Mandatory Access Control and SE Linux

CS 460 Cyber Security Lab

Spring ‘10

Overview

- Review mandatory access control
- Discuss SE Linux
  - Type Enforcement Model
  - MLS or Bell-LaPadula model
  - Multiple Category Security (MCS)

MAC vs DAC

- Discretionary Access Control (DAC)
  - Normal users can change access control state directly assuming they have appropriate permissions
  - Access control implemented in standard OS's, e.g., Unix, Linux, Windows
  - Access control is at the discretion of the user
- Mandatory Access Control (MAC)
  - Enforced by system wide set of rules
  - Normal user cannot change access control schema
- "Strong" system security requires MAC
  - Normal users cannot be trusted

Confidentiality Policy

- Goal: prevent the unauthorized disclosure of information
  - Deals with information flow
  - Integrity incidental
- Multi-level security models are best-known examples
  - Bell-LaPadula Model basis for many, or most, of these

Bell-LaPadula Model, Step 1

- Security levels arranged in linear ordering
  - Top Secret: highest
  - Secret
  - Confidential
  - Unclassified: lowest
- Levels consist of *security clearance* $L(s)$
  - Objects have *security classification* $L(o)$

Bell, LaPadula 73

Example

security level	*subject*	*object*
Top Secret	Tamara	Personnel Files
Secret	Samuel	E-Mail Files
Confidential	Claire	Activity Logs
Unclassified	Ulaley	Telephone Lists

- Tamara can read all files
- Claire cannot read Personnel or E-Mail Files
- Ulaley can only read Telephone Lists

Reading Information

- Information flows *up*, not *down*
  - "Reads up" disallowed, "reads down" allowed
- Simple Security Condition (Step 1)
  - Subject *s* can read object *o* iff, $L(o) \le L(s)$ and *s* has permission to read *o*
    - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
  - Sometimes called "no reads up" rule

Writing Information

- Information flows up, not down
  - "Writes up" allowed, "writes down" disallowed
- *-Property (Step 1)
  - Subject *s* can write object *o* iff $L(s) \le L(o)$ and *s* has permission to write *o*
    - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
  - Sometimes called "no writes down" rule

Basic Security Theorem, Step 1

- If a system is initially in a secure state, and every transition of the system satisfies the simple security condition (step 1), and the *-property (step 1), then every state of the system is secure
  - Proof: induct on the number of transitions
- Meaning of "secure" in axiomatic

