Using Refinement in Formal Development of OS Security Model*

P. N. Devyanin[1], A. V. Khoroshilov[2], V. V. Kuliamin[2], A. K. Petrenko[2], and I. V. Shchepetkov[2]

[1] Educational and Methodical Community of Information Security, Moscow, Russia
peterdevyanin@hotmail.com
[2] Institute for System Programming, Russian Academy of Sciences, Moscow, Russia
{khoroshilov,kuliamin,petrenko,shchepetkov}@ispras.ru

Abstract. The paper presents work in progress on formal development of an operating system security model. The main goal is to measure our confidence in the quality of the model. Additional goal is to simplify its maintenance. We formalized and verified the model using formal method Event-B in two ways — with and without use of the refinement technique — in order to compare them and figure out what approach meets our goals better. Refinement is technique that helped us deal with complexity of the model, improve readability and simplify automatic proofs. However, deep understanding of the security model details and careful planning were absolutely necessary to achieve this. The approach without use of the refinement technique allowed to quickly start formalization and helped to study the details of the security model, but the resulting formal model became hard to maintain and explore.

Keywords: security model; formal verification; refinement; Event-B.

1 Introduction

Traditionally computer security models are expressed by combining a natural language and mathematical notations. Proofs of their correctness and consistency are performed by hand. Such proofs are error prone and not completely reliable. Besides, the reliability of manual proofs decreases with increasing size and complexity of the models.

Formal methods are promising approaches for specification and verification of software systems [10]. Validation of the RBAC ANSI 2012 standard confirmed that they could be successfully used for proving the correctness of security models [8]. Although this standard is widely used in industry, numerous problems were identified and fixed with the help of formal method B [1].

In this paper we present a work in progress on formal analysis of a mandatory entity-role security model of access and information flows control in Linux (the MROSL DP-model [7]), which provides:

* The research was supported by the Ministry of Education and Science of the Russian Federation (unique project identifier RFMEFI60414X0051).

– Mandatory integrity control (MIC).
– Mandatory access control (MAC).
– Role-based access control (RBAC).

The model was implemented in Astra Linux Special Edition[1] as a Linux Security Module by RPA RusBITech in 2014.

In the previous work [6] we chose formal method Event-B and the Rodin platform [2, 3] for the MROSL DP-model formalization and verification in order to confirm that the model satisfies the main security requirements – ability to protect entities and subjects from violating their integrity and security levels through authorized accesses.

Like many other formal methods, Event-B allows to develop formal specifications using refinement — a well-known and a widely recommended technique for incremental development [9, 4]. Numerous systems were formalized with it [5, 11]. Refinement helps to deal with the complexity of systems by decomposing their specifications into separate components. However, it is not clear what disadvantages refinement has, or what additional benefits it offers.

The paper presents the results of comparison of two MROSL DP-model specifications that were developed using different approaches: without use of the refinement technique and with it.

Next section of the paper describe the MROSL DP-model in detail. Sections 3 and 4 give an overview of Event-B and the refinement technique. Section 5 describes formalization of the model. Section 6 provides the comparison of developed specifications. Section 7 summarizes the results of development and verification and outlines further work of the project.

2 Main Features of the MROSL DP-model

Concepts used in the model are entities, sessions, user accounts and roles. Entities represent data objects like files, directories, sockets, etc. Sessions are operating system processes and each of them has the corresponding user account on behalf of which it operates. Roles are containers of permissions allowing to perform certain operations. The main complexity of the model derives from the non-trivial connections between these concepts that in most cases are expressed as functions .

The MROSL DP-model contains three main security features: MIC, MAC and RBAC. Due to MIC each entity, session, user account and role has an integrity level, which can be high or low. High-integrity entities (roles, etc.) are protected from modification by low-integrity sessions. MAC provides a security label to each entity, session, user account and role. MAC prohibits read accesses to an entity for sessions that do not have greater-or-equal security label (labels are partially ordered). Also it prohibits write accesses to an entity for sessions that do not have the same security label. RBAC strictly limits session rights to

[1] http://www.astra-linux.com

perform various operations by providing a set of roles (containers of rights) to each session.

