Formal Verification of OS Security Model with Alloy and Event-B

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
peter_devyanin@hotmail.com
[2] Institute for System Programming, Russian Academy of Sciences, Moscow, Russia
{khoroshilov,kuliamin,petrenko,shchepetkov}@ispras.ru

Abstract. The paper presents a work-in-progress on formal verification of operating system security model, which integrates control of confidentiality and integrity levels with role-based access control. The main goal is to formalize completely the security model and to prove its consistency and conformance to basic correctness requirements concerning keeping levels of integrity and confidentiality. Additional goal is to perform data flow analysis of the model to check whether it can preserve security in the face of certain attacks. Alloy and Event-B were used for formalization and verification of the model. Alloy was applied to provide quick constraint-based checking and uncover various issues concerning inconsistency or incompleteness of the model. Event-B was applied for full-scale deductive verification. Both tools worked well on first steps of model development, while after certain complexity was reached Alloy began to demonstrate some scalability issues.

1 Introduction

Complexity of practical security models used in the industrial and government systems makes their analysis a hard work. A well-known, but not well adopted yet, approach to decrease the effort needed for such an analysis is usage of formal modeling with a variety of features supported by modern tools like model checking, constraint checking, deductive verification, etc.

In this paper we present a work-in-progress on formal analysis of mandatory entity-role model security of access control and information flows of Linux-based operating system (MROSL DP) [1], which includes lattice-based mandatory access control (MAC), mandatory integrity control (MIC), and role-based access control (RBAC) mechanisms [2] and is intended to be implemented inside Linux as a specific Linux Security Module (LSM) [3]. The analysis performed is to check model consistency and conformance to basic correctness requirements - ability to prevent break of integrity and confidentiality levels of data and processes. In addition a kind of data-flow analysis is executed to prove that the model is able to preserve security in the face of certain attacks, namely, to keep

high-integrity data and processes untouched even if an attacker gets full control over some low-integrity processes.

Alloy [4] and Event-B [5] (Rodin [6]) are used for formalization of the model and for its analysis and verification. Alloy was applied to provide constraint-based checking of operation contracts and to uncover various issues concerning inconsistency or incompleteness of the model. Event-B was applied for full-scale deductive verification of the model correctness.

Further sections of the paper describe some details of the security model analyzed, provide the statistics of tool usage, summarize the results obtained, and depict further development of the project.

2 Main Features of the Model

The model under analysis MROSL DP [1] describes security mechanisms of Linux-based operating system in terms of user accounts (representing users), entities (representing data objects under control - files, directories, sockets, shared memory pieces, etc.), sessions (representing processes working with data), and roles (representing arrays of rights on usage or modification of data objects).

Each session has the corresponding user account on behalf of which it works. A session has a set of roles assigned to it and a set of accesses to various entities (which it reads or writes), both these sets can be modified. Entities can be grouped into other entities (containers) and form a Unix-like filesystem (with containers-directories and hard links making possible for an entity to belong to several containers). Roles also form a filesystem-like structure, where roles-containers are used to group the rights of all the included roles.

The main security mechanisms presented in the model are the following.

- *RBAC.* Each operation performed by a session should be empowered by a corresponding right included in some of the current roles of the session.
- *MIC.* Each entity, session, or role has integrity level - high or low. Modification of high-integrity entities or roles by low-integrity sessions or through low-integrity roles is prohibited.
- *MAC.* Each entity, session, or role has security label. Security is described by two attributes - a level of ordered confidentiality (unclassified, confidential, etc.) and a set of unordered categories (e.g., whether the corresponding information concerns financial department or research department). Security labels are partially ordered according to order of levels and inclusion of category sets. Read access to an entity is possible only for sessions (or through roles) having greater-or-equal security labels. Write access is possible only for sessions (or through roles) having exactly the same security labels.

The model defines 34 operations in form of contracts - preconditions and postconditions. Operations include actions on creation, reading, or modification of user accounts, sessions, entities, roles, rights of roles, accesses of a session. For the purpose of data-flow analysis of attacks additional 10 operations are defined,

which describes control capture of a session and information flows by memory or time between sessions and entities.

The model also includes a variety of specific details, like shared containers, container clearance required (CCR) attributes of containers (integrity or confidentiality ignorance flags), write-only entities (like /dev/null), administrative roles used to operate with other roles, and so on. Space restrictions prevent us from discussing them here, but these details are responsible for a large part of model's complexity.

