Runtime Verification of Linux Kernel Security Module

Denis Efremov
Ilya Shchepetkov

ISP RAS
{efremov, shchepetkov}@ispras.ru

OpenCERT 2019

Access Control in Linux

The core of Linux security features is Discretionary Access Control (**DAC**). In DAC, each object has an owner that has complete control over permissions to it. Owner can change permissions freely to its discretion and thus decides how his files can be shared

DAC is simple and quite intuitive, but:

- users can easily set insecure rights
- users cannot control if someone they share a file with will not further share its content (cannot control information flows)

Security Mechanisms

For strict guarantees of information isolation and simplification of administration tasks people usually use other 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

Existing security models

- Most publicly available security models are **outdated** and **not fully compatible** with modern operating systems
- They and are **not verified** using formal methods and may contain **vulnerabilities**
- The task of combining them together is complex and **prone to errors**

HIMACF Security Model

- It is made specifically for Linux based operating systems
- Integrates **RBAC**, **MIC** and **MLS**
- 300 pages of text
- Implemented in Astra Linux Special Edition

Structure of the HIMACF model

Like all security models, the HIMACF model is an abstract state machine with states and transitions

State:

- User accounts, subjects, entities, roles
- Hierarchies of roles, entities and subjects
- Current accesses and access rights
- Integrity and security levels
- Various flags
- Additional relations

State transitions:

- 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

Security properties are described as state invariants and preconditions of state transitions

Certification

Astra Linux is intended to be used by various government structures in Russia. In order to make it possible, it must be certified for compliance with a large number of security requirements, which are based on the Common Criteria standard (ISO/IEC 15408) and state:

- The security model shall be in a formal style
- The model shall provide a formal proof that the it cannot reach an unsecure state

Event-B

Event-B is a formal method for system-level modelling and analysis. Features:

- Easy to learn, based on set theory and first-order logic
- Refinement support to represent systems at different abstraction levels
- Good instrumental support (Rodin platform)
- Ability to use automatic and interactive provers, model checking to verify consistency and correctness of the specifications

Event-B specifications are discrete transition systems and consist of:

- Variables (state)
- Events (change state, transitions)
- Invariants (constrain variables)

Event-B Specification of the HIMACF model

Hierarchical, 4 refinement levels:

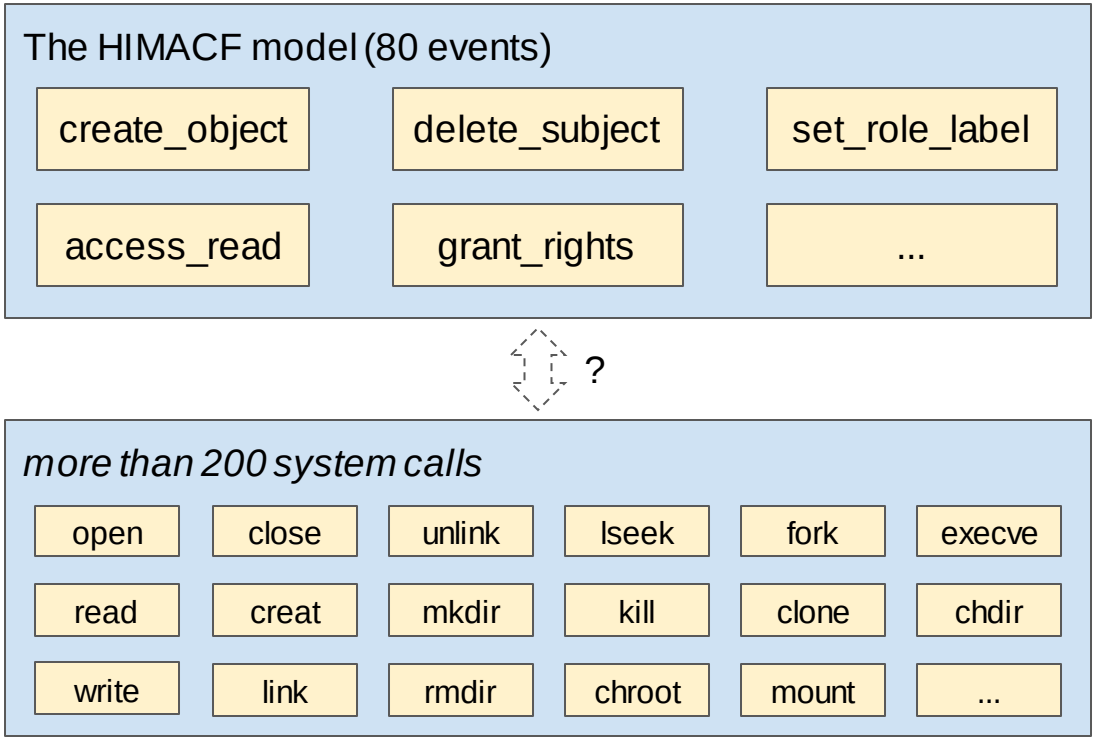
- 5000 lines of code
- 65 variables
- 260 invariants
- 80 events
- ~ 3200 proof obligations

