Special Features of TLA+ Temporal Logic of Actions for Verifying Access Control Policies

1st Andrey M. Kanner
Department of Cryptology and Cyber Security (42)
National Research Nuclear University MEPhI
Moscow, Russia
kanner@mail.ru

2nd Tatiana M. Kanner
Department of Cryptology and Cyber Security (42)
National Research Nuclear University MEPhI
Moscow, Russia
sheikot@mail.ru

***Abstract*—The paper considers special features of applying Lamport's temporal logic of actions when verifying access control policies for arbitrary data protection tools. It justifies the necessity of implementing verification in the process of development and certification of various software tools and algorithms, in particular, policies for controlling subjects' access to objects. It contains a general structure of notation or specification of the system being studied in a formal language suitable for verification, and its particular version in the TLA+ language. The paper considers special features of using Lamport's temporal logic of actions, and gives recommendations regarding dos and don'ts when initializing the modeled system, when forming and using invariants or temporal properties, history and auxiliary variables, safety and liveness properties, as well as when accounting for the termination of the verification process. Such features and recommendations are formulated in a quite universal way and do not depend on the applied verification approach and on the system being studied. The paper lists typical errors that may be done during verification, which make its results useless, while artificially creating a feeling of confidence in the system's "rightness", "correctness" or "security / safety". The conclusion presents key features that can have a significant impact on verification results, as well as on feasibility of its implementation. It proposes one of the possible directions for further research on the development of a general approach to substantiating conformity of the verified system specification in some formal language with its practical implementation.**

***Index Terms*—verification, temporal logic of actions, auxiliary variables, history variables, invariants, temporal properties, safety properties, liveness properties**

I. Introduction

Verification of access control policies for arbitrary tools of data protection against unauthorized access is an extremely useful step in their development or certification.

Within this paper, verification means not just testing and practical confirmation of any properties in an arbitrary system in the course of experimental research, but formal justification for the implementation of such features in all states of the system, which is rather difficult to do manually. To perform the verification, special tools are used that allow to automatically check the truth of formal properties for the algorithms implemented in the modeled system, depending on various conditions and at various time intervals (in the past, present and future).

There are various approaches to modeling and verification and the relevant tools [1]–[3]. However, regardless of the chosen approach, the system or the algorithm being studied must be first presented in the form of a notation or specification in some formal language (often using logical predicates), which describes:

1) Initial state of the system or algorithm (initialization).
2) Variables of the model or algorithm – the entities that can change in the course of work.
3) Operating rules for the system or algorithm (possible states and values of model variables, rules for transition from one state to another – for example, when subjects access objects).

The main advantage of verification over testing or experimental research is that it becomes possible to quickly find errors in the design or implementation of the modeled system. When testing, there is a high probability of missing some hidden corner cases. Verification tools automatically analyze any possible sequence of states, checking the necessary formal properties of the system in each of them. The ability to quickly detect design or implementation errors is especially important for data protection tools that would enhance, rather than weaken, security of the system where they are applied.

In accordance with the international regulatory documents [4] and, for example, some current regulatory documents of the Russian Federation, verification is a necessary development stage only for some classes of data protection tools. Nevertheless, it is desirable to perform verification for any data protection tools, regardless of the security class of the systems where they are applied (and, possibly, for all software tools and algorithms used). This is due to the fact that only verification makes it possible to assert correctness of certain algorithms or talk about execution of any formal properties in the system, including its security (from the point of view of the applied formal security model).

This paper summarizes and details some of the features of access control policy verification encountered by the authors when using one of the approaches to verification – Lamport's temporal logic of actions (TLA) and the *Model Checking* [5], [6] in relation to access control tools described in [7], [8][1]. Further material in the paper refers to formal notations in the

[1]The full text of one of the developed TLA+ specification is available on the author's website https://github.com/kanner/ipes-model

$$
\begin{aligned}
Init \triangleq\ & \wedge S\_active = \{s\_0\} \\
& \text{only } s\_0 \text{ and its process } o\_0 \text{ exist in the initial state} \\
& \wedge O\_func = \{o\_0\} \\
& \wedge O\_data = \{\} \\
& \wedge O\_na = \{o\_sorm,\ o\_2\} \\
& \text{the other subjects have not been activated yet} \\
& \wedge S\ = \{s\_0,\ s\_sorm,\ s\_2,\ s\_3,\ s\_4\} \\
& \wedge Q = \langle q\_0 \rangle
\end{aligned}
$$

Fig. 1. Initial state of the system, describing variables – active and inactive access subjects, available access objects, sequence of accesses.

$$
\begin{aligned}
Next \triangleq\ & \\
& \text{Queries to the system} \\
& \vee CreateProcessD \\
& \vee \ldots \\
Spec \triangleq\ & Init \wedge \Box[Next]_{vars}
\end{aligned}
$$

Fig. 2. *Spec* specification of the system described as the $Init$ state and the possible $Next$ actions that change the $vars$.

