

# Defining the Basis for Trust in a CORBA Security Implementation

Matt Stillerman  
Odyssey Research Associates, Inc.  
[matt@oracorp.com](mailto:matt@oracorp.com)

# Overview

- *About Security Arguments*
- *About CIDER*
- *Architecture*
- *Security Policy*
- *Top Level Argument*
- *Process Registration*
- *Client Identity: Argument Details*

# Structure of a Security Argument

- State the security policy
- Argue that the enforcement mechanism is
  - correct
  - tamper-proof
  - not bypassable

# Divide and Conquer

- Which parts of the system must be trusted?
- What is the critical function of each part?
- Argue that this is sufficient - if all parts do what they are supposed to do, then the security goals are met.
- Argue that each part is individually trustworthy.

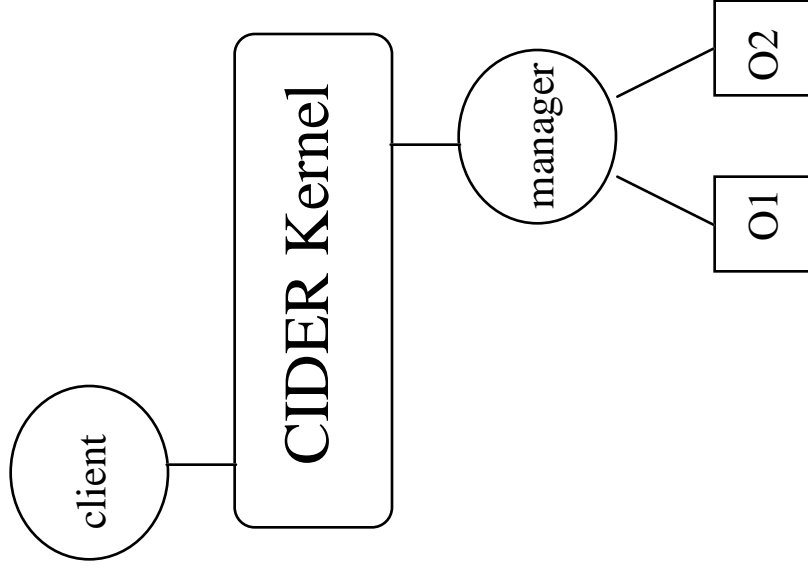
# Techniques for Establishing Trust

- Code inspection
- Design review
- Testing
- History of use
- Formal methods
- Code generation
- Design and code reuse
- Threat analysis
- Software development process
- Reliance on high-assurance security services

# CIDER

- CORBASec ORB (in progress)
- Based on THETA a secure distributed object manager
- Enforces mandatory and discretionary access controls for objects
- Runs on secure OS (e.g. Sun CMW), relies on high assurance native security services.
- Work supported by Rome Lab and NSA

# Architecture



- Kernel is distributed
- Managers own objects, implement operations
- Persistent server activation model
- Kernel handles all messages between clients and managers

# Authentication Policy

- Clients and managers authenticate themselves to the Kernel (and vice-versa) before any object access, establishing a principal.
- Assured routing of requests and replies.
- Client and “target” are assured of each other’s identity.

# Discretionary Access Control Policy (1)

- Each object has an Access Control List (ACL) which records the *rights* of principals and groups w.r.t. this object.
- Each operation has a fixed set of rights that are *required*.
- Each group has a set of members (principals).

# Discretionary Access Control Policy (2)

- A principal is entitled to the union of all of the rights from ACL entries that mention it or a group of which it is a member.
- A requested operation is allowed if the requester's rights with respect to the target are a superset of those that are required for the operation.

# Top Level Argument (1)

1. Kernel establishes identity (principal) of clients and managers.
2. Clients and managers establish identity of the Kernel.
3. Kernel ensures that requests are correctly labeled with client's principal and return address.