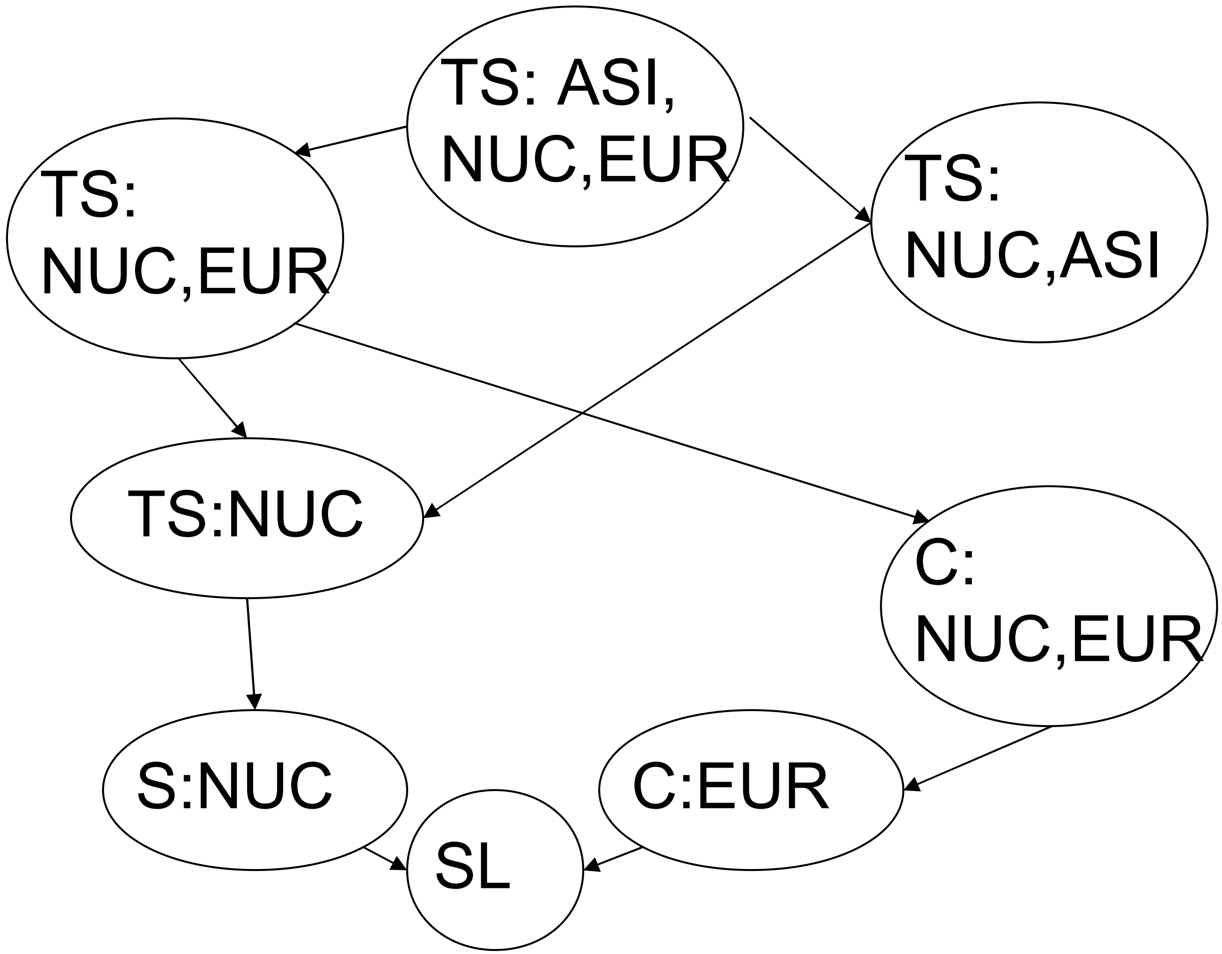
Bell-LaPadula Model, Step 2

- Expand notion of security level to include categories (also called compartments)
- Security level is (*clearance*, *category set*)
- Examples
  - (Top Secret, { NUC, EUR, ASI })
  - (Confidential, { EUR, ASI })
  - (Secret, { NUC, ASI })

Levels and Lattices

- (A, C) *dom* (A', C') iff $A' \leq A$ and $C' \subseteq C$
- Examples
  - (Top Secret, {NUC, ASI}) *dom* (Secret, {NUC})
  - (Secret, {NUC, EUR}) *dom* (Confidential,{NUC, EUR})
  - (Top Secret, {NUC}) ¬*dom* (Confidential, {EUR})
  - (Secret, {NUC}) ¬*dom* (Confidential,{NUC, EUR})
- Let C be set of classifications, K set of categories. Set of security levels $L = C \times K$, *dom* form lattice
  - *Partially ordered set*
  - *Any pair of elements*
    - *Has a greatest lower bound*
    - *Has a least upper bound*

Example Lattice



Levels and Ordering

- Security levels partially ordered
  - Any pair of security levels may (or may not) be related by *dom*
- “dominates” serves the role of “greater than” in step 1
  - “greater than” is a total ordering, though

Reading Information

- Information flows *up*, not *down*
  - "Reads up" disallowed, "reads down" allowed
- Simple Security Condition (Step 2)
  - Subject *s* can read object *o* iff *L*(*s*) *dom* *L*(*o*) and *s* has permission to read *o*
    - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
  - Sometimes called "no reads up" rule

Writing Information

- Information flows up, not down
  - “Writes up” allowed, “writes down” disallowed
- *-Property (Step 2)
  - Subject *s* can write object *o* iff *L*(*o*) *dom* *L*(*s*) and *s* has permission to write *o*
    - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
  - Sometimes called “no writes down” rule

Basic Security Theorem, Step 2

- If a system is initially in a secure state, and every transition of the system satisfies the simple security condition (step 2), and the *-property (step 2), then every state of the system is secure
  - Proof: induct on the number of transitions
  - In actual Basic Security Theorem, discretionary access control treated as third property, and simple security property and *-property phrased to eliminate discretionary part of the definitions — but simpler to express the way done here.

