Using Refinement in Formal Development of OS Security Policy Model

Ilya Shchepetkov
Ivannikov Institute for System Programming of the Russian Academy of Sciences
`shchepetkov@ispras.ru`

Tokyo, November 10, 2018

Security Policy Models

A security policy model is a high-level specification of security properties that a given system should possess, and of *security mechanisms* that enforce those properties

Examples of such properties:

- Users without a clearance must not get access to secret documents
- System files and processes must be protected from modification by regular users

Security Mechanisms

For strict guarantees of information isolation and simplification of administration tasks people usually use the following security mechanisms:

- Role-Based Access Control (**RBAC**)
  - NIST RBAC Model, 2004
- Mandatory Integrity Control (**MIC**)
  - The Biba Integrity Model, 1975
- Multi-level Security (**MLS**)
  - The Bell-LaPadula Policy Model, 1973

All of them have formal models establishing their strict semantics and security properties

Motivation

Most publicly available security models are outdated and not fully compatible with modern operating systems, and are not verified using formal methods. The task of combining them together is complex and prone to errors that can lead to vulnerabilities

MROSL DP-model

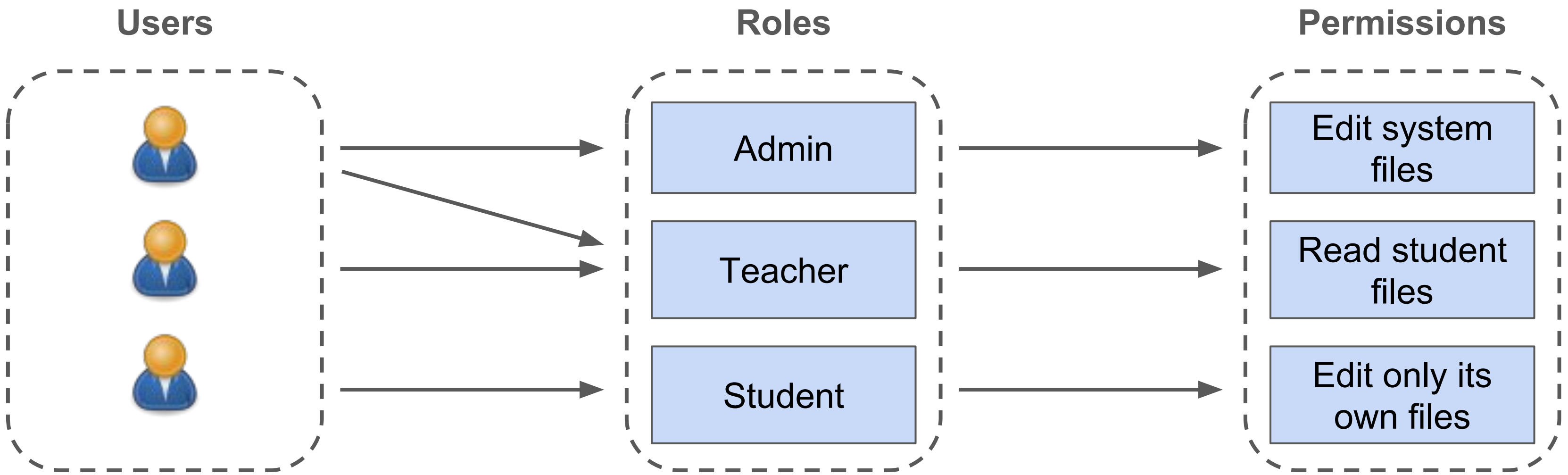
To overcome these issues a new security policy model called *the MROSL DP-model* has been developed. It is made specifically for Linux based operating systems and integrates several security mechanisms preserving their key security properties:

- **RBAC** with administrative and negative roles
- **MIC** with lattice of integrity levels
- **MLS** with information flows analysis