TLA+ language, however this may be considered as a general description without reference to any specific security models, access control tools or any specific approach to verification.

II. Materials and Methods

When using TLA+ and the Model Checking method, the simplest specification of an arbitrary system *Spec* must contain the following elements [7], [8]:

1) Variables of the model ($vars$), for example, a set of objects and subjects of access.
2) Initial state of the system ($Init$), described as a predicate with the initialization of model variables, an example of which is shown in Fig. 1 [8].
3) Possible actions of the system ($Next$), described in the form of predicates of pre- and post-conditions for its execution, for example, the conditions needed for read access and the results of such access.
4) A theorem that is proved during verification and that checks special predicates (formal properties of the system) – invariants and temporal properties [5], [6].

An example of the simplest specification is shown in Fig. 2

The first verification feature is the need to correctly choose the initial state and model variables that allow describing the states of the system. In fact, the whole further verification process and correctness of the modeled system will depend on this choice. Lamport's temporal logic of actions and the Model Checking method make it possible to verify only those systems that are given in the form of finite state automata [7], [8], that is, the Model Checking method can only terminate when there are no more new, reachable and unexamined states of the system. On the one hand, this greatly limits the value of applying this verification approach for arbitrary systems, since it seems that verification results cannot guarantee execution of the same formal properties for any such system (for example, with a significantly larger number of access subjects and objects). On the other hand, this approach is a kind of implementation of justification for the "induction step", meaning that if the modeled system's formal properties cannot change in the case of its expansion, the same properties will retain for an arbitrarily large system (using the mathematical induction method, verification can be performed first for a system having one subject, and then for a system having n and $n+1$ subjects). The problem of the most modern verification tools is that the user considers the choice of the initial state and model variables as not a really significant step, which can be done unconsciously. However, if this stage is performed incorrectly, correctness and objectivity of the entire further verification process cannot be considered reliable.

To restrict the modeled system, that is, to create a finite space of its states, TLA+ uses model values for some system entities ($vars$), for example, for subjects, objects, users. With an increase in the number of model values, the number of system states can grow exponentially and, accordingly, the verification time will also increase exponentially.

Therefore, it is desirable to choose the minimum necessary variables of the modeled system specification for verification purposes, whose increase can't result in a change in the system's behavior (invariants and temporal properties will not be violated). For example, if the specification takes into account the possibility of simultaneous existence of two equally privileged system access subjects (s_3 and s_4 in Fig. 1), the system's behavior within the described specification is likely to remain the same for 4–10 or more users. On the other hand, in such a case, it is not recommended to limit the model values by only one access subject (s_3), since the behavior of the system specification and the truth of the predicates of formal security properties can change when several access subjects work simultaneously (for example, hidden data leakage channels can appear).

Formal properties of the system in TLA+ are represented by invariants and temporal properties. Truth of the invariants is checked in all states and for each implementation of the system; the predicate is compiled for the current state, and truth is checked sequentially in each state. Temporal properties are more complex predicates that can use special temporal operators [1], [5], [6], allowing to take into account the time factor, for example:

- $[\,]P$ means that P is true for all states (an analog of the invariant, but without optimization by verification tools);
- $<>\ P$ means that for every possible implementation, at least one state has P as true;
- $P\ \sim>\ Q$ implies that if P ever becomes true, at some point afterwards Q must be true;
- $<>\ [\,]P$ says that at some point P becomes true and then stays true.