The security model defines 44 operations. 34 of these operations can create or remove entities, sessions, user accounts or roles, change integrity and security labels, add or remove accesses. In addition, 10 operations are defined for more precise security analysis in terms of information flows. Each operation is defined by precondition and postcondition.

Postconditions must not violate any constraints defined in the model. Some of these constraints describe the environment of the security model (e.g. filesystem, processes, accesses), while others define security features mentioned above. Due to the large number of the constraints and their complex nature it is hard to verify the MROSL DP-model by hand without mistakes. Formalizing the model into a machine-readable form allows to prove its correctness in a machine checkable way.

3 Event-B and Rodin

Capability of combining automatic and interactive proofs along with the stories of its successful use for a number of complex projects convinced us to use a B dialect called Event-B. It has a simple notation and comes with a great tool support in the form of the Rodin Platform[2].

An Event-B model (or specification) consists of *contexts* and *machines*. Contexts contain the static parts of the specification: definitions of *constants* and their *axioms*. Machines contain the dynamic or behavioral parts of the specification: *variables*, *invariants* and *events*. Variables and constants can be sets, binary relations, functions, numbers or Boolean data. Values of variables form the current state of the specification and invariants constrain it.

Events represent the way the state may evolve. Each event consists of parameters, guards and actions, though any two of them are optional. Guards restrict values of parameters and states under which the event can occur, while actions change current state of the specification by modifying its variables. The correctness of each change of the specification state needs to be proven since invariants are supposed to hold whenever variable values change.

The Rodin platform supports both modelling and proving and integrates them in a seamless way. Both automatic and interactive proofs are supported. For each case that requires a proof — unambiguity of expressions, invariant preservation and refinement between models (if the refinement technique is used) — Rodin generates a corresponding proof obligation. Full proof of a model means that all generated proof obligations are discharged.

4 Refinement

Refinement technique is well known and supported by many formal methods. Instead of making a monolithic specification that contains all details of the sys-

[2] http://www.event-b.org

tem, refinement offers to build a series of specifications, where each specification is a refinement of the previous one (or abstract one) and the last one is the most concrete one. It allows to build a specification gradually, adding new features step by step, and thus to deal with complexities that arise during formalization of large systems.

There are two major approaches of refinement: *posit-and-prove* and *rule-based* [5]. In the first approach each refinement step must be proved to be correct, while in the second approach refinement is performed automatically using transformation rules, from which it follows that this refinement is correct by construction.

Refinement can be used in two ways. The aim of *horizontal* refinement (often called *superposition* refinement) is to add complexity to the specification by introducing new properties and events extending old ones. The second way is to use refinement to add details to data structures, such as replacing an abstract variable by a concrete one (*vertical* or *data* refinement).

5 Development

Formalization of the MROSL DP-model is a tricky task because of the size of the model (200 pages) and its complex nature. We started from the development of a monolithic specification that represents the entire security model. Then we used this knowledge to develop a new specification with the refinement technique.

To develop the monolithic specification, we used an iterative approach adding small parts of the system step by step and proving their correctness. It allowed to quickly notice and to fix problems occurring due to misunderstanding of the description of the system, but sometimes it was necessary to repeat existing proofs since every change of the specification could violate them. With this approach we developed the formal specification of the MROSL DP-model and proved its correctness. It took more than a year of work. A number of inaccuracies in the initial description of the model was identified and fixed.

The size of this specification is approximately 2 500 lines of code, which is quite small compared with the textual description. More details can be seen in Fig. 1.

	Number	Lines of code
Finite sets	7	7
Constants	26	26
Axioms	39	83
State variables	48	48
Invariants	161	394
Events	43	1954

Fig. 1. The composition and the size of the monolithic specification.

