Using GConf as an Example of How to Create an Userspace Object Manager

James Carter
jwcart2@tycho.nsa.gov
National Security Agency
National Information Assurance Research Laboratory
(NIARL)

Background - SELinux

- Flask architecture
  - Security server
  - Object managers
  - Access vector caches (AVCs)
- Object Managers
  - Bind security labels to their objects
  - Query the security server for labeling and access decisions
  - Enforce the security decisions of the security server

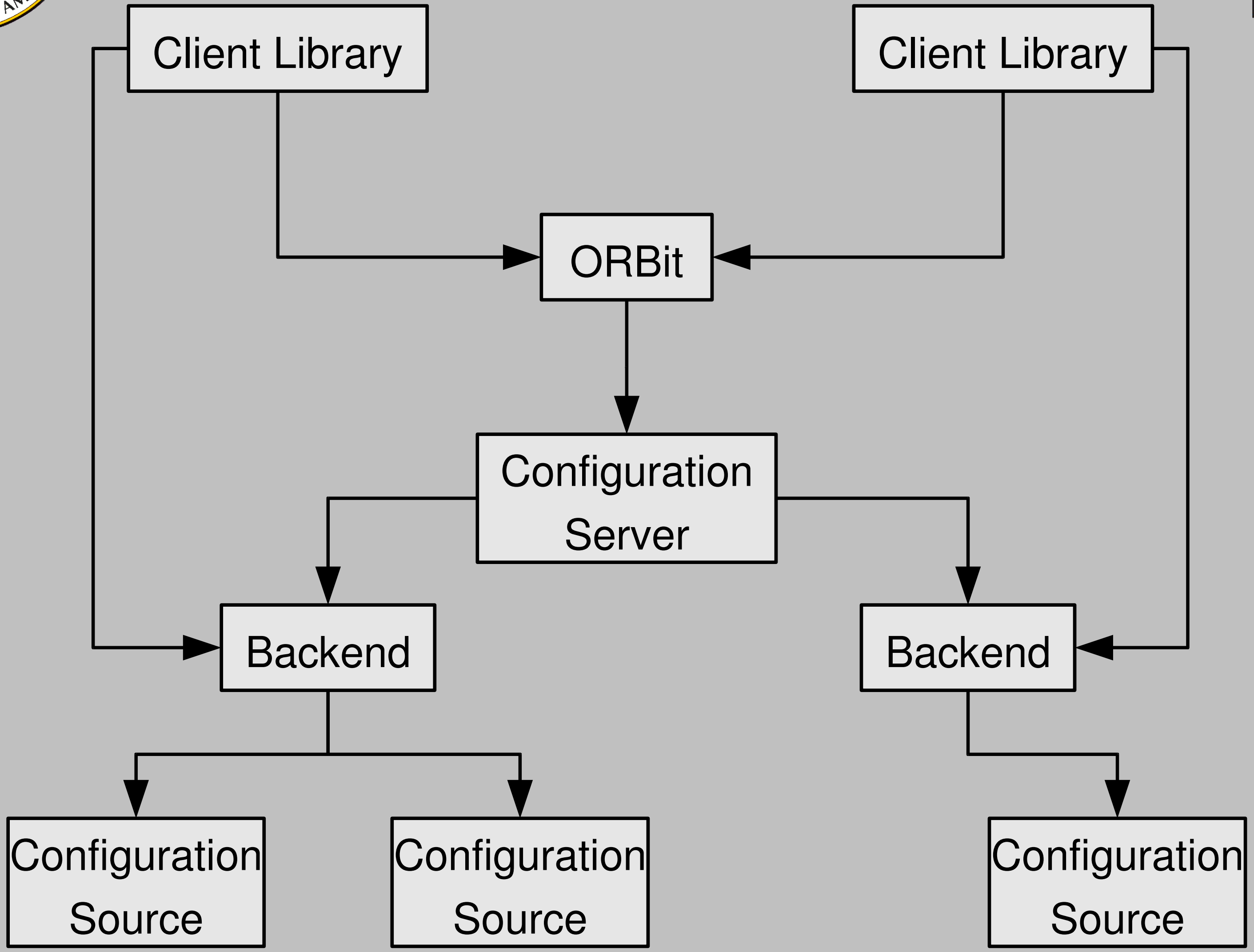
Background - GConf

- Configuration system for GNOME
  - Not GNOME specific
- Stores configuration data for programs
- Provides change notification to programs

GConf Architecture

- Configuration sources
- Client library
- Per-user configuration server
- ORBit
  - CORBA

GConf Operation



Configuration Sources

- Data: Key-value pairs
- Metadata: expected type, default value, description
- Accessed through a backend

