Security-Enhanced Linux: Integrating Flexible Support for Security Policies into Linux

Peter Loscocco
Information Assurance Research Group
National Security Agency
loscocco@tycho.nsa.gov

Outline

Project Strategy

Background

The Flask Security Architecture

Linux Prototype

Project Status

Strategy

Build prototype system that addresses critical DoD/IC problems

- To gain insight and solve problems
- To understand building of secure systems
- Not to build a complete secure system

Improve security of available systems

- Demonstrate reduced vulnerability
- Work toward incorporation into Linux
- Leverage growing popularity of Linux
- Open source provides reference implementation

Critical Security Features

Secure systems require certain features

- Nondiscretionary Access Control
- Trusted Path from user to/from OS or critical security functions
- Protected Path through end system and network

Features help provide guarantees

- Data Separation/Confidentiality
- Data Integrity
- System/Application Integrity
- Nonbypassability/Guaranteed Invocation

Goal is to get features accepted into open source systems

Why not just Linux DAC?

Decisions are only based on user identity and ownership

Each user has complete discretion over his objects

Only two major categories of users: user and superuser

Many system services and privileged programs must run as superuser

No protection against malicious software

Mandatory Access Control

Traditional approach

- Can enforce system-wide security policy based on confidentiality or integrity attributes of subjects and objects
- Can support many different categories of users
- Can confine malicious code

Too limiting for general solution

- Tightly coupled to Multilevel security policy
- Assumes hierarchical relationship
- Ignores least privilege and separation of duty
- Requires trusted subjects
- No binding between security attributes of subjects and their executables (limits protection from executing malicious code)

Basic Design Principles

Secure architecture is driving concern

Flexibility of policy and mechanism

Separation of policy from enforcement

Suitable for high assurance

Performance to increase acceptability

Linux remains Linux

Policy Flexibility

Capable of Supporting wide variety of security policies

- Separation Policies
  - Enforcing Legal restrictions on data
  - Establishing well-defined user roles
  - Restrictions to classified/compartmented data
- Containment Policies
  - Restricting web server access to authorized data
  - Minimizing damages to viruses and other malicious code
- Integrity Policies
  - Protecting applications from modification
  - Preventing unauthorized modification of databases
- Invocation Policies
  - Guaranteeing that data is processed as required
  - Enforcing encryption policies

Type Enforcement

Access matrix defining permission between domains and types

Advantages

- Separates enforcement from policy
- No assumptions in labels
- Supports many policies
- No need for trusted subjects
- Controls entry into domains via program types
- Controls execution of program types by domains
- Enables assured pipelines

Downside is complexity of access matrix

History

DTMach (1993)/DTOS (1994-7)

- Roots in Mach and LOCK
- Microkernel, Security Server, and Secure Local File Service
- US release of prototype and pocumentation, provided support
- Assurance studies/flexibility assessment

Flask (1996-9)

- Security added to DARPA-funded Fluke OS (Default Option)
- Collaborative design/implementation w/U. of Utah
  - Refinements in architecture for dynamic policy support
  - Vehicle to mainstream OS research community
  - Public release early '99
- Flask Assurance
  - Formal Security Policy Model and Formal Top-Level Specification

Linux Prototype

Why Linux?