On top of this experience we developed a refined specification. Despite the fact that we were perfectly familiar with the MROSL DP-model at that moment, several times we had to completely rewrite the specification due to inaccurate planning. As a result we managed to decompose the original model into 16 parts and to determine the order in which they should be implemented as a series of specifications. Our refinement was correct by construction, so no additional proof obligations were generated.

6 Comparison

During the analysis of the refined specification we noticed some differences from the monolithic one. The refined version is easier to understand, more human readable and structured. Moreover, refinement provides a natural way to add new details to the specification. It is easier to change the specification, and, unlike the monolithic specification, changes can affect fewer of discharged proofs. We kept possible changes in mind during refinement planning, and it helped to achieve our additional goal – to simplify maintenance.

Despite the aforementioned advantages of the refined specification, most design decisions (which data structures to use, how to express invariants better, etc.) are the same. The main comparison characteristics like the size of the specification and the difficulty of interactive proofs are also similar in our case. However, the development of the monolithic specification required much less effort — due to the absence of refinement planning, which is quite a difficult task, so our main goal (to prove correctness of the model) is achieved faster with it.

Better readability of the refined model can be exposed with two examples[3]. Fig. 2 demonstrates a part of the `create_session` event from the monolithic specification. This event is an analogue of the corresponding operation from the MROSL DP-model. It models the creation of a new session. It has parameters, preconditions (guards, in terms of Event-B) and postconditions (actions). Guards and actions are lines with `@grd` and `@act` labels, parameters are defined in the `any` section.

Arrows in Fig. 2 group related guards and actions by parts of the model to which they belong. It is easy to notice a lack of structure here: a lot of closely related guards and actions are scattered throughout the text. For instance, `@grd2` — `@grd4`, `@grd7` — `@grd9`, `@grd25`, `@act6`, `@act7` are represent profiles and executable files. Moreover, parts of one event can be associated not only with each other, but also with the existing invariants and parts of other events. There is no way to denote their connection in Event-B without use of refinement.

Actually the refined model is a series of specifications. Each of them describes a specific feature of the MROSL DP-model. Fig. 3 represents one of the refinement steps — adding the specification modeling profiles and executable files. We can see how a particular feature — profiles and exefiles — affects existing

[3] These examples are provided with permission of RPA RusBITech.