Our goal was to develop and verify a formal specification of the MROSL DP-model

Role-Based Access Control

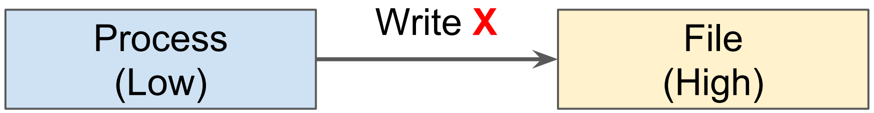
In RBAC, permissions are grouped into roles and are assigned to a user by an administrator or obtained though special administrative roles



Mandatory Integrity Control

In MIC, an integrity level is assigned to all users, processes and files that represents their level of trustworthiness

There are rules that restrict accesses based on these levels. For example, processes must not modify files or processes with higher integrity level (more trusted) even when executed by the root user or a user with root privileges



where `Low < High`

Multi-level Security

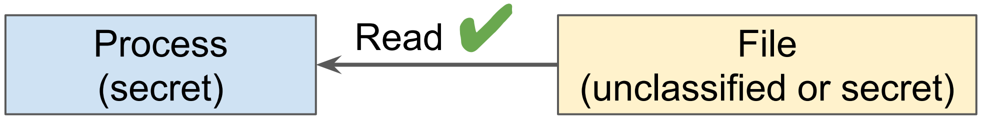
MLS was designed to deal with classified documents in military computer systems. MLS controls access according to the user’s clearance and the file’s classification

These classifications are divided into security levels. For example, the common government classifications are:

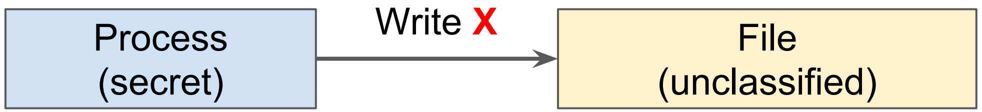
- unclassified
- confidential
- secret
- top-secret

Multi-level Security

Example: processes can read data from the file only if their clearance is more or equal to the classification level of the file



Processes must not write data to the file if their clearance is less than the classification level of the file



Why Event-B?

The MROSL DP-model uses set theory and predicate logic to define the state and the properties that the state must satisfy. It also contains several atomic state transition rules with pre- and post- conditions

This structure perfectly fits to Event-B. Besides, it is modern and well supported, and allows to perform manual proofs to assist automatic provers

State of the Specification

- Sets:
  - user accounts
  - entities (files, directories)
  - subjects (processes)
  - roles: administrative, ordinary, negative
- Functions:
  - integrity and security levels
  - current accesses and access rights (or permissions): to entities and roles
  - hierarchies of roles, entities and subjects
  - additional relations between elements of the model
  - various flags

Invariants of the Specification

- Type of variables
  - $SubjectAccesses \in Subjects \rightarrow (Entities \leftrightarrow Accesses)$
- Security invariants: just like the ones described on previous slides
  - $\forall s, e \cdot s \in Subjects \wedge e \mapsto WriteA \in SubjectAccesses(s) \Rightarrow EntityInt(e) \subseteq SubjectInt(s)$
- Consistency
  - No cycles in the filesystem (it should be a modelled as a tree)
  - Every entity has no more than one owner

Events of the Specification

- Create or delete entities, user accounts, subjects, roles
  - create or delete hard links for entities and roles
  - rename entities or roles
- Get or delete accesses, access rights to roles, entities
- Change security, integrity labels, various flags
- Additional events for analysis of information flows
  - if an entity **x** have write access to a subject **y**, which have write access to a subject **z**, then there can be an information flow from **x** to **z**

Some state transition rules were splitted into several events to simplify proving

Event-B Specification of the MROSL DP-model

First version of the specification (2014):