In accordance therewith, *the second feature of verification* via TLA+ is the need to use predominantly invariants rather than temporal properties as formal properties of the system. This is due to the fact that when checking temporal properties during verification, a huge number of previous or future states can be analyzed, which can lead to impossibility of completing the verification process. Moreover, according to

```
 macro for selecting latest proceeded query
SelectPrevQuery(Sq) ≜ Sq[Len(Sq)]

 temporal property for checking the impossibility of
 changing an object after reading
TemporalProperty ≜
LET obj ≜ SelectPrevQuery(Q).dent
    id ≜ CHOOSE i ∈ ObjectIDs: ∃ o ∈ Objects: o.oid = i
     type ≜ SelectPrevQuery(Q).typeIN
     (type = "read" ∧ obj ∈ Objects)  ∧ obj.oid = id)
          ⇝ □(IF   type = "write"
                THEN SelectPrevQuery(Q).dent ≠ obj
                 wrong, should be:
                 THEN obj.oid ≠ id
                ELSE TRUE)
```

Fig. 3. Predicate of the temporal property of the TLA+ notation, containing an error in using a history variable.

[9], when using some combinations of temporal operators, a so-called state explosion can occur, for example, when using the predicate $\sim (P \sim> Q)$.

If it is impossible to do only with invariants in the specification, try using auxiliary or history variables [10]. Thus, Fig. 1 shows that a Q sequence of all accesses is used as one of the model variables. In fact, this variable is not used in the model specification and is not implemented in practice in access control policies. The only purpose of this variable is to provide a possibility, within the framework of invariants, to check events that occurred in the past (for example, whether a write access was provided to an access object which is being read in the current state, or whether this access object was read in a previous state before execution access was granted). Temporal properties must be used only in cases where it is not enough to use only invariants or invariants with auxiliary variables.

The third feature is related to how auxiliary variables can be applied, as well as how to develop conditions in temporal properties. In Fig. 1, the Q historical variable is a $Sequence$, since it is sometimes required to analyze the order of system transitions for some formal properties of the applied security model. In the absence of such a need, a Set can be used as Q, but in this case only the fact of certain transitions of the system in the past without their order will be taken into account.

In addition, history variables must be carefully used in conjunction with TLA+ temporal operators. At the when temporal operators are executed, predicates for entities that do not exist in the system can be checked, and the value of the history variable may not correspond to the required state of the system, as, for example, in Fig. 3.

Fig. 3 shows the $SelectPrevQuery$ macro, which selects the last access query executed in the system. Further, in the temporal property predicate, obj entity is selected, which will be accessed in the current state of the system. If obj is an object and is being read, there must be no write access to this object in any future states. In the next-to-last line of Fig. 3, there is an error due to specific features of the TLA+ language – the $SelectPrevQuery$ macro will be deployed in such a case for the state in which the temporal property is checked, with obj also deployed for the current state (the predicate will always have the $FALSE$ value). In such comparisons, it must be also taken into account that there can be no obj entity in the system during the temporal property test, and the comparison with obj may either be false or generate an error. Even if the object is deleted and a complete copy is created, the new obj structure will not be comparable to the same structure before deletion. To fix the situation in such a case, one can number and use identifiers for access objects, as in Fig. 3 in the comments. Moreover, instead of checking query types, one can change the object's state field, for example, after accessing it for writing.

Most often, invariants and temporal properties are used in order to form the safety properties of the TLA+ notation, that is, to check that nothing bad will happen when the system runs [9]. However, liveness properties of TLA+ are no less important, that is, checking that the system is functioning and that the expected actions are taking place therein [9]. In accordance therewith, *the fourth verification feature* can be formulated as follows – it is necessary to form both safety and liveness properties in the form of invariants or temporal properties. In [8], the authors managed to identify several errors in the TLA+ notation, which resulted in the following gaps:

1) The system stopped functioning, since the last system process that could generate other subjects and users ceased to work.
2) The system ceased to be manageable, since all administrator users were blocked in it.

Such errors could be eliminated by establishing the appropriate liveness invariants and correcting the predicates of system actions that lead to their violation.

The last, *fifth feature* identified by the authors is the ability to take into account finiteness of the work, which is called termination in the TLA+ notation [9], as well as performance of the modeled system over time. For many algorithms and systems, it is important to check completion of their work in some state. But the *Spec* model specification in Fig. 2 describes a system that can operate indefinitely, which can be detected using a special option of verification tools. The reason for this is the stuttering states of TLA+ – the system states where it is idle, that is, it does not perform any actions. Such states can become a problem for some liveness properties, since there may arise system implementations, where, for example, no user will ever start working or the access control subsystem will never be activated.