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

Conformance of implementation to its specification

Access control mechanisms in Linux are implemented in the kernel, and users can interact with them only via system calls. We propose to intercept system calls to the kernel and automatically check that the results of their execution do not lead to accesses that are forbidden by the security model

System Call Interface

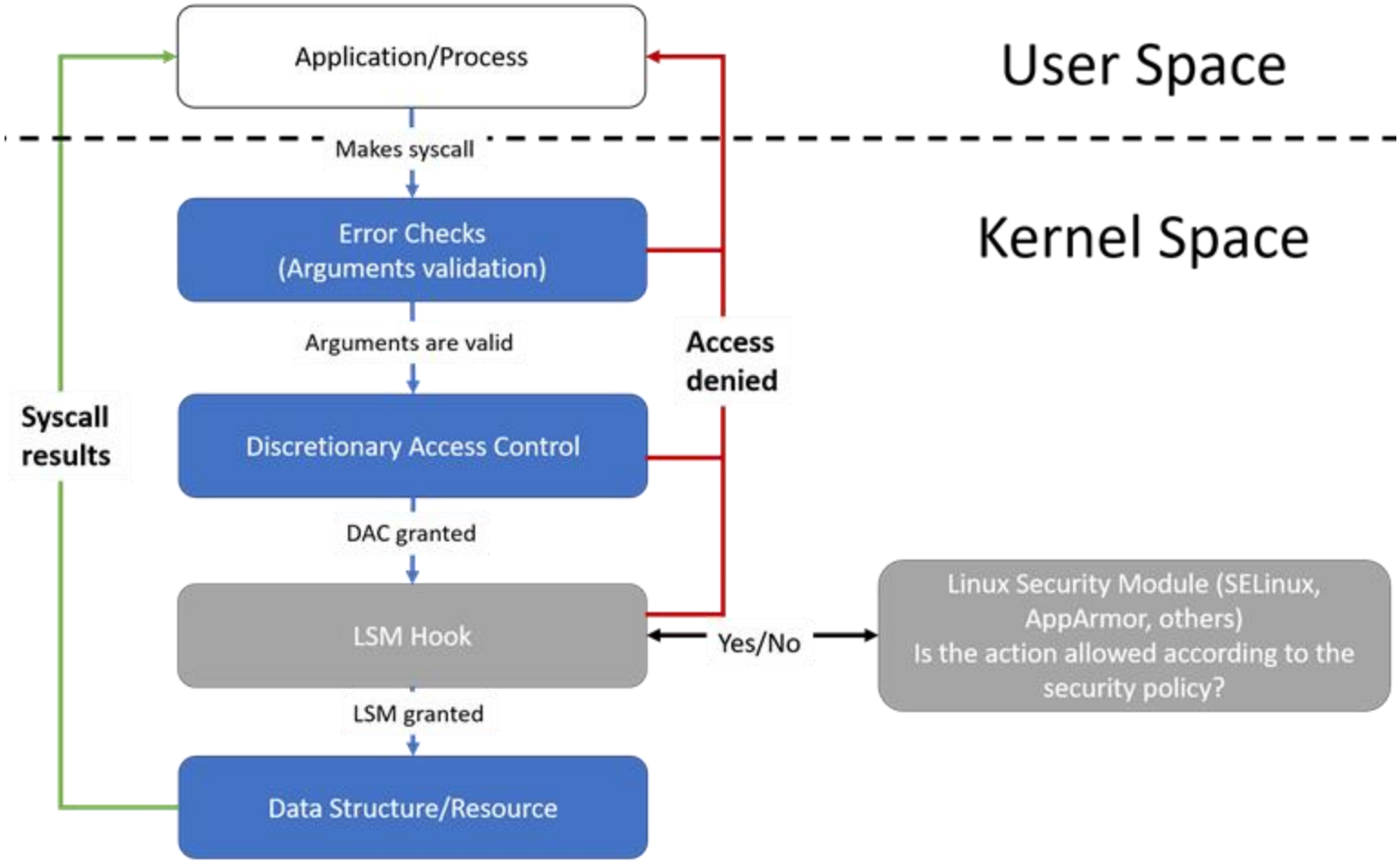


Event-B specification of the System Call Interface

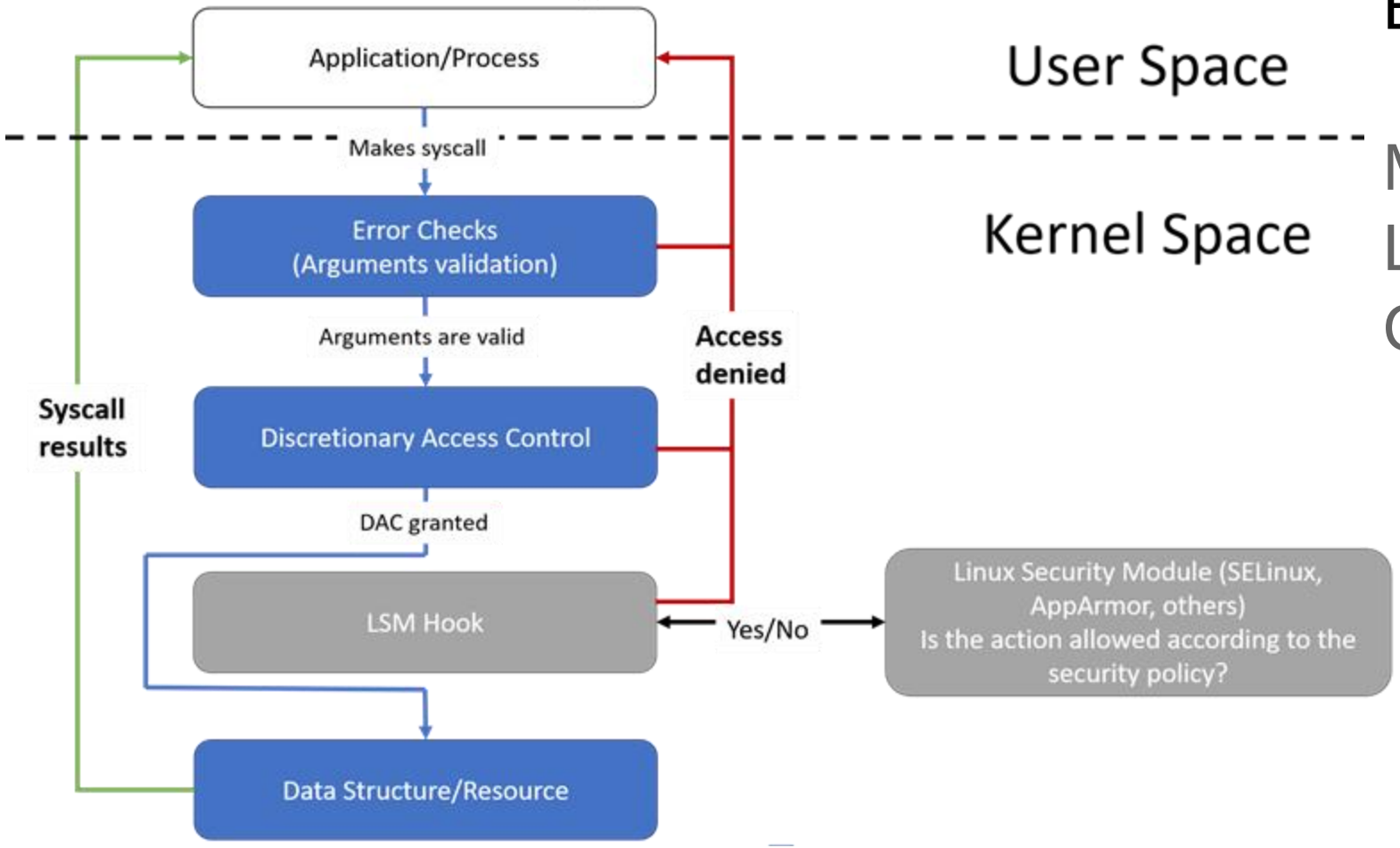
- We developed the Event-B specification of the System Call Interface as refinement of the Event-B specification of the HIMACF model
- We formalize system calls as **graphs of events** and use the special model variable that indicates which event should be executed next, depending on the current state of the specification and the event parameters
- We prove the refinement relation between such events and events of the Event-B specification of the HIMACF model
- Due to the use of refinement, this specification is correct by construction and fully conforms to the rules and events of the HIMACF model

Now, if we will show the conformance between **the specification of the system call interface** and **the Linux kernel**, then the desired conformance between **the HIMACF model** and **the Linux kernel** will be derived automatically