Client Library

- Interface to access the configuration sources
- Caches configuration values
- Allows a specific set of configuration sources to be specified
- Works with the configuration server to notify the client when the value of a registered key changes

Per-user Configuration Server

- Accesses the configuration sources through the appropriate backend
- Presents a unified set of configuration data to the client
- Notifies the client library of all clients effected when the value of a key changes

Providing Security Controls over a Program

- Adequate control is often achieved by merely running an application in the domain of its parent.
- If not, then either:
  - The application should not be run
  - The security goals of the system reduced to allow the program to run, or
  - Security controls must be added

Four Strategies for Adding Security Controls over a Program

- Add SELinux policy for the program
- Add additional or finer-grained controls to SELinux
- Re-architect the program to make use of existing SELinux controls
- Modify the program to become an userspace object manager

Add SELinux Policy

- Does not require modification of the program
  - Least obtrusive strategy
- May be able to use the policy for another program with similar functions
- Custom policy involves:
  - Specifying the security label the process will run in
  - Labeling security-relevant objects
  - Specifying rules for the process and objects to interact with each other and the rest of the system

Add Additional Features to SELinux

- Add additional or finer-grained SELinux kernel controls
- SELinux is meant to have comprehensive controls over kernel objects, so new kernel controls shouldn't be required often
- If new controls are written, then new policy is needed to take advantage of those controls

Re-Architect the Program

- Decompose a program into a small, privileged process and a larger, unprivileged process
- Run multiple copies of the program in different domains
- Rewrite the program

Creating an Userspace Object Manager

- SELinux provides object managers for kernel objects
- New object managers are needed for any object not controlled by the kernel
- Natural part of implementing the Flask architecture on Linux

Functions of an Userspace Object Manager

- Bind security labels to the objects that it controls
- Request labeling and access decisions from the appropriate security server
- Enforce the decisions returned by the security server

Trust Required of an Userspace Object Manager

- Only trusted to control its objects
- Not trusted in all of its operations
- Still controlled by the system's security policy

Steps in Creating an Userspace Object Manager

- Identify the objects in greater detail
- Provide a way to uniquely and reliably label the object
- Add access checks and labeling requests where needed to control the object
- Make the subject's label available at the access checks

Steps in Creating an Userspace Object Manager (Cont)

- Add an access vector cache (AVC) to the program to cache the access decisions of the security server
- Create new SELinux policy classes and permissions as needed
- Create SELinux policy to control the objects

What Needs to be Secured in GConf

- Configuration sources
- Key-value pairs
- ORBit IORs

Adding SELinux Policy to Secure GConf

- Only the configuration server can access or modify the configuration data of the user
- Cannot label the configuration data itself

Strategies Not Used to Secure GConf

- Add additional features to SELinux
  - Configuration data of GConf is only visible to the configuration server at the appropriate granularity
- Re-Architect GConf
  - Some advantages, more disadvantages

GConf Needs to be an Userspace Object Manager

- Using the other strategies, some progress has been made
- Configuration data still not adequately controlled
- Configuration data is only visible at the right level to the configuration server
- The configuration server must be made into an userspace object manager

Labeling the Configuration Data

- Security labels stored in a separate namespace
  - /selinux
- Security labels are normal GConf value strings
- Created functions to access the security labels of a key without knowing about the namespace
- Security label always chosen from the default configuration sources

Adding Labeling Requests and Access Checks

- Access checks are done before an operation on the configuration data
  - For server-side notification registration, the check is done sooner
  - For querying all keys in a directory or all directories in a directory, the check is done after
- Labeling request is done on a set operation if the key doesn't already have a security context

Making the Client's Security Context Available

- Would like to get it from the kernel
  - Can't because the client and server communicate through ORBit
- Would like to get it from a process that the server trusts
  - Modifying ORBit to provide the context would be a lot of work
  - If D-Bus replaces ORBit, then it would be easier
- Actually trusts the client to provide the context

Add an Access Vector Cache (AVC)

- Provided by the library libselinux
- Start the AVC when the configuration server starts
- Used GConf specific memory allocation, logging, and audit callback functions

Create New SELinux Policy Class and Permissions

- Security class
  - gconf
- Permissions
  - get_value, set_value, create_value, remove_value, get_meta, set_meta, relabel_from, relabel_to

Create SELinux Policy to Control Objects

- Sensitive keys must be identified and labeled
- Processes that need to have different accesses to configuration data must run in different domains
  - Currently, most user processes run in one domain
- Only policy to test for proper operation has been written at this time

Conclusions

- Questions?