## Top Level Argument (2)

4. Kernel routes request to correct manager, based on previously established manager identity.
5. Kernel ensures privacy and integrity of message enroute.
6. Manager uses principal in the request to make access control decision.

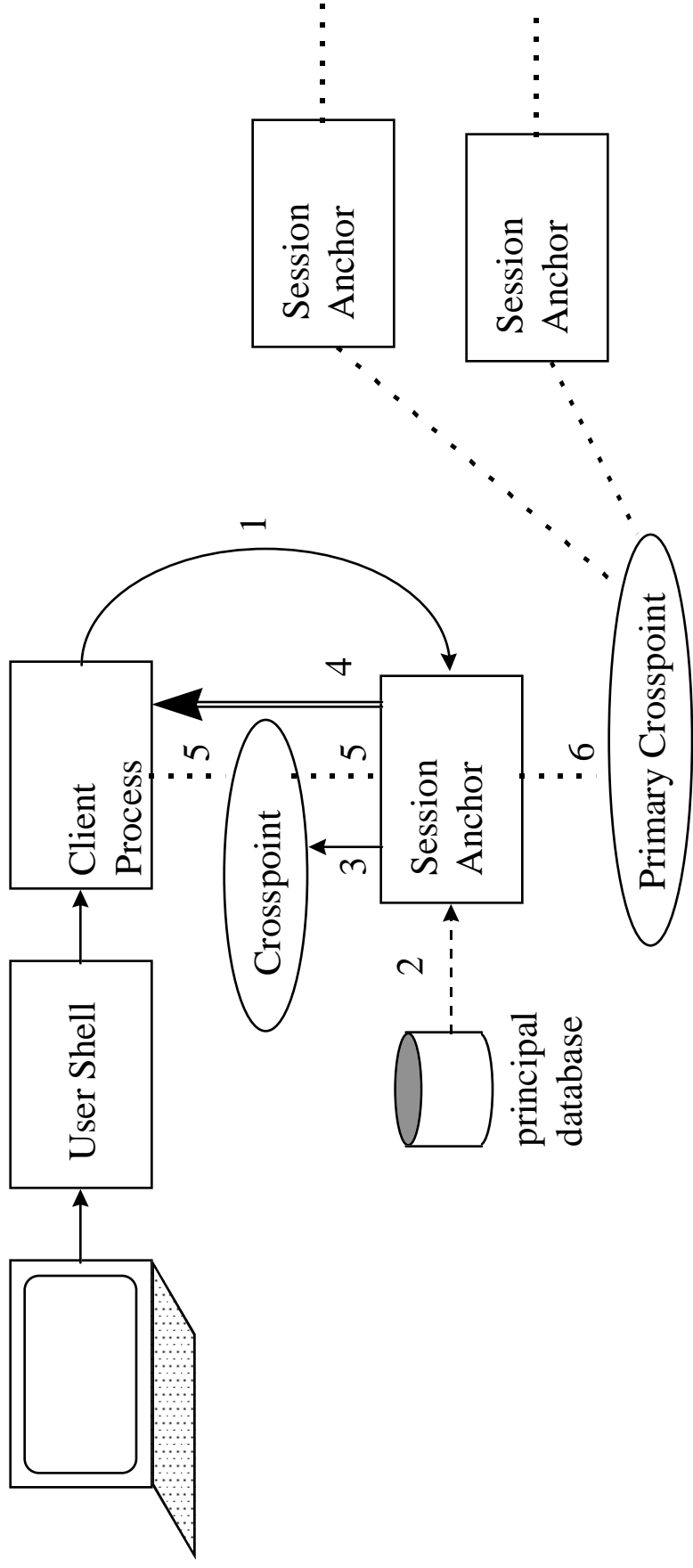
## Top Level Argument (3)

7. Manager executes operation, maintaining object and thread integrity.
8. Manager replies to correct client address, enclosing the request-id.
9. Kernel ensures that reply contains the manager's correct principal.
10. Kernel delivers reply as addressed.