Linux Security Module

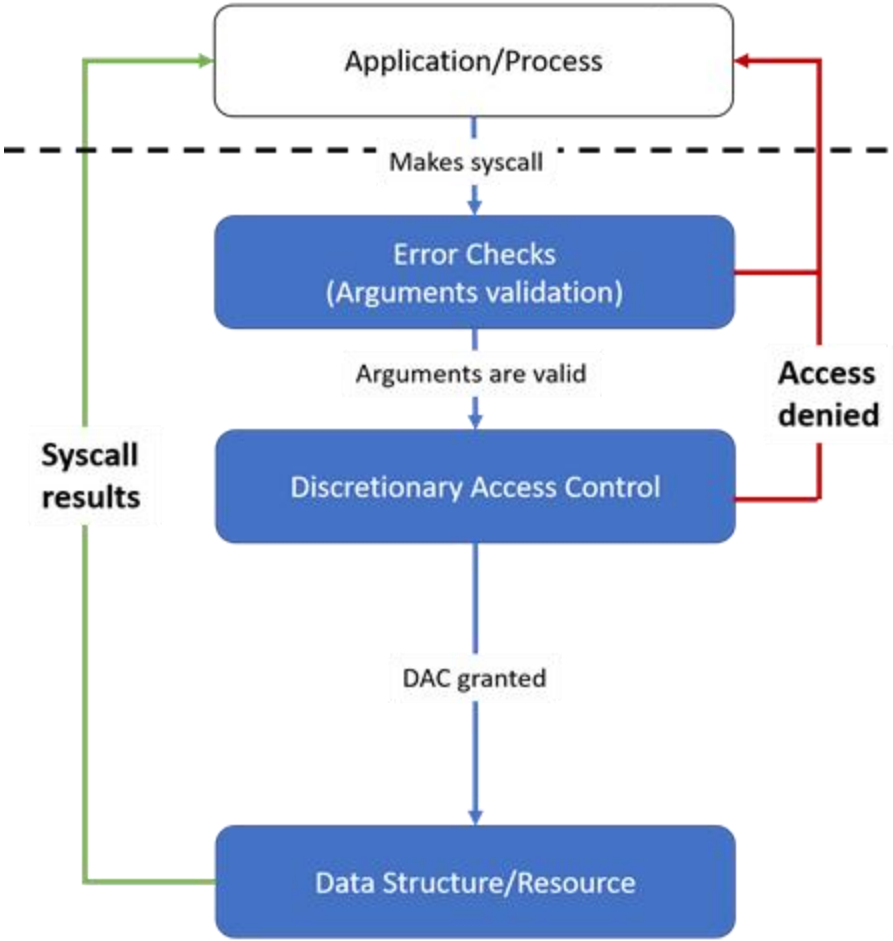


Errors (1)



