Verification of a Model of the Isolated Program Environment of Subjects Using the Lamport's Temporal Logic of Actions

1st Andrey M. Kanner
National Research Nuclear University MEPhI
Moscow, Russia
kanner@mail.ru

2nd Tatiana M. Kanner
National Research Nuclear University MEPhI
Moscow, Russia
sheikot@mail.ru

***Abstract*—The paper considers a modern approach to the creation of formal computer system security models, which consists in describing a model in some formal language suitable for its verification for compliance with the expected properties. The paper provides an example of such a description in the form of a specification of a formal model of the isolated program environment in the language of the Lamport's temporal logic of actions. The specification is formed as an initial state of the system, a list of possible further actions and a set of invariants and temporal properties to which the system's states must correspond. The initial state is described by some entities that must exist in each system implementation. The system's actions are given in the form of predicates of pre- and postconditions, with some model's variables changing in the latter. Invariants and temporal properties are described in the form of predicates, whose truth must be checked in each possible state of the system or depending on the conditions occurring in previous or future states. The paper considers the features of forming a security model specification in TLA+ notation and verifying it using special tools. In its conclusion, the paper describes the results of verifying the specification of the formal model of the isolated program environment of subjects, the existing problems and directions for further research on this topic.**

***Index Terms*—isolated program environment of subjects, security model, verification, temporal logic**

I. Introduction

As a rule, most of the known formal security models of computer systems are formulated in the mathematical notation, using a suitable mathematical apparatus. In such a case, the basic security theorem is the main component of any formal model, which helps to substantiate some formal properties that guarantee security of the system or the data processed therein. The main disadvantages of this approach are as follows:

- Complexity of describing a particular model and possible hidden flaws in the mathematical notation or the basic security theorem, which are difficult to identify manually.
- Detachment of the mathematical notation from real systems, where it is planned to use the security model, as well as possible presence in such real systems of some conditions that would affect the provisions used in proving the basic security theorem.
- Impossibility of ensuring fulfillment of the formal properties, which theoretically guarantee system security in all its states in practice.

Due to the aforementioned disadvantages, verification tools are currently used to confirm the formal security properties of a computer system, which make it possible to automatically check the system security in all possible states. To apply this approach, notation in a certain formal language is used instead of the mathematical notation [1]–[4]. Therefore, modern security models are often developed first in a formal language suitable for verification, and only then, if required, are translated into the mathematical notation. In [5], the authors used the classical approach to describing a model of the isolated program environment of subjects (IPES) in form of the mathematical notation. This model is based on the provisions of the well-known SO-model of the isolated program environment in [6] and [7], in contrast to which, when decomposing the system into entities, subjects will consist of users and system services (not user processes), and objects will consist of objects functionally associated with such subjects (processes) and data objects with the ability to dynamically change their composition over time. A distinctive feature of the IPES model is inclusion of the protection subsystem as a system entity along with other subjects of the system, and substantiation of the impossibility of violating the current access control rules due to the property of absolute correctness (isolation) of access subjects. Such works as [8] and [9] contain experimental research of implementing the IPES model, which, however, do not allow guaranteeing fulfillment of formal security properties in all possible states of the system in practice. Other obvious problems consist in improvement of the IPES model and the mathematical apparatus used to describe it, as well as the associated need for repeated experimental research. In accordance with this, the authors decided to apply a modern approach to the development of security models, which involves verification and testing for implementation of security properties in practice. In this paper, verification means not practical testing or experimental studies, but rather formal confirmation of some properties of

the security model in all possible future states of the system.

II. Materials and Methods

There are many approaches to modeling arbitrary systems or algorithms used to verify their compliance with some formally described properties [1]–[3]. The approach to verification using Lamport's temporal logic of action (TLA, Temporal Logic of Actions) and the Model Checking method in [10] and [11] is one of such options, which allows, within the framework of formal notation in the TLA+ language, to describe all the necessary system entities and operations, as well as the security properties of the IPES model, which are required for verification in all its states.