Problem

- Colonel has (Secret, {NUC, EUR}) clearance
- Major has (Secret, {EUR}) clearance
- Can Major write data that Colonel can read?
- Can Major read data that Colonel wrote?
- What about the reverse?

Solution

- Define maximum, current levels for subjects
  - *maxlevel*(*s*) *dom curlevel*(*s*)
- Example
  - Treat Major as an object (Colonel is writing to him/her)
  - Colonel has *maxlevel* (Secret, { NUC, EUR })
  - Colonel sets *curlevel* to (Secret, { EUR })
  - Now *L*(Major) *dom curlevel*(Colonel)
    - Colonel can write to Major without violating "no writes down"
  - Does *L*(*s*) mean *curlevel*(*s*) or *maxlevel*(*s*)?
    - Formally, we need a more precise notation

Adjustments to “write up”

- General write permission is both read and write
  - So both simple security condition and *-property apply
  - S dom O and O dom S means S=O
- BLP discuss append as a “pure” write so writeup still applies

BLP in OS's

- Multi-level systems (MLS) implemented in OS's follow BLP
  - Many Trusted OS's evaluated over the years.
  - Trusted Solaris is probably most widely deployed
- Often people use the concepts of MAC and MLS and BLP interchangeably
  - But there exist other MAC models
- There are also mandatory integrity models
  - But we won't go there today…

Example Scenario

Role	User	Clearance	Projects
Project Manager	Alice	High	Proj1,Proj2, Proj3
Intern	Bob	Low	Proj1,Proj2
Dev Manager	Charles	High	Proj1

Sensitivity Labels

User	**Sensitivity Label**
Alice	High:Proj1,Proj2,Proj3
Bob	Low:Proj1,Proj2
Charles	High:Proj1

Operations

- What is the highest Proj1 file label such that
  - Alice and Bob can both read?
  - Alice and Charles can both read?
  - All three can read
- What about write?

SE Linux Security Architecture

- A bolt-on to the basic Unix security model
  - Implements a security server to interpret security policy
  - Leaves basic Unix security mechanisms alone. But replace key programs to require security server approval as well
    - E.g. the SE Linux identity and the Linux user are two separate things.
    - SE Linux labeling and Unix DAC are both applied

SELinux Architecture

- Sponsored by NSA
- Evolved from Flask architecture precursor
- Originally direct kernel patch
  - Moved to use the Linux Security Module (LSM)
  - Limited number of tools that can hook into LSM
- Meeting Critical Security Objectives with Security-Enhanced Linux
  - http://www.nsa.gov/research/_files/selinux/papers/ottawa01-abs.shtml

Key SELinux Concepts

- Users – Identifier for a single user or an equivalence class of users
- Class – Type of an object, e.g., file or process
- Roles – Specification of privileges or actions that can be taken by user fulfilling a role
- Domains – Classification of a subject
- Types – Classification of an object (really the same thing as a domain but applied to objects)

SELinux Concepts

- Two basic security enforcement decisions
  - **Access control**: Can subject access object?
  - **Labeling**: What label should a new object have?
- Very general policy language enables the specification of many models.
  - Ships with a targeted policy enabled.

SE Linux Type Enforcement (TE)

- Access controlled by unstructured label called a type
  - When labeling a process the type is sometimes called a domain
- Policy defines access rules in terms of process and file types
  - ```
    allow <subject type>
    <target type>:<class set> <permission set>
    ```
  - ```
    allow httpd_t http_config_t:file
                      { read, write };
    ```

Example TE mapping

User	Domain or type
Alice	Proj1, Proj2, Proj3, SecretProj1, SecretProj2
Bob	ROProj1, ROProj2
Charles	Proj1, SecretProj1

TE Rules

- `allow Proj1 ProjData:file { read, write, execute };`
- `allow ROProj1 ProjData:file { read, execute };`
- `allow SecretProj1 SecretProj1Data: file { read, write, execute };`

Operations

- How must data be labeled for Alice, Bob, and Charles to coordinate on Proj1?
- How must sensitive Proj1 data be labeled?
- Can Bob write any Proj1 data?