Missing LSM Call

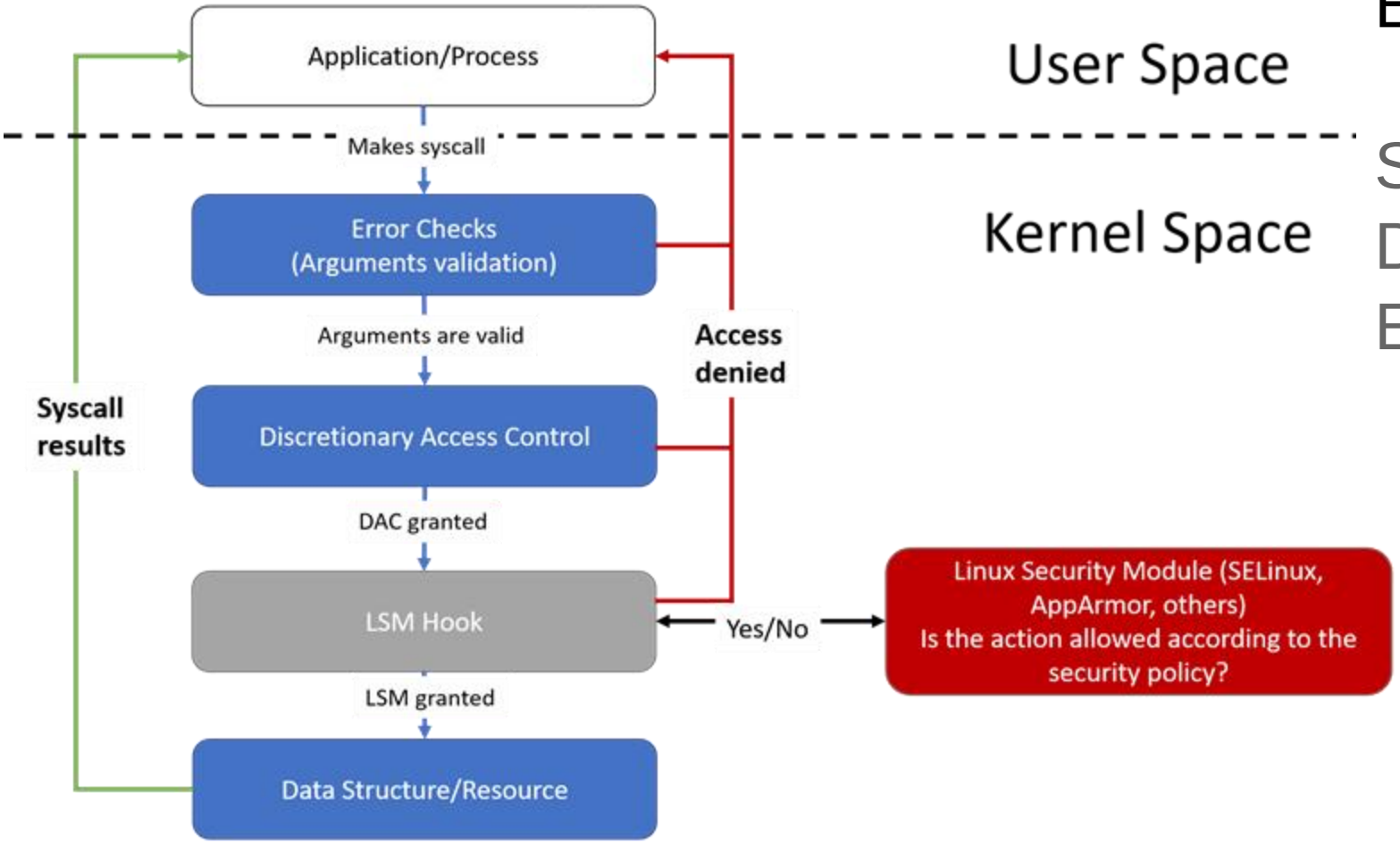
Errors (2)



LSM Incompleteness

Linux Security Module (SELinux, AppArmor, others)
Is the action allowed according to the security policy?

Errors (3)



Security Driver Error

Runtime verification

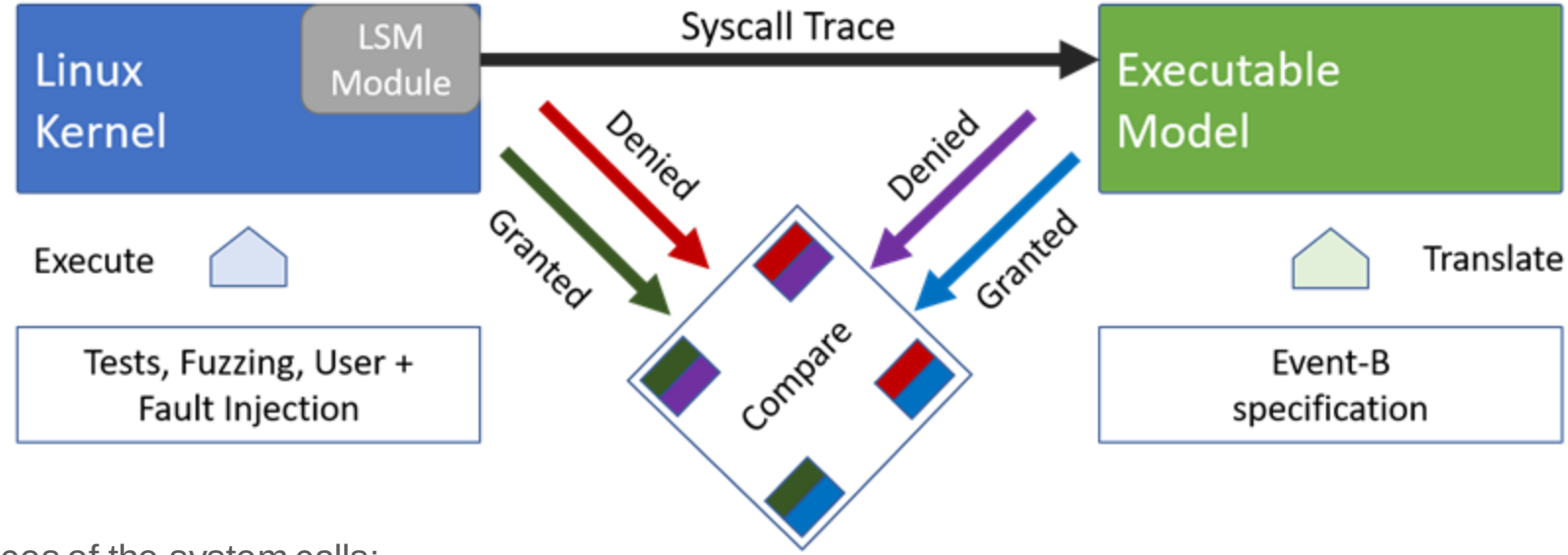
Challenges:

