Extensible Security For X: Motivation and Design
Eamon Walsh
SELinux Team
Information Assurance Research
National Security Agency
Summary

- Working towards an open source, trusted desktop.
- Need to have infrastructure for doing fine-grained access control in the X server.
- Hooks only – no specific policies.
- Local to server – no protocol changes.
- Branch development model.
What Is SELinux?

• Fine-grained Mandatory Access Control for Linux.

• Policy system based on Flask architecture.
  – Strong separation of security domains and roles.
  – Controls over process execution & resource access.
  – Diminish severity of program vulnerabilities.

• Kernel module; uses LSM security hooks.

• Some userspace changes.
<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1985</td>
<td>LOCK (early Type Enforcement)</td>
</tr>
<tr>
<td>1990</td>
<td>DTMach / DTOS</td>
</tr>
<tr>
<td>1995</td>
<td>Utah Fluke / Flask</td>
</tr>
<tr>
<td>1999</td>
<td>2.2 Linux Kernel (patch)</td>
</tr>
<tr>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td>2.4 Linux Kernel (patch)</td>
</tr>
<tr>
<td>2002</td>
<td>LSM</td>
</tr>
<tr>
<td>2003</td>
<td>2.6 Linux Kernel (mainline)</td>
</tr>
<tr>
<td></td>
<td>Present</td>
</tr>
</tbody>
</table>
SELinux Precursors

• LOCK
  – Early type enforcement.

• Distributed Trusted Mach (DTMach)
  – Improved design and implementation in Mach.

• Flux Advanced Security Kernel (Flask)
  – Flexible MAC architecture in the Flux OS.
SELinux Distributions

• Fedora Core 2
• Hardened Gentoo
• Debian (packages)
• SE-BSD (port)
• SE-Darwin (port)
SELinux Research Agenda

- Security architecture research
- Kernel prototype code
- Kernel production code
- Userspace enhancements
  - Local GUI security
- Labeled networking
- Network-wide policy
Current State of SELinux

- Core utilities: init, pam_selinux, ls, ps
- X Window System
- D-Bus
- Kernel
- Network
- File System
- /selinux
- libselinux
- /init
GUI Security

• GUI security is the last piece of the complete SELinux desktop system.

• X Window System operations should be policy-controlled.

• Need to write policy for the X Window System and have the X server enforce it.

• Generalize: make it easy to write access control extensions for the X server.
SELinux/X architecture

- X Server
- SELinux extension
- policy cache
- libselinux
- /selinux
- Kernel
- policy

security framework extension
X Security and the Network

- SELinux is currently a local system.
- SELinux does not have labeled networking or network authentication.
- X Window System big problem is client authentication over the network.
- Local security engine, new auth solution can be independent; complementary.
Goals for Security Framework

• Based on existing work.

• Easily extensible.

• Non-intrusive: based on callbacks, not local code.

• Works at dispatch (DIX) layer to avoid performance issues.

• Provides framework for arbitrary decision-making (access control) extensions.
Current XC-Security Extension

Decision Engine:
- trusted = good
- untrusted = bad

Xext/security.c
Generalized Security Extension

Xext/yourext.c
Your code here
Your state here

client structure

DIX
OS

Xext/xace.c
Easily Extensible

Xext/xsf.c

DIX

OS

FancyPants Extension
Non-Intrusive

- At decision point, only need to pass parameters to a hook function and check the result.
- Actual security code is in the callback functions.
- Separates security code from the core code.
- Whole framework is compile-time option.
Code Examples

```
ProcDoSomething(...)
{
    rval = SecurityLookupIDByType
        (client,
         MyResType, stuff->id,
         SecurityReadAccess);

    if (!rval) return BadSomething;

    DoNiftyStuff();
}
```

```
#define XACE
if (!SecurityHook
    (XACE_FOO_ACCESS,
     client, whatever))
    return BadSomething;
```
Sample Hooks

**CORE_DISPATCH**
- Replace XC-Security shadow dispatcher.

**RESOURCE_ACCESS**

**DEVICE_ACCESS**

**PROPERTY_ACCESS**
- Replace SecurityCheck*Access() functions.

**MAP_ACCESS**

**BACKGRND_ACCESS**
- Replace untrusted child & background “None” checks.
Performance Issues

• Keep hooks at the DIX layer.
• $O(1)$ hook calls per protocol request.
• Make decision before starting graphics operation.
Provides General Framework

• Arbitrary new extensions can be written to use the framework's interface.
  – Provide own state for server objects and own callback functions.

• No client-side work necessary (except for proper error handling).
How to make Security Decisions?

• Need information about the connected client.

• Obtain once - store as client state.

• Can get:
  - From the local system.
  - From the system security policy.
  - From the authentication mechanism.
Local System

ClientPtr → osPtr → fd

getpeercred → UID, GID, PID

getpeercon → security context

/proc
Local Security Policy

X server w/security framework

hook function
security extension

description of access event
decision

policy server or trusted 3rd party
Authentication Protocol

X Server

os auth layer

AUTH_AVAIL security hook

client structure

remote client

auth data
Authentication Protocol, cont'd.

- Opportunity to combine power of the security framework with new, secure authentication methods.
- Design protocol, then write security extension to do fine-grained access control.
- At connect time, pass auth data to a security hook.
- Callbacks on that hook can set client state based on the auth data.
Other Security Issues

- Trusted window labeling
  - Pass some String label to window manager on request.
  - Define a standard way to do this (new extension).
  - Or, use a Property on the window (that other clients can't mess with).
In Closing

- Flexible MAC on the open-source desktop is within reach.
- Generalized security engine, as described, will benefit SELinux project and others.
- Combine with better authentication for full solution.
Contact Information

- http://www.nsa.gov/selinux
- selinux-team@epoch.ncsc.mil
- ewalsh@epoch.ncsc.mil