The Flask Security Architecture: System Support for Diverse Security Policies

Ray Spencer *Secure Computing Corporation*
Stephen Smalley, Peter Loscocco *National Security Agency*
Mike Hibler, David Andersen, Jay Lepreau *University of Utah*
http://www.cs.utah.edu/flux/flask

Outline

- Motivation
- Background
- Policy Flexibility
- Insufficiency of Popular Mechanisms
- Flask Architecture
- Results
- Future and Ongoing Work

Motivation

No single definition of security suffices

Need for many policies and even types of policies

Computer security solutions must be flexible enough to support wide range of security policies

This policy flexibility must be supported by the OS mechanisms

Background

Distributed Trusted Mach (DTMach)

- outgrowth of TMach and LOCK
- integrated generalization of type enforcement into Mach

Distributed Trusted Operating System (DTOS)

- improved design and implementation in Mach
- prototype released to universities for research
- studies of policies, composability, security, assurability

Flux Advanced Security Kernel (Flask)

- integrated DTOS security architecture into Fluke OS
- added support for dynamic policies

Defining Policy Flexibility

Can not define through a list of known policies

Defined in context of a state machine model

- atomic operations to transition from one state to next
- policy can interpose atomically on set of controlled operations
- policy may use knowledge of portion of system state

Policy Changes

Even simplest policies undergo changes

Risk of enforcing obsolete policy

Need for effective atomicity in policy changes

Complicated by migrated permissions

- access rights explicitly cached in data structures
- access rights implicitly cached by operations in progress

Insufficiency of Popular Mechanisms

Capability-based systems

- propagation of access rights
- Hydra, KeyKOS, EROS: provide enhancements to limit propagation, but still lacking in support for policies
- SCAP, ICAP, TMach: do not define mechanisms by which policy is queried to validate capabilities

Interposition

- mismatch between functional interface and security needs
- does not support revocation of migrated permissions

Flask Architecture

Security server

- provides labeling, access and polyinstantiation decisions
- security contexts and security identifiers (SIDs)

Object managers

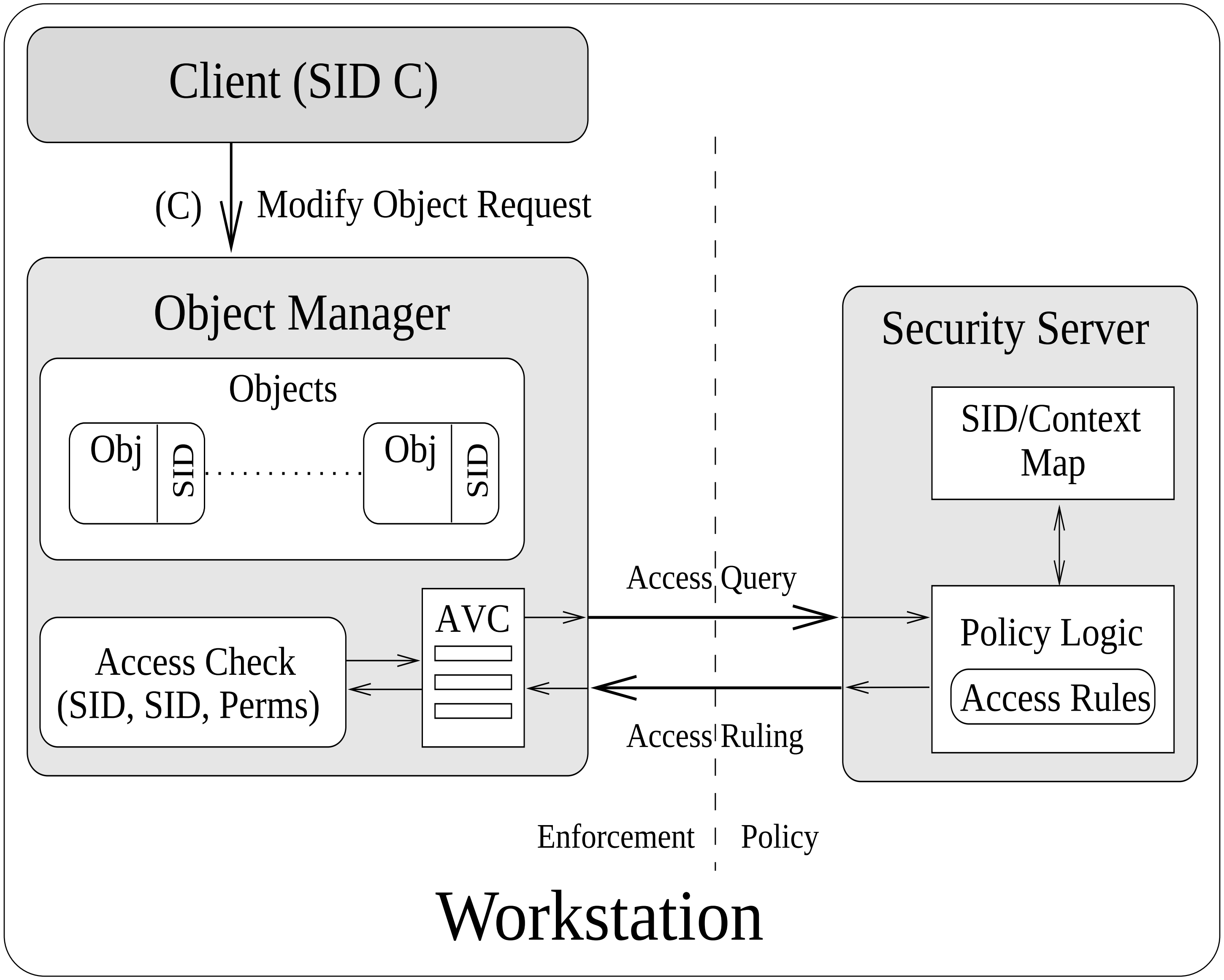
- bind labels to objects
- enforce access decisions
- direct clients to appropriate instances

