one of the experts in the area)?
The right construction

- Encryption does not provide authenticity/integrity, it provides confidentiality

\[ [O_i, r_1, r_2, \ldots r_n], \text{HMAC}(k, [O_i, r_1, r_2, \ldots r_n]) \]

- So how would you attack the preceding construction?
A (fictional) Capability Example

- We use the “ls -lt” command to view the contents of our home directory in a OS implementing capabilities:
  - Initially, our shell process has RWX capabilities for our home directory, and RX capabilities for all the directories to the root.
  - The “ls -lt” command is exec()ed, and the shell delegates the directory permissions by giving “ls” the capabilities
    - Note that the capabilities are _not_ tied to any subject
  - The “ls -lt” process exercises the rights to read the directories structure all the way down to the local
  - Of course, the “ls -lt” process now need to obtain read rights to the files (to get their specific meta-information), and obtains them by appealing to the security manager (in kernel) -- the request fulfills the policy, and they are added and exercised
  - The “ls -lt” uses access rights given to the terminal to write output
- Note: there are many ways that the policy can be implemented, rights handed off, etc. We will talk about a couple in the following discussions.
Procedure-Level Protection Domains

• HYDRA
  – Each procedure defines a new protection domain

• Procedure
  – Code
  – Data
  – Capabilities to other objects
    • Caller-independent
    • Caller-dependent templates

• Local Name Space
  – Capabilities are bound here
  – Record of a procedure invocation (procedure instance)

• Process
  – Stack of LNSs
How HYDRA works

• Q: Which object defines the protection domain?
Implications of Fine-Grained Protection

• Programmer
  – Must define templates for procedure
  – Connect the procedure rights together

• Performance Impact

• Q: Do we need to manage rights at this level?
Linden’s Capability View

• Achieve flexible, effective security by
  – Small protection domains
  – Extensible set of types

• Implies a capability system
  – Small protection domains with least privilege permissions
  – Extensible types enable composition of systems reliably
  – Capabilities can be passed among protection domains and into new subsystems

• Protected Procedures
  – Like HYDRA
  – Change domain with each procedure invocation
  – New procedure is a new instance

• Protection Domain switch time is key (high in modern processors)
Correctness Claim

• “It is far more difficult to build a 50,000 line program than 1,000 programs that are each 50 lines long.”
  – What is your opinion of this?
  – Is it just the procedure development that is important?

• Two problems
  – Decomposition results in inefficiencies
  – Interactions between procedures are not captured
Flexibility vs. Security

• Small protection domains are desirable because:
  – Enables solving finer-grained problems
  – Less rigid protection
  – Independent accounting
  – Reliable and redundant security controls
  – Individual controls are easier to understand

• Top-down vs. bottom-up; Fine vs coarse-grained
Secure Capability Systems

• SCAP
  – Karger’s extension of the Cambridge CAP system

• EROS
  – Shapiro’s reimplementation of the KeyKOS system
Capabilities and the \(*\)-Property

• Capabilities and Lattice Models Don’t Mix

• Suppose A is higher secrecy than B
  – A can read B’s capabilities

• Q: Can a Trojan horse running as A write to Obj?
SCAP *-security

• Mediate requests to load capabilities
  – Must be loaded into a capability cache before use

• Enforce MLS requirements on capability load
  – If \textit{subject label} dominates capability’s \textit{object label}, then
  – Change the capability to \textit{read-only}

• Expensive to test for MLS on every load

• For \textit{general confinement} test against confinement property for every load (uses ACLs!)
EROS *-security

• Define *weak* capabilities
  – If a weak capability is used to fetch a capability (transitively), then the fetched capability becomes read-only and weak

• Assign weak capabilities to higher-secrecy subjects for accessing a lower-secrecy write capability
  – becomes read-only and weak

• No need to test against a policy at runtime
  – Faster performance is possible

• For *general confinement* use an confined processes or authorized capability sets
  – Not clear these really worked for general confinement
Capability Management

- How’d you get those capabilities?
  - Stored with program, user
  - Compare with getting permissions by a process label

- How do I get them back?
  - Once granted, nearly impossible to revoke
EROS Revocation

- Defined by Redell
  - Use a layer of indirection
- Revoker capabilities
  - If you may revoke, create a revoker
  - The grant capabilities to the revoker
  - When you delete the revoker, all descendants become invalid
SCAP Revocation

- Chain the capabilities
  - "revocation by chaining"

- All capabilities to an object are stored in a ring
  - Can then revoke one
  - Motivate reassessment of all others
  - How do I know that I am revoking a particular capability?

- Compare with using revoker capabilities
  - the memory/performance cost
  - the flexibility of revocation
Result

• Generally, the security problems with capability systems can be solved

• So, why aren’t cap systems more broadly used?
  – Capability management is difficult
    • How do I know what rights to give out in the first place?
  – Defining and testing confinement is expensive or limiting
    • Test every grant is expensive (supposed to be lots)
    • Predefine a safe domain is limiting and counterintuitive

• Setup per process is key
  – For ACLs it is setup per object -- may be less volatility