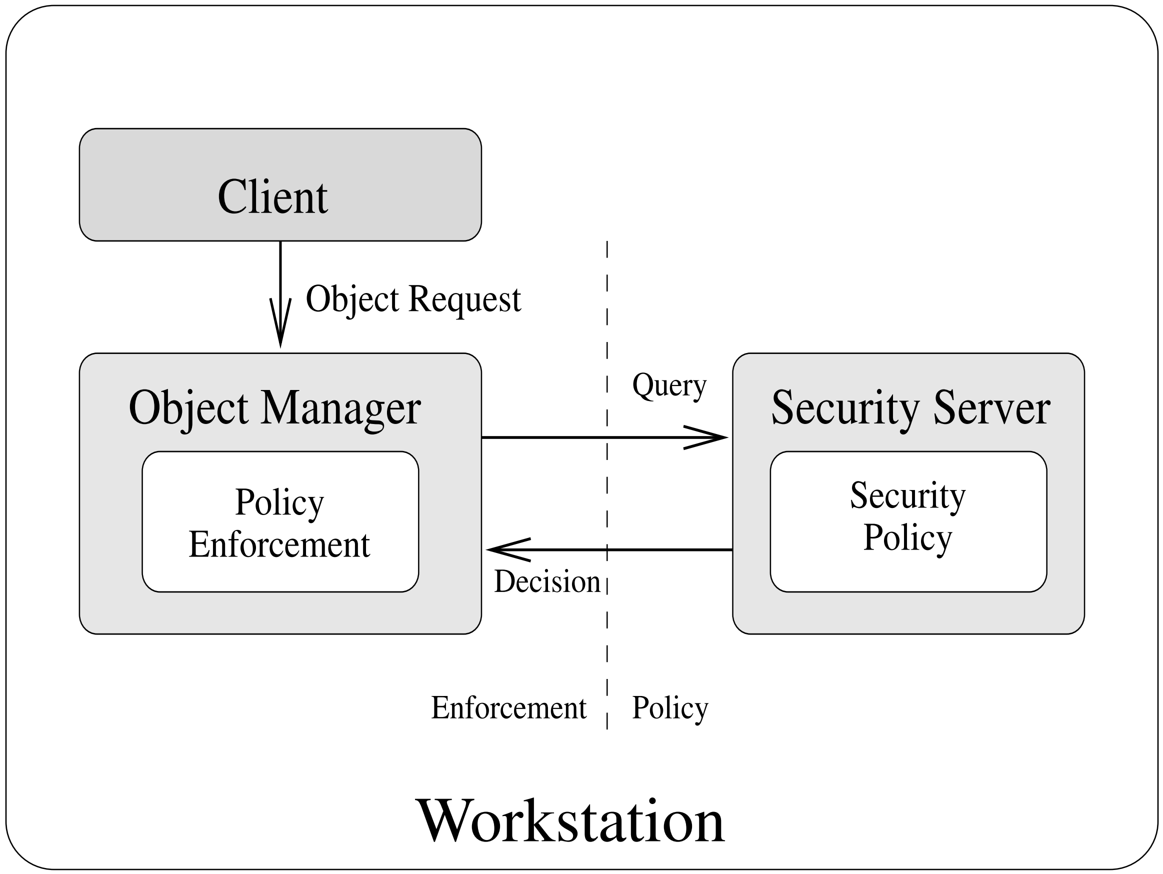
- Recognized need to move to mainstream platform
- National Security Council interest in Open Source
- Technology transfer opportunities

Application of Flask Architecture to Linux

Initial Design work Summer ‘99

Initial Public Release December ‘00

Flask Security Architecture



Architecture’s Primary Elements

Security Server Interface

- Object Labeling
- Access Decisions
- Polyinstantiation Decisions

Access Vector Cache

- Minimizes performance overhead

Object Manager Notifications

- Registered w/security server
- Notifies object managers of security policy changes