# Not Bypassable

11. Manager will only accept requests from the Kernel.
12. Kernel will only deliver requests from a registered client.
13. Object state, including ACL is isolated.

# Client Process Registration



# Registration Steps (1)

- (1) Client process launches the session anchor process
- (2) Session anchor determines principal by looking up its process user ID in database
- (3) Session anchor creates shared memory crosspoint and initializes it.
- (4) Session anchor notifies client of crosspoint ID using private pipe.

## Registration Steps (2)

- (5) Client and session anchor each attach and *validate* the shared memory crosspoint.
- (6) Session anchor attaches to the primary crosspoint, validates it, and records the client's principal there.

# Kernel establishes identity (principal) of clients (1)

- 1.1 OS determines the user's identity at login and labels the shell with that user-id.
- 1.2 When the client process is launched, the OS labels it with the user-id, copied from the shell.
- 1.3 When client launches the session anchor process, the OS labels it with the client's user-id.

# Kernel establishes identity (principal) of clients (2)

**1.4** The session anchor examines its own *real* user-id, and maps it to a principal. In order to do this it consults a database (file) which defines a partial function from local user-ids to principals. This database is assumed to have high integrity.

**1.5** The session anchor writes client's principal to primary crosspoint correctly.

# Kernel establishes identity (principal) of clients (3)

- 1.6 Other parts of the Kernel trust this notation because only the session anchor (or other trusted component) could have written it there.
- 1.7 These mechanisms are tamperproof.
- 1.8 These mechanisms are not bypassable.

# Client Identity, Justifications (1)

- 1.1.1 Secure OS - assumed to have an adequately secure login procedure.
- 1.2.1&1.3.1 Secure OS - assumed to correctly propagate the process user-id to child process after *fork* and to preserve user-id after *exec*.
- 1.4.1 Secure OS - assumed to give correct answer to *getuid* system call.

# Client Identity, Justifications (2)

## 1.4.2 Integrity of the principal database

1.4.2.1 Review the procedures for administering the database.

1.4.2.2 Database file has low MAC label, preventing writes.

1.4.2.3 Database file owned and writable only by trusted user.

# Client Identity, Justifications (3)

- 1.4.3 Examine session anchor code which reads principal database file.
- 1.5.1 Examine session anchor code which writes to the primary crosspoint shared memory.
- 1.5.2 Examine session anchor code which *deregisters* to confirm cleanup of registration record in primary crosspoint.

# Client Identity, Justifications (4)

- 1.6.1 Session anchor is setuid-to-Kernel and is privileged.
- 1.6.2 Primary shared memory crosspoint is only accessible to “Kernel”.
- 1.6.3 Confirm that no other part of the Kernel with sufficient privilege will alter the principal in a registration record, by code inspection.

# Client Identity, Tamperproof (1)

- 1.7.1 OS assumed to be tamperproof
- 1.7.2 All programs which can write the primary crosspoint have executables that are privileged. Modification or replacement would invalidate that privilege and would cause OS TCB verification to fail.
- 1.7.3 Session anchor protected by ordinary OS MAC and DAC file protections.

# Client Identity, Tamperproof (2)

- 1.7.4 Separation of client and session anchor processes is guaranteed by OS.
- 1.7.5 Primary crosspoint is owned by Kernel and is accessible only to Kernel.
- 1.7.6 Primary crosspoint has a MAC label that would prevent any ordinary process from attaching.

# Client Identity, Tamperproof (3)

1.7.7 Session anchors validate the primary crosspoint by checking ownership, permissions, MAC label, etc. to rule out spoofing.

# Client Identity, Not Bypassable

- 1.8.1 Client cannot write directly in primary crosspoint -- privilege and/or unrealistic clearance required.
- 1.8.2 Client cannot use some other program besides the session anchor -- cannot create or subvert a privileged program.