SE Linux Concepts

- Entities are labeled with a *security context*
  - User, Role, Type or Domain
  - E.g., Bob:user_r:corporate_t
  - When displayed from the “id” command means
    - Logged on as user Bob fulfilling the user_r role in the corporate_t domain
  - When displayed off file foo from “ls –Z foo” means
    - Created by user Bob while in user_r role. Member of corporate_t type

Policy Language Overview

- Type declaration
  - **type** *type-name* [**alias** *alias-id*] [, *attr-id*] **;**
  - E.g., **type sshd_t, domain, privuser, privrole;**
  - Binds a type name to some attributes
- Attributes are arbitrary tags associated with types at definition type
- In many places in policy attributes can be used in place of direct types
  - **`allow domain unlabeled_t:file { read, write, execute };`**
- Also used in implementing MLS. More later.

Type Transition

- **Defines the rules for the type of a new object**
  - **type_transition** *source_types target_types* **:** *classes new_type* **;**
    - Source_type is the type/domain of the creating subject.
    - Target_type is the type of the parent object, e.g. directory in the file system case
  - E.g., type_transition sshd_t tmp_t : devfile_class_set cardmsg_dev_t ;
    - When sshd daemon creates a device file in the tmp directory, the new file is labeled with cardmsg_dev_t
    - devfile_class_set is a M4 macro

Access Vector Rules

- Rules that determine which domains can access which types
  - (**allow | auditallow | dontaudit**) *src_type target_type* **:** *classes permissions* **;**
  - When a subject of src_type accesses an object of target_type, it has the specified permissions if object is one of the specified classes
  - E.g., allow sshd_t shell_exec_t : file execute;

Role Based Access Control

- Provide indirection between a user and the privileges of a user
  - A user can fulfill multiple roles
  - Multiple users can fulfill the same role
  - User groups can act as a weak substitution for Roles
- User may be capable of multiple roles but will only operate with one active role
  - Reduce privilege exposure

Role Syntax

- Role Definition
  - **role** *name* **type** *type_set* **;**
  - Defines which domains (types) a role can be assumed in
  - E.g., role staff_r type staff_t;
- Role Allow
  - **allow** *current_role new_role* **;**
  - E.g., allow staff_t sysadm_t ;
  - If not specified cannot take one new role from current role

Domain Transitions

- By default new process inherits domain of creating process
- Can create additional rules to enable a domain transition
  - type_transition d1 d2:process f1
  - Plus three allow rules to permit execute access between the three types

TE Policy Problems

- Explicit rule base policy gives expressibility, but…
- Policies become very large
  - 150,000 rules in “targeted” SE Linux policy (after macro expansion
- Policy language is powerful, but very low level
  - Macros used to approximate program modularity
  - Analysis tools work post macro

Modular Policy

- Pre-FC5 all policy files just plugged into a single file and compiled
  - Must reload whole policy to add policy for new app
  - App-specific policy must depend on specific names of existing policy
- Modular mechanism enables defining type parameters
  - http://www.redhat.com/docs/manuals/enterprise/RHEL-5

MLS in SE Linux

- A parallel security model that can be executed in addition to type enforcement
- Augment the security context with a sensitivity label
  - Sensitivity label equals one of 16 clearance levels, and a subset of 256 compartments
  - Bob:user_r:corporate_t:s0_c0,c5,c10

MLS in SE Linux

- Leverages the TE constraint policy language to express the BLP access rules
- Added mlsconstrain statement
  - mlsconstrain { dir file lnk_file } { read getattr execute }

```
(( l1 dom l2 ) or
  (( t1 == mlsfilereadtoclr )
    and ( h1 dom l2 )) or
  ( t1 == mlsfileread ) or
  ( t2 == mlstrustedobject ));
```

MCS in SE Linux

- Multiple Category Security
  - Attempts to use the MLS infrastructure to provide a more useable security mechanism for mainline RedHat distributions
- Use the sensitivity labels
  - But only allow a single clearance
  - Effectively assign sets of categories to subjects and objects
  - Using the sensitivity label in the security context

MCS in SE Linux

- While MCS uses the MLS mechanisms it is **not** a mandatory control
- Regular users can assign any category associated with them to a file they have access to
  - Regular users have the labeling discretion
- Functionally equivalent to ACLs, but stylistically different
  - May be easier to understand

Summary

- MAC is not the same as Bell LaPadula
- SE Linux offers several access control models
  - Type enforcement, MLS, and MCS
- SE Linux flexibility can be complex
  - Once the complex mandatory policy has been created and proven, the normal user cannot evade it
- More execution details for SELinux in upcoming class exercise.