3 Formalization and Verification Process and its Results

Alloy and Event-B are used for model formalization. Both formalizations are close and consist of type definitions for basic model concepts (user accounts, sessions, entities, roles, rights, accesses), state data structure, basic invariants representing well-definedness of data objects (wd-invariants), invariants representing correctness of a state or conformance to basic requirements presented in the list in the previous section (c-invariants), and operation contracts.

For now data-flow-related constraints and operations are not yet recorded in the tool-supported formal models, so the further information is provided for incomplete models.

Some statistics on models elements and size is presented in the Table 1. Lines of code numbers shown are rounded to tens.

	Alloy model		Event-B model	
	Number	Lines of code	Number	Lines of code
Type definitions	20	100	9	1
State variables	25	40	43	1
wd-invariants	17	300	76	200
c-invariants	31	230	31	120
Operation contracts	32	2200	32	1300
Total		2900		1650

Table 1. Models' composition and size.

Let $I(s)$ denote that all invariants hold in the state s, for each operation one have precondition $pre(s)$ and postcondition $post(s, s')$, the latter depending on pre-call state s and post-call state s'. We try to ensure that the statement $I(s) \wedge pre(s) \wedge post(s, s') \Rightarrow I(s')$ holds for all operations specified. Alloy is used for quick finding omissions of necessary constraints in operation contracts or invariants by means of constraint checking, which can find small counterexamples for wrong statements. Several omissions and inconsistencies were found in the original model with the help of Alloy. Rodin with corresponding plugins was used to prove formally all the same statements in Event-B model. Rodin generates from 30 to 90 assertions for an operation, from which up to 25% require human intervention to prove. In total proof of about 15% of generated assertions need human aid.

When the number of invariants in the model exceeded 20, Alloy stopped to generate counterexamples for wrong statements. Sometimes we met this problem before, but then it can be resolved by increasing the number of top-level objects. Since more than 20 invariants are used, this doesn't help or just causes memory overflow. We haven't managed to use ProB [7] tool for constraint checking, mostly due to model complexity.

The following issues of Event-B make the work harder: lack of possibility to use auxiliary predicates in contracts, and lack of possibility to introduce some abbreviations, which can help to ease the notation rigidity, for example, to use more suitable notation for second and third elements of triples. This can be implemented with lambda-expressions, but they hinder automated proofs. Additional problem for possible work splitting is provided by the structure of proof log stored. Since it is represented as XML, its size in our example is about 200 MB, and usually a little change in the proof of some assertion strangely result in many changes in the XML-file stored. So it is hard to split the proof elaboration between several persons, since merging significantly changed big files is hard.

4 Conclusion and Future Work

The presented in the paper work on formalization and verification of the security model of Linux-like OS is not finished yet, but the results obtained already can be used to make some conclusions. First, the project as usual shows that formalization of requirements helps to uncover some bugs that can become more serious on the implementation phase. Second, the project demonstrates that sometimes tools supporting formal analysis of models are not scalable enough for industrial use, but other suitable tools can be usually found.

On the next steps of the project we are going to finalize model formalization and verification with the help of Event-B. Then, the security mechanisms modeled are to be implemented as an LSM and the implementation code is to be verified on conformance to the model with the help of Why [8] platform.

References

1. P. N. Devyanin. The models of security of computer systems: access control and information flows. Hot line - Telecom, Moscow, 2013 (in Russian).
2. M. Bishop. Computer security: art and science. Pearson Education Inc., Boston, 2002.
3. C. Wright, C. Cowan, S. Smalley, J. Morris, G. Kroah-Hartman. Linux Security Modules: General Security Support for the Linux Kernel. Proc. of the 11-th USENIX Security Symposium, pp. 17-31, 2002.
4. D. Jackson. Software Abstractions: Logic, Language, and Analysis. MIT Press, 2006.
5. J.-R. Abrial. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010.
6. J.-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, L. Voisin. Rodin: An Open Toolset for Modelling and Reasoning in Event-B. International Journal on Software Tools for Technology Transfer, 12(6):447-466, 2010.

7. M. Leuschel, M. Butler. ProB: A Model Checker for B. Proc. of FME 2003, LNCS 2805:855-874, Springer, 2003.
8. F. Bobot, J.-C. Filliâtre, C. Marché, A. Paskevich. Why3: Shepherd Your Herd of Provers. Proc. of Boogie 2011: 1-st Intl Workshop on Intermediate Verification Languages, pp. 53-64, 2011.