Policy Independent Data Types

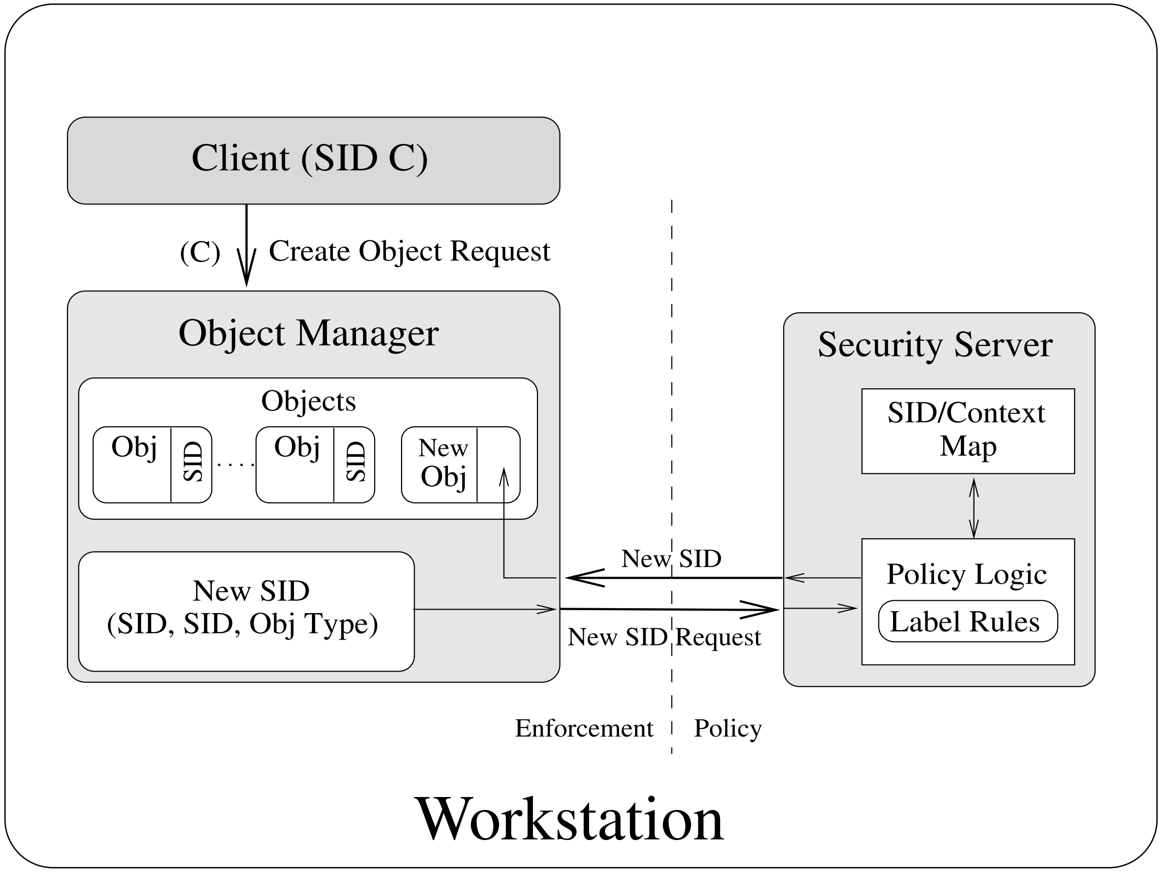
Security Context

- Represents set of security attributes
- Format dependent on security policy
- Interpreted by any application with understanding of security policy
- Variable-length string

Security Identifier (SID)

- Represents security context
- Mapping provided by security server
- Allows object manager independence from content/format of security contexts
- fixed-size value
- non-global/non-persistent