Modeling in the language of Lamport's temporal logic of actions makes it possible to describe and further automatically verify the systems given in the form of finite automata [12]. However, this means that when using this approach there occur some restrictions on the possibility of verification, since the Model Checking method does not allow to perform a full-fledged formal verification of the system. In accordance with this, within the framework of the TLA+ notation of the security model, model values were introduced for some system entities: the number of subjects and objects in [1] and [2]. These model values are introduced to ensure that the verification process may be completed and at the same time the modeled system itself is not significantly restricted. So, in the TLA+ notation, two more user subjects and one system subject may simultaneously exist along with s_0 system subject (the operating system kernel) and s_{sorm} access control subsystem. In a state when all of the listed subjects are active, they may have one associated process object and, in addition, several objects associated or not associated with access subjects may be created. An increase in the given model values of the number of possible subjects or objects will not affect the modeled system, but will significantly increase the verification time, the number of possible states and system implementations.

In the IPES model, the initial state of each system implementation is assumed to have only s_0 system subject, which is required for its further functioning. The initial state of the model in the TLA+ notation is shown in Fig. 1.

The following elements of the IPES model are the variable $vars$ entities of the system corresponding to Fig. 2:

- Set of active subjects (S_{active});
- Set of functionally associated objects (O_{func});

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

Fig. 1. Initialization of the IPES model.

$$vars \triangleq \langle S\_active,\ O\_func,\ O\_data,\ O\_na,\ S,\ Q \rangle$$

Fig. 2. Variables of the IPES model.

- Set of associated data objects (O_{data});
- Set of unassociated objects (O_{na});
- Set of both active and inactive subjects (S);
- Sequence of the queries sent to the system (Q).

The following queries of the IPES model may be made as further actions in the system: $CreateProcessD$, $DeleteProcessD$, $CreateUserD$, $CreateShadowD$, $DeleteSubjectD$, $ExecD$, $ReadD$, $WriteD$, $CreateD$, $DeleteD$, corresponding to standard operations in the operating system. Each action is described using pre- and postconditions for its implementation, for example, as for the $CreateProcessD$ query in Fig. 3. Preconditions represent the predicates that must be met to perform an action. Postconditions determine how the model variables change upon performance of the action, that is, what new state the system will have.

$$
\begin{aligned}
& CreateProcessD \\
& \text{Creation of a functionally associated object} \\
& CreateProcess(s,\ o,\ o\_c) \triangleq \\
& \qquad \wedge\ O\_func' = O\_func \cup \{o\_c\} \\
& \qquad \wedge\ Q' = Append(Q,\ [subj \mapsto s,\ proc \mapsto o, \\
& \qquad\qquad\qquad\qquad dent \mapsto o\_c, \\
& \qquad\qquad\qquad\qquad type \mapsto \text{“create\_process”}]) \\
& \qquad \wedge\ \text{UNCHANGED}\ \langle S\_active,\ O\_data,\ O\_na,\ S \rangle \\
& \\
& CreateProcessD \triangleq \\
& \qquad \text{activating impact of the subject and its process} \\
& \qquad \exists\, s \in S\_active : \\
& \qquad \exists\, o \in SelectSubjProc(s) : \\
& \qquad \exists\, st \in ObjectStates : \\
& \\
& \qquad \exists\, x \in ObjectIDs : \\
& \\
& \qquad\quad \text{let us denote a new object as} \\
& \qquad\quad \text{LET}\ o\_c \triangleq [oid \mapsto x, \\
& \qquad\qquad\qquad\qquad type \mapsto \text{“func”}, \\
& \qquad\qquad\qquad\qquad subj\_assoc \mapsto \{s.sid\}, \\
& \qquad\qquad\qquad\qquad state \mapsto st]\ \text{IN} \\
& \\
& \qquad\quad \wedge\ \ \text{IF}\ o.state \neq s.sid \\
& \qquad\qquad\quad \text{THEN}\ st = StateChanged \\
& \qquad\qquad\quad \text{fork inherits the state} \\
& \qquad\qquad\quad \text{ELSE}\ \ st = o.state \\
& \\
& \qquad\quad \text{selection of a free identifier} \\
& \qquad\quad \wedge \forall\, obj \in SelectObjects : obj.oid \neq o\_c.oid \\
& \qquad\quad s\_sorm \text{ access rules} \\
& \qquad\quad \wedge\ SormCheckPerm(s,\ o\_c,\ \text{“create\_process”}) \\
& \\
& \qquad\quad \text{postconditions} \\
& \qquad\quad \wedge\ CreateProcess(s,\ o,\ o\_c)
\end{aligned}
$$

Fig. 3. Post- and preconditions for a query to create functionally associated objects (processes).