- Large codebase
- Frequent code updates (Linux, security module)

Check that the observable behavior of the operating system conforms to the security model:

- Trace system calls
- Check system calls on the Event-B specification
- Check real access results and results of specification events for equivalence
- Measure specification/code coverage

Replay of the real system behavior on the model



Traces of the system calls:

- Kernel + Security module
- Syzkaller for fuzzing
- Phoronix Test Suite, Spruce, ...
- GCOV for coverage
- SystemTap for monitoring

Replay and compare:

- Executable model: Event-B -> Python
- Replay on the executable model
- Errors: the model forbids an access, but the kernel grants

Compare Results

+System +Model

- Accept, Continue

-System +Model

- Check error code
- Revert, Continue

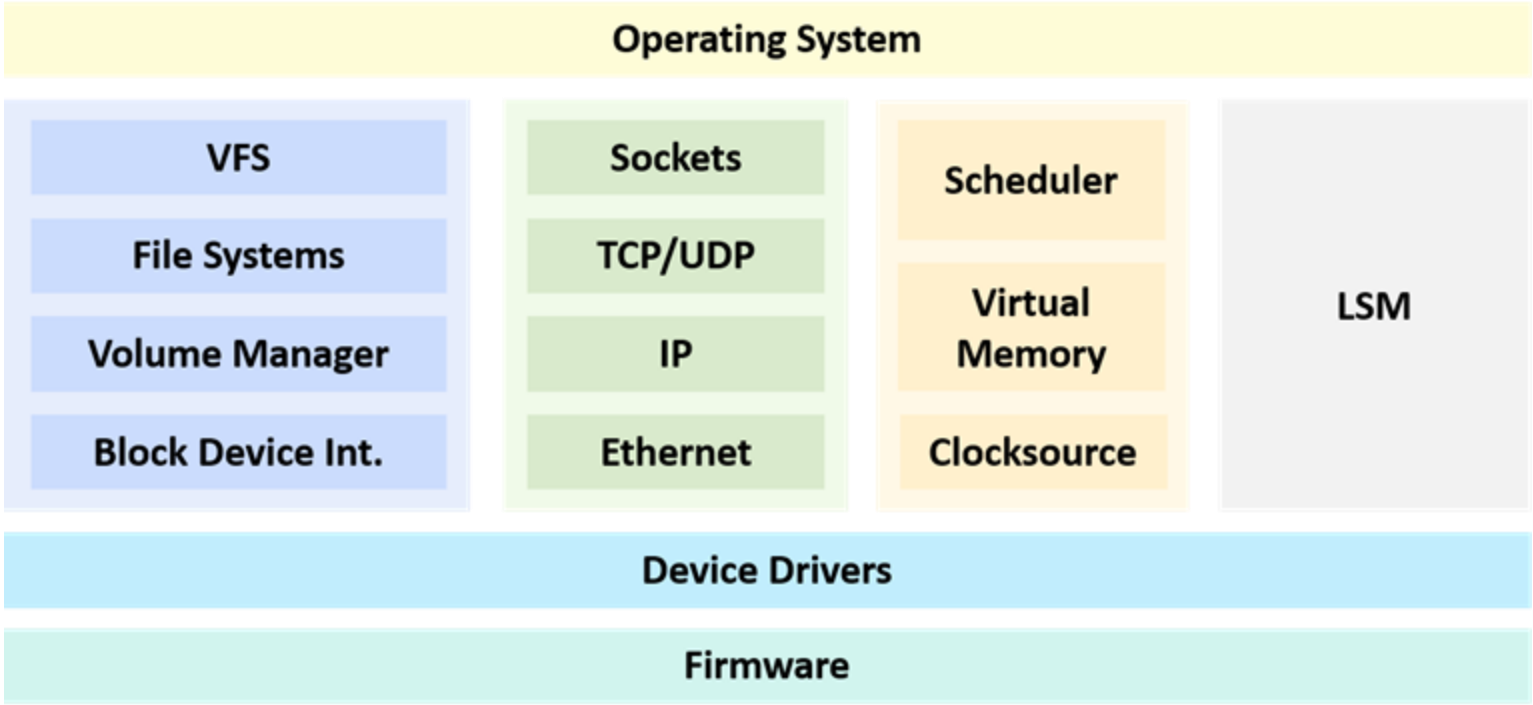
+System -Model

- Stop, Error

