Porting Legacy Multilevel Secure Applications to Security Enhanced Linux

Andy Suchoski

Rick Supplee
Hewlett Packard

Contents

- Legacy multilevel secure systems (MLS) environment
- MLS Features and architecture
- Mapping MLS security features to Security Enhanced Linux (SELinux) security features
- Code Examples
- Conclusions

Legacy Multilevel Secure (MLS) Systems

- Produced to meet specifications in
  - The Orange Book (DOD TCSEC)
  - Defense Intelligence Agency Compartmented Mode Workstation (CMW) Specification
- Examples
  - Sun Trusted Solaris
  - SCO CMW
  - HP-UX CMW
  - IBM AIX CMW
  - And others

Legacy MLS System Features

- Modifications of standard commercial products:
  - Sensitivity Labels
  - Privileges
  - Roles and Authorizations
  - Trusted Networking
  - Trusted Windows
- Lagged commercial product releases
- Difficult to support
- Relegated to mostly DOD and Intelligence agencies

Sensitivity Labels

- New security attribute on every subject and object
- Sensitivity level plus zero or more compartments
- Bell-LaPadula access rules

  "No read up; no write down"
- New and modified system utilities
- File system media changes
- Process clearance as well as sensitivity label
- CMW had advisory information label (float)
- Systems had extensive API for manipulating sensitivity labels in binary format

Privileges

- Elimination of EUID=0 privilege
- Discrete privileges to override both Discretionary Access Controls (DAC) and Mandatory Access Controls (MAC)
- Program file privileges and process privileges
- Privilege bracketing using Application Programming Interface (API) calls
- Approximately 60 to over 80 discrete privileges

Roles and Authorizations

- Roles and authorizations provided administrative abilities and selective override of restrictions
- Roles and authorizations are assigned to user accounts
- Programs check for presence of role or authorization and use privilege to accomplish task
- Extensible and implemented entirely by trusted programs

Other Security Features

- Trusted Networking
  - Labeled network packets with security information
  - Controlled flow of information across interfaces to hosts based on sensitivity level of data
- Trusted Windows
  - Extended MAC to window objects
  - Window security bar
  - Color associated with sensitivity level

Mapping Legacy MLS Features to SELinux

- Sensitivity Labels are in policies in FC5 and later
- Privileges map to a combination of Linux and SELinux security features
- Roles are defined in SELinux policy
- Trusted networking is in RHEL5
- Trusted X11 Windows is in development

Sensitivity Labels in SELinux

- Range is one of the security elements of the security context
  - For processes, effective sensitivity level and process clearance; low label-high label. (mls policy)
  - Always string values in user space instead of binary structures in legacy MLS systems
  - SELinux API is not as extensive as legacy MLS systems
    - No level comparisons or level bounds
    - No label initialization to Syshi or Syslo
    - Equivalent functionality may require multiple calls in SELinux

Privilege in SELinux – Multiple Mechanisms

- DAC security done in Linux kernel
  - EUID=0
  - Linux capabilities
- MAC done in SELinux security module (LSM)
  - Capabilities assigned to process domain
  - Domain attributes for security context
- Privilege limited by type enforcement
- API for Linux capabilities only
- Legacy MLS program privileges require policy work

Roles in SELinux

- Roles were completely implemented by trusted programs in legacy MLS systems
- In SElinux, there is no comparable API for roles
  - Domains are assigned to roles
  - Processes run in domains
    -> No API
- Roles in SELinux require policy work rather than program code

Trusted Solaris Code to get Sensitivity Label

```
#include <tsol/label.h>
main()
{
  int retval, length = 0;
  bclabel_t fileCMWlabel;
  bslabel_t fsenslabel;
  char *string = NULL;
  /* Get file CMW label */
  getcmwlabel("/app/foobar", &fileCMWlabel);
  /* Get sensitivity label portion */
  getcsl(&fsenslabel, &fileCMWlabel);
  /* Translate file SL and print */
  bsltos(&fsenslabel, &string, length, LONG_CLASSIFICATION);
  printf("File sensitivity label = %s\n", string);
}
[OUTPUT]
File sensitivity label = CONFIDENTIAL
```

SELinux Code to get Sensitivity Label

```
#include <stdio.h>
#include <selinux/selinux.h>
#include <selinux/context.h>
main()
{
  int retval;
  security_context_t seccon;
  context_t contxt;
  /* Get file context */
  retval=getfilecon("/app/foobar", &seccon);
  /* Convert to a context_t struct*/
  contxt=context_new(seccon);
  /* Print the range */
  printf("File Range is %s\n", context_range_get(contxt));
}
[OUTPUT]
File Range is Confidential
```

Comparison

- Straightforward port
  - Trusted Solaris
    - getcmwlabel – retrieve CMW label (sensitivity label plus information label
    - getcsl – extract sensitivity label from CMW label structure
    - bltos – convert binary sensitivity label to string
  - SELinux
    - getfilecon – get retrieve security context from file
    - context_new – initialize a new security context structure
    - context_range_get – extract the range (sensitivity label)

Trusted Solaris Code to change a file Sensitivity Level and reread the label

- Comparable API label routines
- Privileges needed to change MAC attribute
  - PRIV_FILE_DAC_WRITE – file owner not EUID of process
  - PRIV_FILE_MAC_READ – reread of upgraded label
  - PRIV_FILE_UPGRADE – ability to upgrade the label
  - PRIV_SYS_TRANS_LABEL – ability to use a higher label
- Privilege call to raise privileges before change
- Privilege call to lower privileges after label is reread
- Privilege assignment to program file

SELinux Code to change a file Sensitivity Level and reread the label

- Comparable API label routines
- Policy work to create a new domain with
  - Domain type attributes
    can_change_object_identity – change an element of the security context when not the user in the context
    mlsfileupgrade – upgrade the range element
    mlsfilereadtoclr – read a file with a sensitivity level higher than the process level but less than the process clearance
  - Capability
    fowner – change a file attr where EUID<> file owner
- Additional type enforcement rules

Conclusions

- Porting code from legacy MLS systems will require both code translation and policy work
- Applications will have fewer security specific API calls
- Processes may run with EUID=0 instead of using capabilities to pass Linux DAC
- Sensitivity label code may require extra work
- API to get/set security attributes is presently only for processes and file types

Questions?

Solaris to Linux Porting Guide

http://devresource.hp.com/drc/topics/solaris_linux.jsp

Contact information:

andy.suchoski@bearingpoint.com