SELinux and MLS: Putting the Pieces Together

Chad Hanson
Trusted Computer Solutions, Inc.
121 W Goose Alley
Urbana, IL 61801 USA
chanson@TrustedCS.com
Today’s Presentation

- What is MLS?
- Why MLS in SELinux?
- Dynamic MLS Support
- MLS Policy Enhancements
- MLS Privileges
- MLS Policy Language
- Compact Notation
- MLS Translation
- MLS Policy Creation
- Current MLS Status
What is MLS?

- Multi-Level Security (MLS) Theory
  - Mandatory Access Control Policy
  - Confidentiality
  - Bell-LaPadula Model (BLP)
    - Policy Rules (No Read Up, No Write Down)
      - Simple Security Property (No Read Up)
      - *-Property “Star-Property” (No Write Down)
Why MLS in SELinux?

- MLS is a complimentary model to Type Enforcement (TE)
  - Can easily describe complex confidentiality relationships
  - Utilized by the DoD and Intelligence Community

- Flexibility of Flask Architecture
  - Modular Support for Policies

- Strength in Combination of MAC Models
  - Simplify Integrity and Confidentiality
  - Stronger than existing MLS models
  - Strong Privilege Model
  - Static Analysis
Dynamic MLS Support

- Rework experimental framework MLS to be acceptable upstream
  - MLS support should be transparent
  - Kernel
    - Remove config option
    - Runtime support MLS policy
  - User-space
    - Remove requirement for separate binaries
    - Runtime support for MLS policy generation
MLS Policy Enhancements

- Removed base permission model
- Defined policy through an extension of constraint language
  - Constraint rules define requirements on a class – permission pairing
  - Added MLS component

- Policy Additions
  - validatetrans
  - range_transition

- Highly flexible to meet custom policy needs
  - Allow granular overrides
  - Allow policy experimentation
MLS Privileges

- A granular mechanism for special actions and overrides
- A fine-grained set of MLS privileges is implemented using type attributes

**Examples:**

```c
attribute mlsfileread;
attribute mlsfilewrite;
```

- Processes gain the use of an MLS privilege by executing in a domain which has the associated attribute.

**Example:**

```c
typeattribute init_t mlsfileread;
mld_file_read_up(init_t)
```
MLS Policy Language

- New process expressions in constraint language:
  - L1, H1 & L2, H2 – MLS range (low, high) of context 1 & 2

- Label Comparison Operators: \textit{eq, dom, domby, incom}

Example:

\begin{verbatim}
mlsconstrain \{ \text{dir file lnk\_file chr\_file blk\_file}
  sock\_file fifo\_file \} \{ \text{read getattr execute} \}
  (( l1 dom l2) or
   (( t1 == mlsfilereadtoclr ) and ( h1 dom l2 )) or
   ( t1 == mlsfileread ) or
   ( t2 == mlstrustedobject ));
\end{verbatim}
• `validate_trans` rule defines additional requirements for upgrading or downgrading an object
  – Not tied to a particular permission
  – Defined for the security class as a whole
  – Triplet (object 1, object 2, process)
  – New Process Expressions
    • U3, R3, T3 – User, Role and Type of context 3

**Scenario:**
  – Relabeling a file from U to SBU requires `mlsfileupgrade`
Example:

```plaintext
mlsvalidatetrans { dir file lnk_file chr_file blk_file
   sock_file fifo_file }
   ((( l1 eq l2 ) or
     ((( t3 == mlsfileupgrade ) and ( l1 domby l2 )) or
       ((( t3 == mlsfiledowngrade ) and ( l1 dom l2 )) or
         ((( t3 == mlsfiledowngrade ) and ( l1 incomp l2 )))
     )
    and
    ((( h1 eq h2 ) or
      ((( t3 == mlsfileupgrade ) and ( h1 domby h2 )) or
        ((( t3 == mlsfiledowngrade ) and ( h1 dom h2 )) or
          ((( t3 == mlsfiledowngrade ) and ( h1 incomp h2 )))));
```
• MLS is the last component in the security context
• MLS Range comprised of
  Sensitivity Labels (SL)
  – Effective (low)
  – Clearance (high)
• SL contains
  – Classification / Sensitivity
    • Hierarchical
  – Compartment / Category
    • Non-hierarchical
MLS and SELinux Security Context

• Subject
  – Effective (low)
  – Clearance (high)

  Example:
  `system_u:system_r:initrc_t:s0-s15:c0.c255`

• Objects
  – Single Level, directories maybe allowed a range
  – Objects can specified as “trusted” to allow subject access

  Example:
  `system_u:object_r:initrc_exec_t:s0`
Compact Notation

• With a default setting of 255 compartments and support for much larger compartment sets, the security context can be very large

• Introduced concept to collapse adjacent compartments

• Greatly reduces context size when all compartments are active

\[
\text{s2:c0.c2} \quad \iff \quad \text{s2:c0,c1,c2}
\]
MLS Translation

- Flexible mechanism to give meaningful names to SLs
- Support is integrated into libselinux
- Translation library (libsetrans) supports two interfaces
  - Native to Translated
  - Translated to Native
- Translation library is replaceable by third party apps
- Useable by other policies

`s15:c0.c255` ↔ `SystemHigh`
MLS Policy Creation

- Objects
  - Stored at SL representing the highest level of the data within
  - Most system files are at SystemLow
    - Binaries, libraries, etc...
  - Some system objects are at SystemHigh
    - Audit logs
    - System Memory (/dev/kmem)
    - Hard Disks (/dev/sda)
  - Special objects maybe “trusted” to allow writes
    - Null Device (/dev/null)
  - Yet others are best virtualized
    - Home Directories
MLS Policy Creation

- **Subjects**
  - SL should represent the highest level of the data
  - Label Changes restricted following actions
    - New executable image
      - Policy rule - range_transition
      - Process execution context - setexeccon
    - Dynamic transition
      - Change process context – setcon
        » Process must have MLS privilege
        » Process must remain in Type Hierarchy
Current MLS Status

• MLS accepted into Linux Kernel 2.6.12
• MLS support in upstream user-space policy tools
• MLS policy present in Fedora Core 5
• MLS translation is in Fedora Core 5
• MLS Audit is development
• Additional MLS user-space utilities in development

• X Windows
  – Ported XACE to X.org 6.8.2 release
  – Prototype ported to Xorg 7.0
    • Working with community for acceptance
Simplify Secure Information Sharing

www.TrustedCS.com