Object Labeling



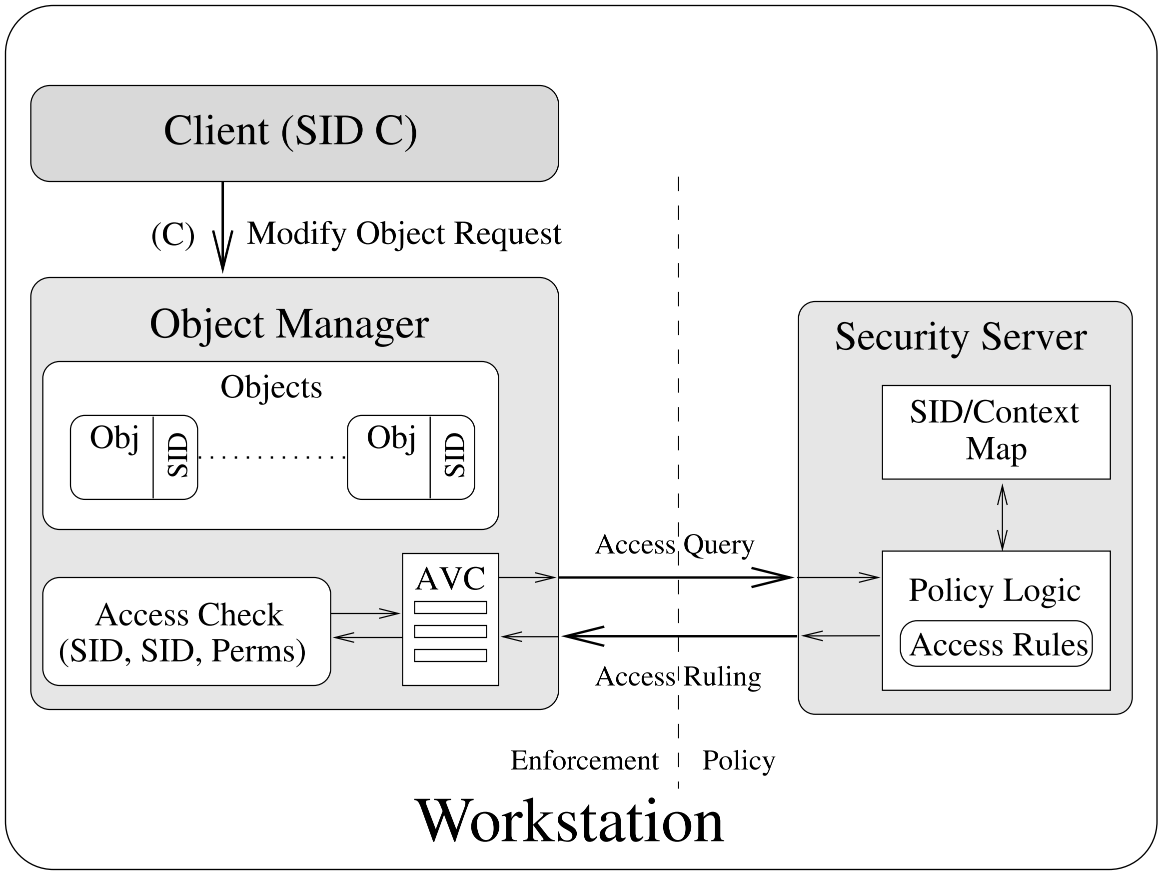
Requesting Security Decisions

Security server makes all access decisions

Accesses Vector Cache

- Eliminates need for communication and computing decisions
- Implemented as common library shared by object managers
- Coordinated security decisions and policy changes
- Access Vector contains results for related permissions

Access Decisions



Polyinstantiation Support

Allows restriction of the sharing of a fixed resource

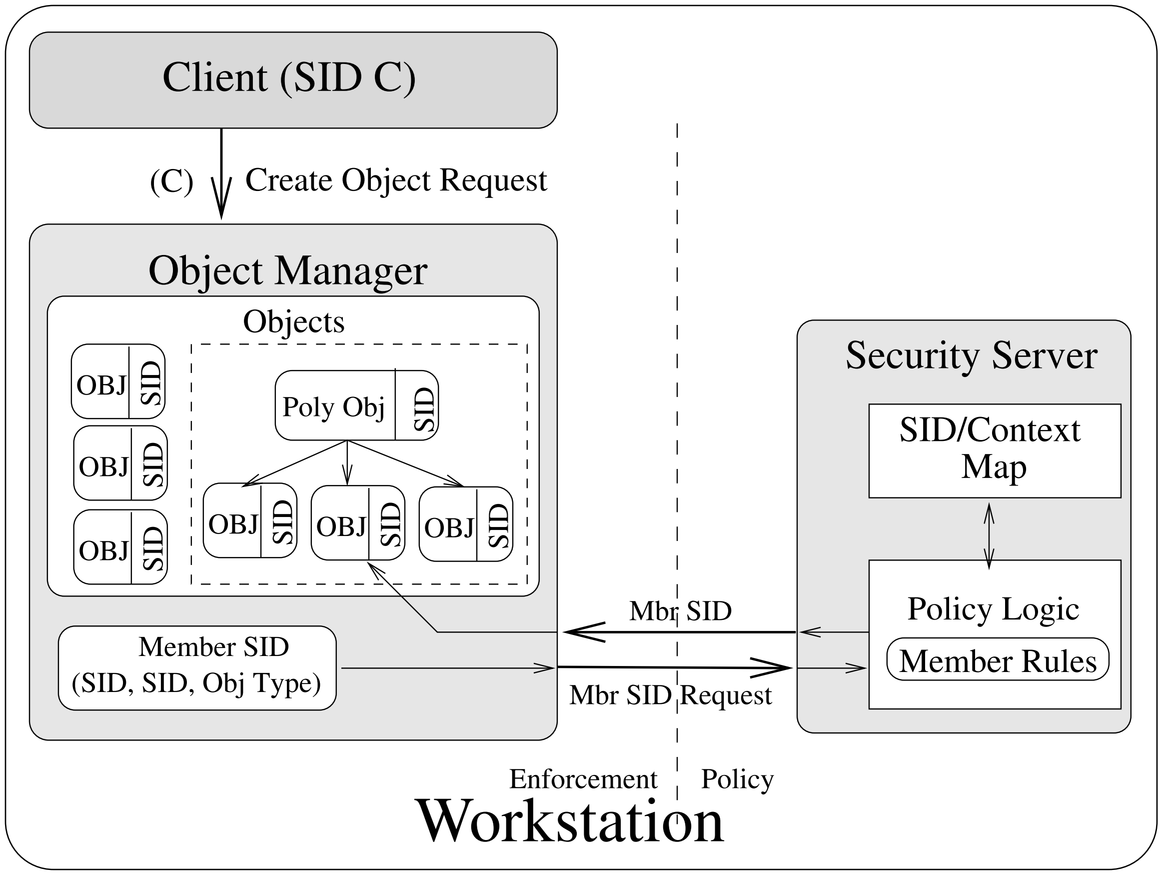
Supported via security server interface

- Security policy indicates which member to use

Examples

- /tmp in Unix
- TCP or UDP port spaces

Polyinstantiation



Validation of Architecture

Experience with Prototypes

Flask Assurance Work

- FSPM and FTLS
- Composability Report

Lessons from DTOS

- DTOS FSPM and FTLS
- DTOS Final Report
- DTOS Generalize Security Policy Specification