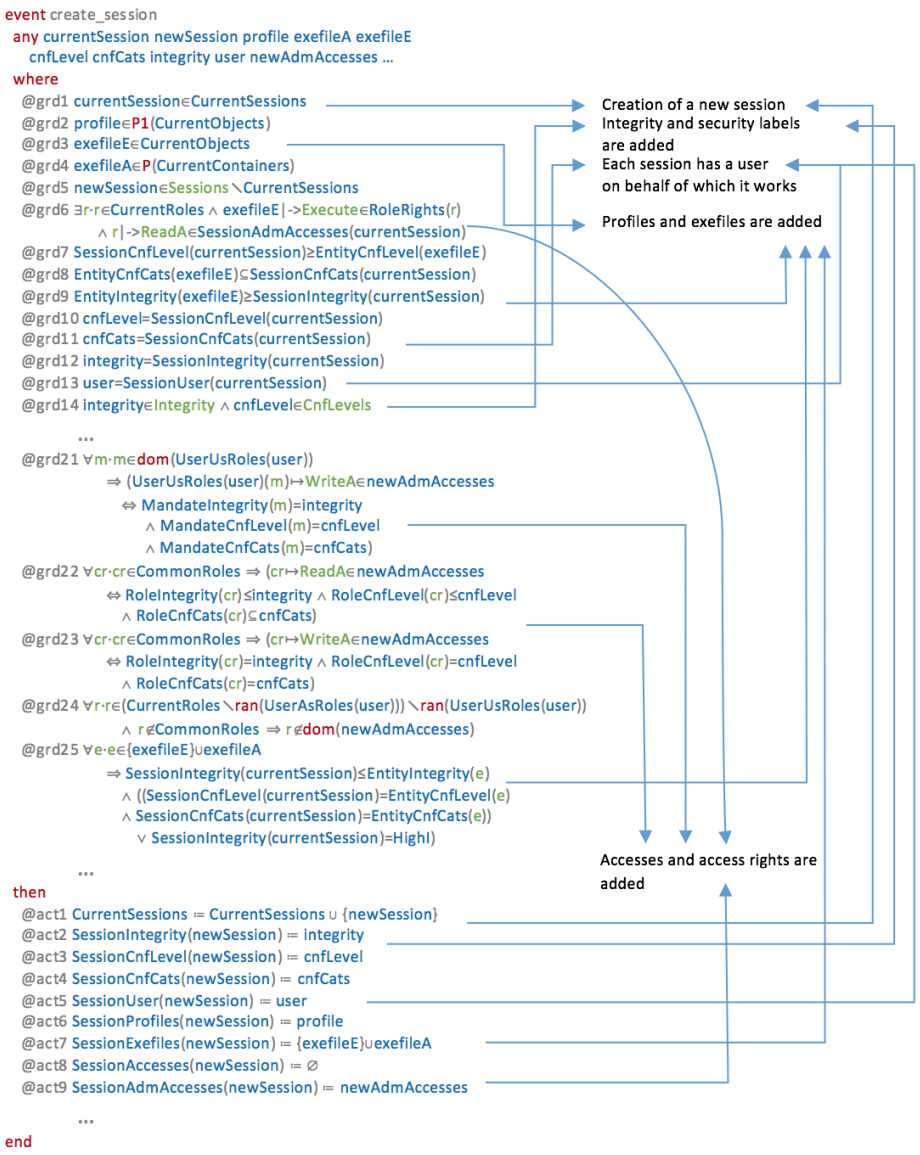

Fig. 2. A part of the `create_session` event from the monolithic specification.

events (`create_user`, `set_user_labels` and `create_session`) and the requirements (invariants) it must satisfy. This specification is quite compact, it is easy to understand and maintain.

To achieve this it is vital to know possibilities of refinement, its restrictions, features of the chosen formal method, and of course the system itself. Each wrong

```
machine N11 refines N10 sees C3

...
invariants
  @UserProfiles_type UserProfiles∈CurrentUserAccounts→P1(CurrentEntities)
  @SessionProfiles_type SessionProfiles∈CurrentSessions→P(Entities)
  @SessionExefiles_type SessionExefiles∈CurrentSessions→P(Entities)
  @UserProfilesIntIsCorrect
    ∀u,p·u∈CurrentUserAccounts ∧ p∈UserProfiles(u) ⇒ UserIntegrity(u)=EntityIntegrity(p)
  @UserProfilesCnfIsCorrect
    ∀u,p·u∈CurrentUserAccounts ∧ p∈UserProfiles(u) ⇒ UserCnfLevel(u)=EntityCnfLevel(p)

...
events

...
event create_user extends create_user
 any profiles
 where
  @grd33 profiles∈P1(CurrentEntities)
  @grd34 ∀p·p∈profiles ⇒ EntityIntegrity(p)=integrity
  @grd35 ∀p·p∈profiles ⇒ EntityCnfLevel(p)=cnfLevel
 then
  @act16 UserProfiles(user) ≔ profiles
end

event set_user_labels extends set_user_labels
 any profiles
 where
  @grd58 profiles∈P1(CurrentEntities)
  @grd59 ∀p·p∈profiles ⇒ EntityIntegrity(p)=integrity
  @grd60 ∀p·p∈profiles ⇒ EntityCnfLevel(p)=cnfLevel
 then
  @act16 UserProfiles(user) ≔ profiles
end

...
event create_session extends create_session
 any exefileA exefileE profile
 where
  @grd12 profile∈P1(CurrentObjects)
  @grd13 exefileE∈CurrentObjects
  @grd14 exefileA∈P(CurrentContainers)
  @grd15 EntityCnfLevel(exefileE)⊆SessionCnfLevel(currentSession)
  @grd16 EntityCnfLevel(exefileE)⊆UserCnfLevel(user)
  @grd17 EntityIntLevel(exefileE)≥integrity
then
  @act7 SessionProfiles(newSession) ≔ profile
  @act8 SessionExefiles(newSession) ≔ {exefileE}∪exefileA
end
end
```