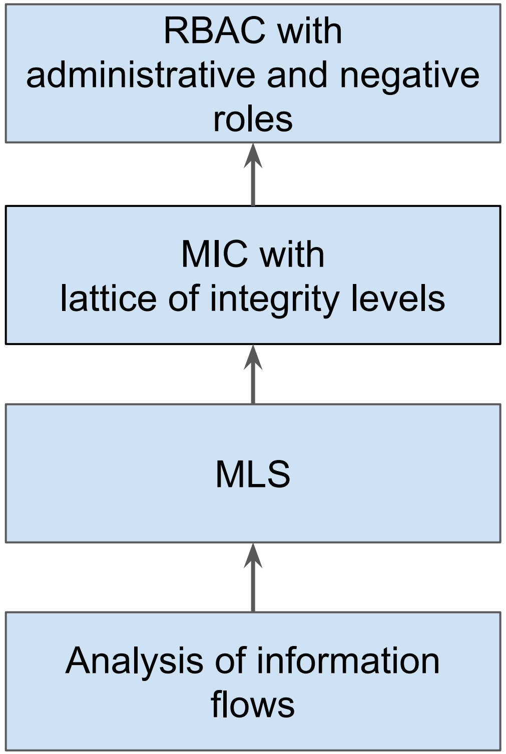
- monolithic, only one context and one machine;
- hard to maintain, extend and prove;
- 1700 lines of Event-B code based on 200 pages of text of the MROSL DP-model

Event-B Specification of the MROSL DP-model

Now:

- hierarchical, 4 main levels;
- still hard to prove, but far easier to understand and extend;
- about 5000 lines of Event-B code based on 300 pages of text of the hierarchical MROSL DP-model

Refinement Hierarchy


RBAC with administrative and negative roles
MIC with lattice of integrity levels
MLS
Analysis of information flows

Some Statistics and Results

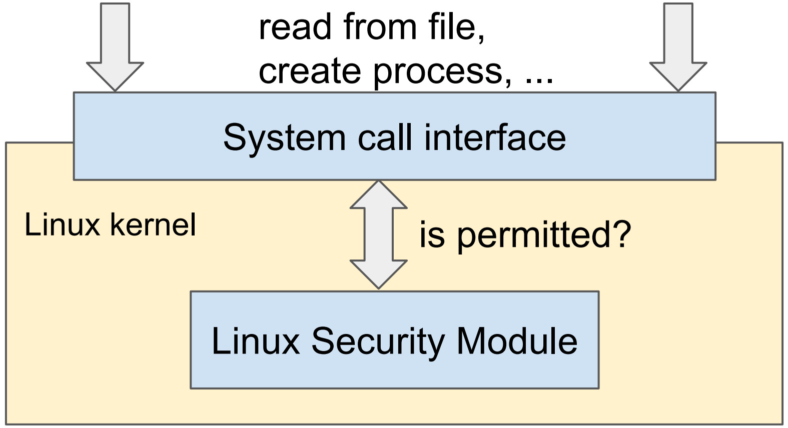
4 main levels:

- 65 variables
- 260 invariants
- 80 events
- ~ 3200 proof obligations
  - 75% are discharged automatically or semi-automatically (no more than a couple simplifying manual actions)
  - 25% require up to a month of interactive proving

The specification is complete and fully proved, and a number of issues in the MROSL DP-model are successfully found and fixed

Astra Linux

The MROSL DP-model is implemented in the certified Linux distribution called Astra Linux Special Edition as a Linux Security Module by RusBITech company



Encountered Issues with Event-B and Rodin

- Manual proofs often contain repeated steps that is not possible to automate;
- We have several big predicates that are duplicated in the guards of different events. It would be great to define them only once, like macro functions in C;
- The hierarchical MROSL DP-model uses multiple inheritance. Sometimes it even redefines things that are defined on the previous levels. Heavy workarounds are required to support both these features using refinement.

Links

- First level of the Event-B specification of the MROSL DP-model (with role-based access control):
  - https://github.com/17451k/base-model
- Our websites:
  - http://linuxtesting.org/
  - http://www.ispras.ru/en/
- (in russian): Book about formal development and verification of OS security policy models, 2018:
  - http://www.ispras.ru/publications/2018/security_policy_modeling_and_verification/