Thus, the $Spec$ specification of the IPES model in the TLA+ notation presented in Fig. 4 must determine the initial state of the system and a list of possible further actions (operations or queries to the system).

Invariants (security properties) and temporal properties written as predicates are used in the specification as system security properties. The invariants must be met in all states and for each system implementation, and they also can, by using the Q sequence of queries to the system, check the conditions occurred in the past, for example, during the last system transition. Temporal properties, in contrast to invariants, can use special temporal TLA+ operators [1], [2], [10], [11], which help to compose predicates depending on the execution time and certain events in the past or future.

The first part of the invariants of the IPES model specification is introduced in the TLA+ notations to control whether operations are described correctly in each possible state: to control the types of model variables ($TypeInv$), to control consistency of the sets of objects associated and not associated with the subjects and uniqueness of system entities ($ConsistencyInv$), to control blocking of unregistered access subjects ($BlockedInv$) and to control system performance (continuous presence of s_0 subject in the $OSKernelExists$ invariant).

The second part of the invariants is introduced in the TLA+ notation to check security properties of the IPES model [5] in each possible state: to control activation of the access control subsystem ($SormInits$), to check the correctness property of the system subjects ($Correctness$), to check the new property of the subject's absolute correctness in the opposite sense ($AbsCorrectnessOpp$). The subject's correctness property is shown in Fig. 5 and presupposes that it is impossible to change the associated objects of another access subject. In modern systems, this property is partially achieved by using virtual address spaces of processes and restrictions on inter-process interaction.

The property of subject absolute correctness in the opposite sense is shown in Fig. 6, and introduced in the opposite way in comparison with the absolute correctness property (in the direct sense) [5]. The essence of this property is that you may not make queries to the system to read objects that have been changed by other access subjects.

This property is introduced in the IPES model, since it is often problematic in practice to check events in the future, while checks of past system states (for example, a violation of access object integrity) are easily implemented in practice. Therefore, the absolute correctness property of subjects of the

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

Fig. 4. Specification of the IPES model.

```
Subject correctness property in case of system transition
Correctness ≜
LET ent ≜ CHOOSE e ∈ Entities : e = SelectPrevQueryDent(Q)
    subj ≜ CHOOSE s ∈ Subjects : s = SelectPrevQuerySubj(Q)
    proc ≜ CHOOSE p ∈ Objects  : p = SelectPrevQueryProc(Q)
    r ≜ CHOOSE q ∈ QueryTypes : q = SelectPrevQueryType(Q) IN
  you may not change the state of associated objects of another subject
  IF r ∈ QueriesStateChange
  THEN
    "write", "delete", ...
    IF ent ∈ Objects ∧ ent.type ∈ {"func", "data"}
    THEN ent.subj_assoc ⊆ {subj.sid}
    ELSE IF ent ∈ Subjects  delete_subject
         THEN proc.subj_assoc ⊆ {ent.sid}
         ELSE TRUE
  ELSE IF (r ∈ QueriesAssocChange
           ∧ ent ∈ Subjects)  create_user, create_shadow
  THEN   proc.state ∈ {ent.sid, s_0.sid}
  ELSE
    TRUE
```

Fig. 5. Invariant used to check the correctness property of access subjects.

```
Absolute correctness property of subjects in the opposite sense
AbsCorrectnessOpp ≜
  LET ent ≜ CHOOSE e ∈ Entities : e = SelectPrevQueryDent(Q)
      subj ≜ CHOOSE s ∈ Subjects : s = SelectPrevQuerySubj(Q)
      proc ≜ CHOOSE p ∈ Objects : p = SelectPrevQueryProc(Q) IN
   a previously changed object may not become an associated object
   IF EntityStateChanging(ent)
   THEN
     IF EntityStateChanged(subj, proc, ent)
        integrity has been violated
     THEN FALSE
     ELSE TRUE
   ELSE TRUE
```

Fig. 6. Invariant used to check the absolute correctness property of subjects in the opposite sense.