Fig. 3. A part of the refined specification related to profiles and executable files.

decision can eventually lead to the need of total redesign of a specification. In some cases wrong refinement can even complicate formalization and analysis.

In our experience, automatic provers operate a little better on the refined specification. Some statistics can be found in Fig. 4. It could be explained by the fact that the Rodin platform was specifically designed to support the refinement technique.

Specifications	Monolithic	Refined
Automatically proved	70% of the model	75% of the model

Fig. 4. Statistics of operation of automatic provers.

The MROSL DP-model contains precise description of operations in form of preconditions and postconditions. It simplifies the development of both specifications, but also restricts the refinement possibilities. Thus, in our case (and for the class of similar systems) refinement offers only better readability and structure of specifications, in exchange for increased complexity of development.

7 Conclusions and Future Work

We completed formalization and verification of the MROSL DP-model using formal method Event-B in two ways — without use of the refinement technique and with it. The approach without refinement allowed us to develop a monolithic specification that contains the entire security model, while refinement was used to build an ordered series of specifications, where each specification is a refinement of the previous one (or abstract one) and the last one is the most concrete one.

We have found that refinement improves readability and simplifies automatic proofs. But to achieve this it is vital to know possibilities of refinement, its restrictions, features of the chosen formal method, and the system under verification (the MROSL DP-model, in our case). Careful planning is also needed. The development of the monolithic specification required much less effort, but the resulting specification is hard to maintain and explore.

On the next steps of the project we are going to use refinement for formalization and verification of a hierarchical MROSL DP-model, which is under development. It will contain a basic layer that can be extended to a security model of hypervisor, operating system, etc. We believe that there will be difficulties with refinement due to a nonlinear hierarchy of the model and multiple inheritance.

References

1. Abrial, J.-R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (1996)
2. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
3. Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: An open toolset for modelling and reasoning in Event-B. International Journal on Software Tools for Technology Transfer 12(6), 447–466 (2010)
4. Abrial, J.-R., Hallerstede, S.: Refinement, decomposition, and instantiation of discrete models: Application to Event-B. Fundamentae Informatica 77(1,2), 1–28 (2007)
5. Damchoom, K.: An incremental refinement approach to a development of a flash-based file system in Event-B. Ph.D. thesis, University of Southampton, School of Electronics and Computer Science (2010)
6. Devyanin, P., Khoroshilov, A., Kuliamin, V., Petrenko, A., Shchepetkov, I.: Formal verification of OS security model with Alloy and Event-B. In: Abstract State Machines, Alloy, B, TLA, VDM, and Z. pp. 309–313 (2014)
7. Devyanin, P.N.: Security models of computer systems: access control and information flows (in Russian). Hot line - Telecom (2013)
8. Huynh, N., Frappier, M., Mammar, A., Laleau, R., Desharnais, J.: Validating the RBAC ANSI 2012 standard using B. In: Abstract State Machines, Alloy, B, TLA, VDM, and Z. pp. 255–270 (2014)
9. Wirth, N.: Program development by stepwise refinement. CACM: Communications of the ACM 14 (1971)
10. Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.: Formal methods: Practice and experience. ACM Computing Surveys 41(4), 1–39 (2009)
11. Yeganefard, S., Butler, M., Rezazadeh, A.: Evaluation of a guideline by formal modelling of cruise control system in Event-B. In: NFM 2010. pp. 182–191 (2010)