If it is supposed to exclude stuttering states from the model, one can use special temporal assumptions during initialization, for example, TemporalAssumption in Fig. 4.

Such a temporal assumption provides the Weak Fairness property of the modeled system [5] – if it is possible to perform any action in the system, it must be performed.

Temporal liveness-property

$$TemporalAssumption \triangleq$$

if the action is available, the system must perform it
(otherwise the system will remain in the stuttering state)

$$\wedge \, \mathrm{WF}_{vars}(Next)$$

$$Spec \triangleq Init \wedge \Box[Next]_{vars} \wedge TemporalAssumption$$

Fig. 4. *Spec* specification of the system with a temporal property for excluding stuttering states.

III. Results and Discussion

The paper describes special features of the TLA+ temporal logic of actions for verifying access control policies. It gives recommendations for taking these features into account in the verification process. It also specifies possible errors that may occur when verifying arbitrary data protection tools.

One of the most important features is the need to correctly select the initial state of the system and model variables. In the case of an inadequate choice, verification will have no sense in practice, since the "induction step" required to execute formal properties of the system in all possible states will be unreasonable.

In addition to those considered, it is important to take into account one more general key feature for various approaches to verification – the need to select specific formal properties that require justification. Such properties must cover the verification tasks and directly result in some system characteristics (for example, impossibility of information flows from objects with a high confidentiality level to objects with a low confidentiality level). Otherwise, verification will be useless, since it will only create a false impression of system "rightness", "correctness" or "security/safety", which cannot be its objective characteristic.

It is important to note that a subtle point of the verification process is the need to substantiate correspondence of the high-level description of the modeled system in a formal language suitable for verification to the real system. A short system specification cannot completely cover all the nuances of its implementation described in long source codes. During verification, the system is often simplified, which is subsequently not taken into account when checking formal properties. In this regard, even is using modern tools, it is possible to verify a model that does not correspond to the system implemented in practice. Further research should be done on this topic.

References

[1] A.V. Kozachok, "TLA+ based access control model specification," Proceedings of the Institute for System Programming of the RAS, vol. 30(5), 2018, pp. 147–162, DOI:https://doi.org/10.15514/ISPRAS-2018-30%285%29-9.

[2] P.N. Devyanin, A.V. Khoroshilov, V.V Kuliamin, A.K. Petrenko and I.V. Shchepetkov "Formal Verification of OS Security Model with Alloy and Event-B," Lecture Notes in Computer Science, vol. 8477. Springer-Verlag: Berlin/Heidelberg, 2014, pp. 309–313, DOI:https://doi.org/10.1007/978-3-662-43652-3_30.

[3] G. Klein [et al.], "SeL4: formal verification of an OS kernel," Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, 2009, pp. 207–220, DOI:https://doi.org/10.1145/1629575.1629596.

[4] International Organization for Standardization "ISO/IEC 15408-3. Information technology Security techniques – Evaluation criteria for IT security – Part 3: Security assurance components", 2008.

[5] L. Lamport, "The temporal logic of actions," ACM Trans. Program. Lang. Syst., vol. 16(3), 1994, pp. 872–923, DOI:http://doi.acm.org/10.1145/177492.177726.

[6] L. Lamport, J. Matthews, M. Tuttle, and Y. Yu, "Specifying and verifying systems with TLA+," Proceedings of the ACM SIGOPS 10th workshop, 2002, pp. 45–48, DOI:https://doi.org/10.1145/1133373.1133382.

[7] Kanner A. M., Kanner T. M., "Modeling and verification of the access control subsystem of Accord-X data security tool," Information Security Questions, vol. 3(130), 2020, pp. 6-10

[8] Kanner A. M., Kanner T. M., "Verification of a Model of the Isolated Program Environment of Subjects Using the Lamports Temporal Logic of Actions," Proceedings of the VII Engineering & Telecommunication Conference, 2021, in press.

[9] H. Wayne, "Practical TLA+: Planning Driven Development," Apress, 2018, DOI: http://doi.org/10.1007/978-1-4842-3829-5.

[10] L. Lamport, S. Merz, "Auxiliary variables in TLA+," arxiv.org preprint:1703.05121, 2017, URL: https://arxiv.org/pdf/1703.05121.pdf.