Consultation with NSA evaluators

Feedback from OS Research Community

Security Server

Primary Functions

- Provides security policy decisions (kernel and user-level object managers)
- Maintains mapping between SIDs and Security Contexts
- Provides SIDs for new objects
- Manages Object Manager Access Vector Cache

Additional functions

- Enforces policy over its own services
- Coordinate with other security servers in distributed environment
- Provides its own internal cache

Security Server (cont.)

Changes to policy

- Change policy database
- Change/replace security server code
- Does not require change to object managers

Current policy is combination of:

- Type Enforcement
- Dynamic Role-based Access Control (RBAC)
- Identity-based Access Control (IBAC)
- Multi-level security (experimental)

Example Security Policy

General-purpose security policy configuration

- Partial specification for reference
- Customizable
- Refinement is work in progress

Security objectives

- Kernel integrity and initialization
- System software and configuration integrity
- Protection of system administrator role and domain
- Confinement of damage caused by exploitation of flaws by limiting privileges
- Protection against privileged processes executing malicious code

File System

Controlled (labeled) objects

- File systems
- Directories
- Files
- File description objects

Labels must be persistent

- SID vs. PSID

Permissions defined for each service that observes or modifies state of file or directory

- Differs from Unix controls
- Necessary to generally support nondiscretionary policies
- Supports fine-grained distinctions between services (write vs. append for files or add_name vs. remove_name for directories

Networking

Ensures every network IPC is authorized

Must trust peer to enforce security policy

Socket is the principal controlled object

- May also bind SID to each message (datagrams)

May apply cryptographic protections

- Will use IPSEC

Negotiation

- Will use ISAKMP
- Passes security contexts not SIDs

Process Management

Controls process abstraction

- Maintains SID binding
- Controls functions like execve, kill, wait, fork, etc.
- Controls tracing
- Controls capability use
- Controls process entry into type enforcement domains

Responsible for POSIX process' secure initialization

- Security context transitions only on exec
- Shared responsibility with file system

Key Distinctions

Comprehensive and flexible system with a well-defined security architecture validated by several prototypes

Provides

- Clean separation of policy and enforcement with well-defined policy interfaces
- Specific policy, policy language, and label format independence
- Support for policy changes
- Individual labels and controls for kernel objects and services
- Caching of access decisions for efficiency
- Fine-grained controls over file systems, directories, files, open file descriptions, sockets, messages, network interfaces, and capability use
- Linux binary compatibility
- Transparency to security-unaware applications via default behavior

Current Status

Prototype

- Initial prototype with Filesystem, Process Mgmt, Network, Capabilities, and System V IPC
- Login, System Initialization and Configuration

Security Policy

- Initial base policy completed
- Apache web server, cron

Support Functions

- Modified utilities - including: find, tar, ps, ls
- New functions - runas, relabel

Extended Team

NSA

- Kernel work, policy configuration, and support customers
- Design and prototype analysis

NAI Labs

- Kernel work and policy configuration

MITRE

- Policy work, application modifications, and demonstration

Secure Computing Corp.

- Collaboration on earlier prototypes
- Policy work and system configuration

Remaining Work

Kernel Work

- Controls for additional functionality (*i.e.* IPSEC, NFS, etc.)
- Track kernel development
- Port to other open source systems
- More extensive performance testing and tuning

Policy Work

- Enhancements to policy language
- Specification and analysis tools
- Complete general-purpose policy

Applications

- Finding the best way to take advantage of new functionality

Documentation

Availability

Release

- Available at http://www.nsa.gov/selinux
- Patches to Linux 2.2.12 and 2.2.18 Development Kernel
- Patches to Linux 2.4.2 Development Kernel
- Red Hat 6.1 and 6.2 for rest of system
- Documentation

Questions

- Developers Mailing List
  - Subscribe to selinux@tycho.nsa.gov using Majordomo@tycho.nsa.gov
- Contact us directly
  - tos@epoch.ncsc.mil