IPES model (in the direct sense) is presented in the form of $AbsCorrectness$ temporal property in the TLA+ notation in Fig. 7. It checks the conditions related to future accesses, which may not be checked within the TLA+ invariants or in the pre- or postcondition predicates for queries to the system. The essence of this property is that you may not write in the objects that will become associated with other access subjects in future system states.

The possibility of using the system is also expressed in the form of $OSUsabilityLiveness$ temporal property, which means that one more entity will be generated in any system implementation along with s_0 and s_{sorm} subjects: a user or a system subject.

All system operations include special $SormCheckSubj$ and $SormCheckPerm$ checks, which first of all check whether the access control subsystem has been initialized using $SormInitialized$ macro, and only then perform the checks required to fulfill the correctness properties of the IPES model.

Verification of the described specification with respect to the given invariants and temporal properties allows us to assert automatic proof of the theorem in the TLA+ notation shown in Fig. 8, which confirms fulfillment of security requirements for all possible states and system implementations.

```
Absolute correctness property of subjects
AbsCorrectness ≜
  LET ent ≜ CHOOSE e ∈ Entities : e = SelectPrevQueryDent(Q)
      subj ≜ CHOOSE s ∈ Subjects : s = SelectPrevQuerySubj(Q)
      proc ≜ CHOOSE p ∈ Objects : p = SelectPrevQueryProc(Q) IN
   if the object has been changed
   (EntityStateChanged(subj, proc, ent))
   the object will not become associated with another subject in the future
     ⇝ ◇□(IF   SelectPrevQueryDent(Q) = ent
          THEN ¬EntityStateChanging(ent)
          ELSE TRUE)
```

Fig. 7. Temporal property used to check the absolute correctness property of subjects.

```
The theorem, which takes into account the invariants
and properties: is proven upon verification
THEOREM Spec ⇒ ∧ □TypeInv
               ∧ □ConsistencyInv
               ∧ □BlockedInv
               ∧ □OSKernelExists
               ∧ □SormInits
               ∧ □Correctness
               ∧ □AbsCorrectnessOpp
               ∧ OSUsabilityLiveness
               ∧ AbsCorrectness
```

Fig. 8. The theorem checked during verification of the security model specification.

III. Results and Discussion

When forming the TLA+ notation of the IPES model, the authors encountered the following features that should be noted.

Some variables and entities of the model specification do not need to be implemented in practice. Such entities include Q sequence of queries, which is an auxiliary (history) variable [13] and is used only to check invariants or temporal properties, which are not implemented in practice as well.

In the above TLA+ notation, Q history variable is a sequence, not a set, since this is the order of the performed actions and not only the fact of their presence in the past, which is important in the IPES model. History variables must be used with care, especially when using temporal TLA+ operators. This is connected with the fact that some system entities may stop to exist in it by the time of execution of temporal operators. Moreover, instead of a certain value of the history variable, you can mistakenly check its current value at the time when the temporal property is executed. In addition, the temporal properties themselves, in contrast to the invariants, can be executed too long when working with complex or long values of history variables.

In the TLA+ notation, in contrast to the mathematical notation, it is not needed to reflect entities such as the set of all states or implementations of the system. These entities arise only during the verification process using the Model Checking method – during the verification, all possible sequences of states and queries of the system are generated, taking into account the restrictions and model values.

In the model specification, we had to reduce the number of stuttering states by introducing the temporal assumption $TemporalAssumption = WF_{vars}(Next)$, which ensures the Weak Fairness [10]. In this case the model does not include any unnecessary states and implementations in which the system does nothing – the so-called stuttering states. If it is possible to perform any action in the system, it must be performed. Otherwise, such a system implementation may arise during verification, in which it would be impossible to generate users yielding violation of $OSUsabilityLiveness$ temporal property. The requirements for system termination are not considered in the model, therefore the verification ends only when the tools go through all possible states of the system, and all new states begin to repeat the previous ones.

The verification of the developed specification of the IPES model was carried out using TLC2 v2.15 tool on a computer running under Intel Core $i5-9400$ (3.80 GHz) and having 16 GB RAM in 64-bit Linux OS with $v5.4.38$ kernel. The verification took 20 hours 25 minutes using 6 separate streams, with the total number of analyzed states amounting to 124299849.

