Integrating XFree86 With Security-Enhanced Linux

Eamon F. Walsh
Information Assurance Research Group
National Security Agency
ewalsh@tycho.nsa.gov
http://www.nsa.gov/selinux

Abstract

The proposed paper will discuss work, currently in progress, to add Security-Enhanced Linux policy support to the X Window System. The work consists of two modifications to the XFree86 X server implementation. The first modification is the replacement of the existing Security extension with a more general framework of hook functions and security state. The second modification is a new SELinux extension which will extend the enforcement of a kernel-based security policy to the X server in userspace.

1 Introduction

The Security-Enhanced Linux project (SELinux) is an effort to add mandatory access control to the Linux kernel, providing a level of security beyond the capabilities of traditional, discretionary UNIX permissions [8]. SELinux is built upon the Linux Security Module (LSM), a framework of hook functions and security state fields designed to support arbitrary kernel security modules [7]. The access control policy implemented by SELinux is based on the Flask architecture, a model which cleanly separates a security policy from its enforcement mechanism [6].

SELinux has gone through several years of development and, along with LSM, has been accepted into the mainline Linux kernel source code. SELinux works by labeling classes of kernel objects, such as processes, file handles, sockets, and I-nodes, with security contexts. In memory, security contexts are stored as LSM security state. Files on disk are labeled with security contexts using extended attributes, which are supported in recent versions of the Linux kernel, the GNU C library, and filesystems such as `ext2` and ReiserFS.

Each security context contains the "type" or "domain" of the object, which specifies the type of process the object belongs to (for example, `httpd_t`). SELinux security policy consists of rules specifying allowable interaction between types. For example, a process with type `httpd_t` would be permitted to access files of type `http_log_file_t`, but not files of type `passwd_t` (such as `/etc/passwd`). Security contexts also contain "user" and "role" fields, which can be used to further restrict interactions between types based on expected usage patterns for different roles. For example, execution of the `telinit` program can be restricted to processes having the `sysadm_r` role. Regular users (`user_r`) should not be able to run `telinit`.

A rich policy base has evolved, including specific policy for a large number of Linux applications and services [3]. Until now, however, SELinux has been primarily kernel-based. The only userspace work has been modifications to basic programs like `ls`, `ps`, and `login`, allowing listing and querying of security state.

2 The Problem

Window-based graphical user interfaces are subject to several security issues, among them, the need to adequately protect access to window contents and the need to securely label and identify the owner of each window [4]. Support for security policy allowing control of such access and labeling is viewed as a necessity for SELinux as Linux becomes more widely used on desktop systems.

Unfortunately, the X Window System, which provides window-based GUI functionality to UNIX and GNU/Linux operating systems, is implemented in userspace. The existing kernel-based SELinux implementation thus has no control over (or concept of) internal X objects such as windows, cursors, and color maps. The X Protocol allows client applications to read pixmap data from other windows, send events to other clients, and capture keyboard and mouse input. Window titles and icons are set by each individual client, making it possible for one client to "spoof" a window belonging to another [4].

The XFree86 X implementation currently has a "Security" extension, which provides a basic two-level trust hierarchy and has some facilities for resource access control, window content censoring, and auditing [1]. This extension is not widely used, however, and much finer-grained control is needed for SELinux policy support. Each object in the X server must be labeled with a security context, allowing, for example a Drawable object belonging to a web browser to be labeled with type

Callback Name	Basic Arguments	Return Value
RESOURCE_ACCESS	client; resource client wishes to access	Bool allow/deny
DEVICE_ACCESS	client; device client wishes to access (keyboard)	Bool allow/ignore
PROPERTY_ACCESS	client; window; name of property	Bool allow/ignore
DRAWABLE_ACCESS	client; drawable client wishes to access	Bool allow/censor
EXT_ACCESS	client; extension client wishes to use	Bool allow/ignore
HOST_ACCESS	client attempting to access the host control list	Bool allow/deny
BACKGRND_ACCESS	client attempting to create window with no background	Bool allow/censor
SITE_POLICY	check for site policy string (see [1])	Bool recognized/unrec
DECLARE_EXT_SECURE	notification that extension has declared itself trusted	void

Table 1: Callbacks available through the X Security Module.

`browser_t`. Once this is done, policy can be written governing interactions between X clients. For example, the client corresponding to a screen capture application may be permitted by the policy to access the window contents of other clients. The client corresponding to a calculator program would not receive this permission.

Providing SELinux policy support for windowing operations requires a new mode of operation for SELinux: userspace policy enforcement. The X server itself must be modified to perform the appropriate labeling of internal objects and to consult the policy whenever an object is accessed. Then, by having the X server cooperate with the kernel to obtain the policy, the desired window system security can be achieved.