-System -Model

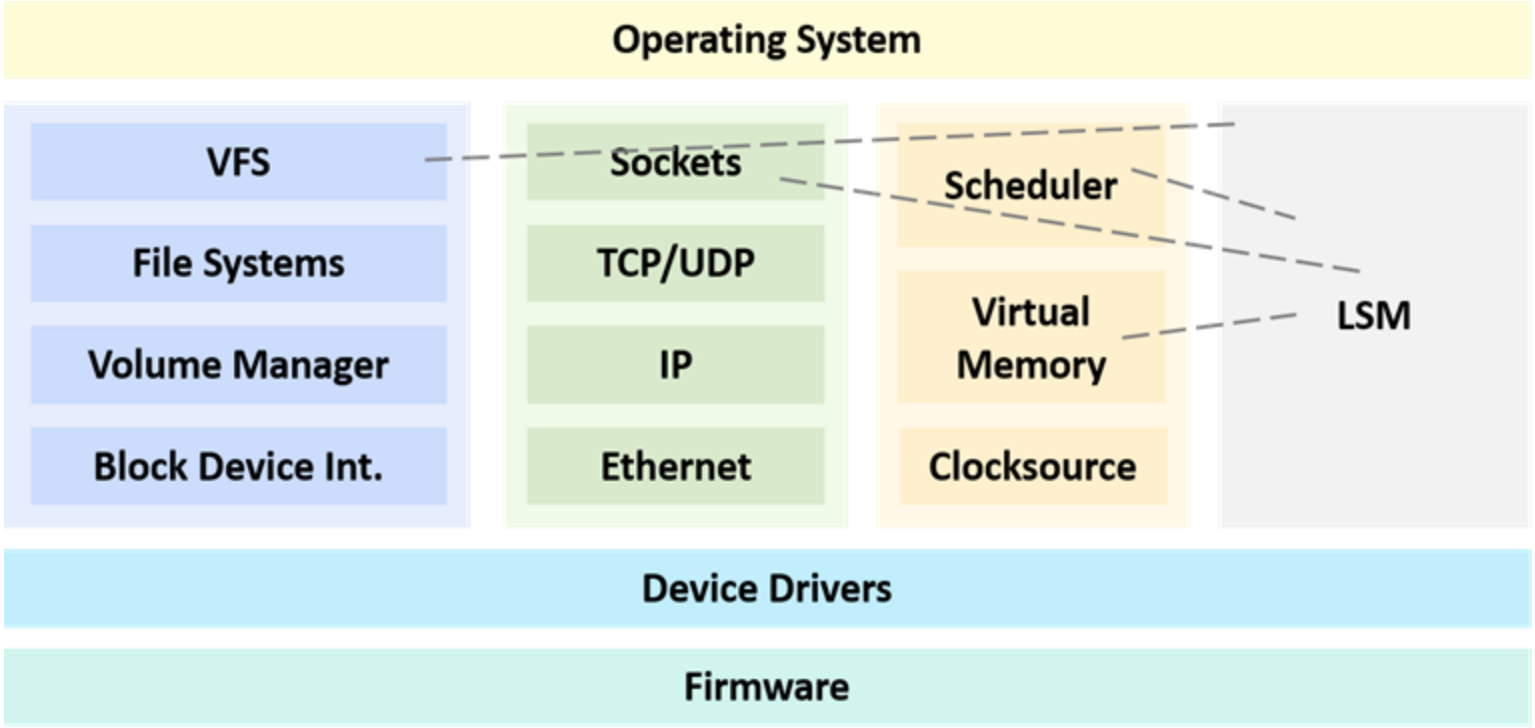
- Accept, Continue

Code Coverage



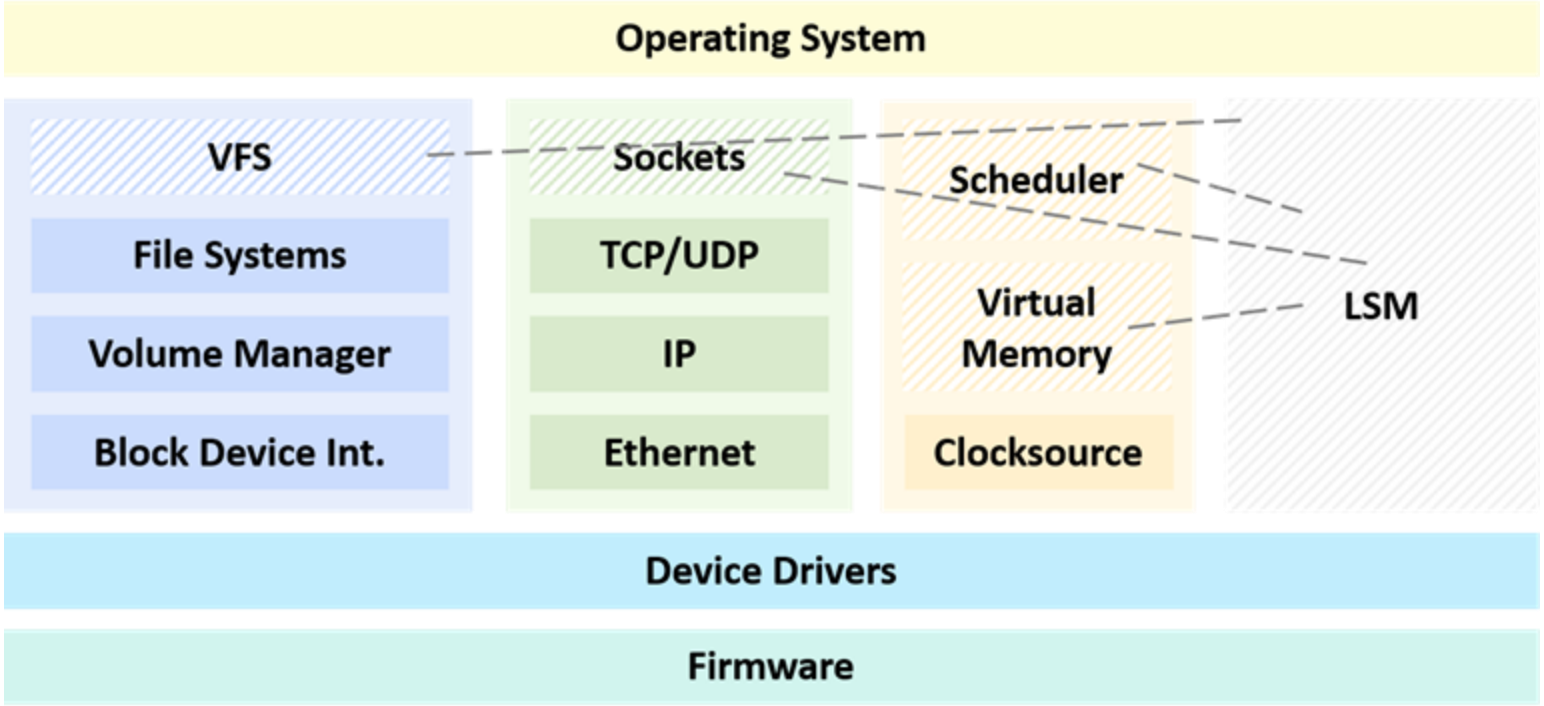
This image is based on Brendan Gregg's observability diagram. (CC BY-SA 4.0)

Code Coverage



Hooks covered

Code Coverage



Hooks covered

Lines covered

This image is based on Brendan Gregg's observability diagram. (CC BY-SA 4.0)

Specification Coverage

```
event create_user
  any user // parameter
      profile // parameter
      session // parameter
  where
    @grd1 user∈UserAccounts\CurrentUserAccounts
    @grd2 profile∈ℙ1(CurrentEntities)
    @grd3 session∈CurrentSessions
  then
    @act1 CurrentUserAccounts = CurrentUserAccounts ∪ {user}
    @act2 UserProfiles(user) = profile
end
```

Inputs

Pre-conditions

State update

Specification Coverage (Decision coverage)

```
event create_user
  any user // parameter
      profile // parameter
      session // parameter
  where
    @grd1 user∈UserAccounts\CurrentUserAccounts
    @grd2 profile∈ℙ1(CurrentEntities)
    @grd3 session∈CurrentSessions
  then
    @act1 CurrentUserAccounts = CurrentUserAccounts ∪ {user}
    @act2 UserProfiles(user) = profile
end
```

Inputs

Pre-conditions

State update

Results

- Base level of the Event-B specification of the HIMACF model

  https://github.com/17451k/base-model/blob/open/base-model.txt

- An example of the Event-B specification of the system call interface

  https://github.com/17451k/base-model/blob/open/base-model-with-open.txt

- Method for verification of the security module for conformance with its abstract specification
  - Specification was manually translated to an executable model
  - SystemTap for kernel tracing, GCOV for coverage, Spruce+Phoronix Test suites
- LSM interface incompleteness detected for inotify syscall

Thank you!
Questions?