The above verification allowed to do as follows:

- To check fulfillment of the required security properties in all possible states and implementations of the system in order to justify the impossibility of violating the access control policy acting in the system;
- To confirm identity of such security properties of the IPES model as the absolute correctness of subjects in the direct and opposite sense;
- To describe specific access rules given in the form of $SormCheckSubj$ and $SormCheckPerm$ predicates, which, when introduced, allow the required security properties to be implemented in the system.

The full text of the developed specification of the IPES model is available on the author's website[1].

IV. Conclusion

Use of a modern approach for describing security models in a certain formal language suitable for verification has a number of advantages in comparison to the classical approach, which supposes description in the mathematical notation. The main advantage is the ability to quickly find hidden errors at an early stage of creation of a security model, since security invariants can be created as long as the basic operations of the model are described. Moreover, it is not required to involve highly qualified specialists in the verification of the main provisions of the model, since fulfillment of the expected properties can be checked automatically.

Nevertheless, modeling in formal languages suitable for verification cannot fully eliminate possible errors and all the shortcomings of the classical approach specified in the introduction. The difference is that such shortcomings can be quickly detected and corrected, while there is no need to simultaneously correct several notations at once – the

[1] https://github.com/kanner/ipes-model

mathematical one and one in a formal language for verification. Another difficult task is to substantiate compliance of the formal model with its implementation in the program code. According to the sources known to the authors, tools for automatic generation of a model specifications from the program code or, conversely, generation of a source code for a given formal model specification are still under development. There are only particular solutions within the framework of academic research [14].

This work can be developed further by adding concurrency and multiprocessing in the security specification. This is needed to better correspond to real systems, where several operations can be performed at once in one unit of time, and the threshold between the earlier and the later operations is blurred. In the current implementation, the IPES model specification is simplified – only one query is executed per one time unit, and 1 or 2 subjects and 2 or 1 system objects are involved, respectively. The second direction for development can consist in a transition from the Model Checking method to use of deductive verification using TLAPS or Izabelle tools, as well as other SMT solvers.

REFERENCES

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

[2] A.V. Kozachok, "Specifikatsiya modeli upravleniya dostupom k raznokategoriynym ustroistvam kompyuternykh sistem", Voprosy kiberbezopasnosti, 2018, vol. 4(28), pp. 2–8. (In Russ.)

[3] P.N. Devyanin [et al.], Modelirovaniye i verifikatsiya politik bezopasnosti upravleniya dostupom v operatsionnykh sistemakh, Moscow: Goryachaya liniya – Telekom, 2019. (In Russ.)

[4] 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.

[5] A.M. Kanner, "Correctness of data security tools for protection against unauthorized access and their interaction in GNU/Linux", Global Journal of Pure and Applied Mathematics, 2016, vol. 12(3), pp. 2479–2501.

[6] A.Yu. Scherbakov, Sovremennaya kompyuternaya bezopasnost'. Teoreticheskiye osnovy. Prakticheskiye aspekty, Moscow: Knizhny mir, 2009. (In Russ.)

[7] A.Yu. Scherbakov, Khrestomatiya spetsialista po sovremennoy informatsionnoy bezopasnosti, vol. 1. Saarbrucken: Palmarium Academic Publishing, 2016. (In Russ.)

[8] A.M. Kanner, "Experimental research of access control in Linux operating system", Information and Security, 2017, vol. 20 (4), pp. 604–609. (In Russ.)

[9] A.M. Kanner, "The effectiveness of implemented security features against unauthorized access in Linux operating systems", Information security questions, 2017, no. 2, pp. 3–8. (In Russ.)

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

[11] L. Lamport, J. Matthews, M. Tuttle, 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.

[12] A.M. Kanner, T.M. Kanner, "Testing software and hardware data security tools using the automata theory and the graph theory", Proceedings of Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology, 2020, pp. 615–618, DOI: https://doi.org/10.1109/USBEREIT48449.2020.9117757.

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

[14] A. Methni , M. Lemerre, B.B. Hedia, S. Haddad, K. Barkaoui, "Specifying and verifying concurrent C programs with TLA+", International Workshop on Formal Techniques for Safety-Critical Systems, Springer, Cham, 2014, pp. 206–222, DOI: https://doi.org/10.1007/978-3-319-17581-2_14.