Access Vector Cache (AVC) library

- coordinates access decisions
- minimizes performance impact

Underlying IPC mechanism

- must provide identification of clients and servers


Client (SID C)
(C)
Modify Object Request
Object Manager
Objects
Obj
SID
Obj
SID
Access Check
(SID, SID, Perms)
AVC
Access Query
Access Ruling
Security Server
SID/Context Map
Policy Logic
Access Rules
Enforcement
Policy
Workstation

Revocation Support

Object manager atomicity requirements

- completeness
- promptness

Protocol between security server and object managers

- system-wide atomicity for policy changes
- Security server notifies object manager AVC modules
- AVC modules update cache state, invoke callbacks
- Callbacks update migrated permissions
- AVC modules notify security server of completion

Flask Prototype

Implemented in Fluke microkernel-based OS

Security server

- combination of MLS, TE, IBAC, dynamic RBAC

Object managers

- microkernel
- virtual memory manager
- process manager
- file server
- network server

Evaluation of Flexibility

Support for policy changes

- architecture provides support for system-wide atomicity
- microkernel meets object manager atomicity requirements
- other object managers lack support for migrated permissions

Set of operations controlled by policy

- fine-grained controls over all object services

Set of operations that may be invoked by policy

- object manager interfaces, AVC module interface

System state available to policy

- SID pairs sufficient for most policies (DTOS)
- use prototype to research need for richer interface

IPC Performance Overhead

message size	Fluke (μs)	Flask (percentage overhead)		
		naive	client identification	client impersonation
"Null"	13.5	2	9	6
16-byte	15.0	2	4	6
128-byte	15.8	1	2	5
1-KB	21.9	2	2	4
4-KB	42.9	1	1	2
8-KB	78.5	1	5	1
64-KB	503	0	6	0

200Mhz PPro with 256KB L2 cache, 64MB RAM

Macrobenchmark

Compiling and linking a simple application

OS Config	Time (sec)
BSD	18.6
Fluke	39.9
Flask	41.7 (4.5%)
Flask kernel only	40.9 (2.5%)
Fluke/memoryfs	24.7
Flask/memoryfs	27.4 (11%)

200Mhz PPro with 256KB L2 cache, 64MB RAM

Future and Ongoing Work

- OSKit
- JANOS
- Linux

Questions?

http://www.cs.utah.edu/flux/flask

sds@epoch.ncsc.mil