3 A General Security Framework for X

The work now being done on the X Window system closely follows the development of SELinux in the kernel. Two modifications are being made to the XFree86 X server implementation [5]. The first modification, tentatively termed the "X Security Module," will replace the existing Security extension with a more general framework of hook functions and security state. This work is very similar to the LSM in the kernel, and will allow the implementation of arbitrary, stackable security extensions for XFree86.

The Security extension has snippets of code throughout the "device-independent architecture" layer of the XFree86 X server implementation. These snippets catch accesses to the X server's resource table, input devices (currently only the keyboard), and other X server extensions [2]. Calls are made back into the Security extension to check the trust level of the client making the access, and untrusted clients are denied access to untrusted extensions and resources owned by trusted clients (the trust level of each client and extension is stored in a field of their respective structures).

The X Security Module will simply replace these code snippets with more general hooks to callback lists (Table 1), and will replace the Security extension fields in the client and extension structures with a more general mechanism for storing security state. This will allow arbitrary security extensions (including a stub extension implementing the original Security functionality) to be written and loaded through the existing XFree86 module loader. These extensions may register callback functions for any hooks they wish to catch, and may obtain space to store security state with each client and extension object.

4 Kernel/Userspace Cooperation

The second modification to the XFree86 X server will be an extension which will use the X Security Module to add SELinux functionality. The extension will cooperate with the Linux kernel to obtain current security policy and will register callbacks with the security module to enforce that policy. The security context of each client will be stored in the client structure as security state and will be used to restrict access, via policy, to all resources belonging to clients (drawables, graphics contexts, etc.) Additionally, the extension will provide a new X Protocol request that will allow window managers to obtain an appropriate security label for each top-level window.

Since SELinux policy resides in the kernel, an X server (or any userspace policy enforcer) must cooperate with the kernel to ensure that the correct policy is being enforced in userspace at all times. The SELinux kernel module provides a filesystem interface, `selinuxfs`, which is similar to the `/proc` filesystem exported by the regular Linux kernel [9]. Using this interface, userspace programs can obtain information, including policy decisions, from the kernel.

A userspace policy decision cache has already been implemented for use with the X server. This cache mirrors an existing kernelspace cache, and prevents the need for a kernel trap on each and every userspace policy query. However, the kernel must notify the X server of policy change so that the cache can be flushed. The filesystem interface is poorly suited for this purpose since the X server must constantly poll a file handle to be made aware of a change. Thus, part of the project will add new messaging functionality to SELinux, per-

haps using process signals or a netlink socket to notify userspace of policy changes. This work will be applicable to all userspace policy enforcers, not just X Windows.

5 Availability

The X Window System is a trademark of the Open Group (`http://www.opengroup.org`).

The XFree86 project is an X Window System implementation, freely distributable in source form under the X11 license (`http://www.xfree86.org`).

The Security-Enhanced Linux project (`http://www.nsa.gov/selinux`) is freely distributable in source form and is available in the stock 2.6-series Linux kernel.

References

[1] Wiggins, David. "Security Extension Specification: Version 7.1." X Consortium, Inc. (1996).

[2] Wiggins, David. "Security Extension Server Design (Draft Version 3.0)." X Consortium, Inc. (1996).

[3] Smalley, S. "Configuring the SELinux Policy." NAI Labs Report #02-007 (2002). Available URL: `http://www.nsa.gov/selinux/docs.html`.

[4] Kilpatrick, D., Salamon, W., and Vance, C. "Securing the X Window System with SELinux." NAI Labs Report #03-006 (2003). Available URL: `http://www.nsa.gov/selinux/docs.html`.

[5] The XFree86 Project. Available URL: `http://www.xfree86.org`.

[6] Loscocco, P., Smalley, S., Muckelbauer, P., Taylor, R., Turner, S., and Farrell, J. "The Flask Security Architecture: System Support for Diverse Security Policies" In *Proc. 8th USENIX Conference (Security Symposium)* 1999.

[7] Smalley, S., Vance, C., and Salamon, W. "Implementing SELinux as a Linux Security Module." NAI Labs Report #01-043 (2001). Available URL: `http://www.nsa.gov/selinux/docs.html`.

[8] Loscocco, P., and Smalley, S. "Integrating Flexible Support for Security Policies into the Linux Operating System." In *Proc. 10th USENIX Conference (FREENIX Track)* 2001.

[9] Smalley, S. "NSA Security-Enhanced Linux (SELinux)." Ottowa Linux Symposium presentation (2003). Available URL: `http://www.nsa.gov/selinux/docs.html`.