UDC 004.056
https://doi.org/10.32362/2500-316X-2024-12-2-16-27

RESEARCH ARTICLE

Analysis of information flow security using software implementing business logic based on stored database program blocks

Aleksey A. Timakov @

MIREA – Russian Technological University, Moscow, 119454 Russia
@ Corresponding author, e-mail: timakov@mirea.ru

Abstract
Objectives. Verification of software security is typically performed using dynamic and static analysis tools. The corresponding types of analysis do not usually consider the business logic of the software and do not rely on data access control policies. A modern approach to resolving this problem is to implement language-based information flow control. Despite a large amount of research, mechanisms for information flow control in software are not widely used in practice. This is because they are complex and impose increased demands on developers. The aim of the work is to transfer information flow control from the language level to the level of formal verification. This will enable the functions of controlling data integrity and confidentiality in software to be isolated into a separate task, which can be resolved by information security analysts.
Methods. The research is based on general formal security methods for computer systems and formal verification methods. The algorithm developed by the author for checking security specifications and resolving security violations uses temporal logic of actions.
Results. The technology is presented as a step-by-step approach to resolving specific tasks, including the following: designing a database (DB) for storing and processing sensitive information; analyzing dependencies and identifying relevant sets of program blocks in the DB; generating TLA+ specifications for the identified program blocks; labeling specifications according to global security policy rules and additional constraints; applying the specification verification algorithm, and resolving security violations while providing recommendations for software developers. The procedure also involves analyzing labeled data, in order to control the spread of verified program block output values in external software modules.
Conclusions. The technology presented herein does not require developers to include redundant annotations describing security policy rules. The function of analyzing information flows with reference to predefined access restrictions is moved to a separate stage of the software development life cycle.

Keywords: information flows, information flow control, formal verification, language platform, security policy, abstract semantics, non-interference

• **Submitted:** 11.05.2023 • **Revised:** 03.10.2023 • **Accepted:** 07.02.2024

For citation: Timakov A.A. Analysis of information flow security using software implementing business logic based on stored database program blocks. *Russ. Technol. J.* 2024;12(2):16–27. https://doi.org/10.32362/2500-316X-2024-12-2-16-27

Financial disclosure: The author has no a financial or property interest in any material or method mentioned.

The author declares no conflicts of interest.

НАУЧНАЯ СТАТЬЯ

Технология анализа безопасности информационных потоков в программном обеспечении, реализующем бизнес-логику с использованием хранимых программных блоков баз данных

А.А. Тимаков @

МИРЭА – Российский технологический университет, Москва, 119454 Россия
@ *Автор для переписки, e-mail: timakov@mirea.ru*

Резюме
Цели. Проверка свойств безопасности программного обеспечения (ПО) при построении информационных систем с высоким уровнем доверия, как правило, осуществляется с использованием инструментов динамического и статического анализа. Соответствующие виды анализа обычно не учитывают бизнес-логику ПО и не опираются на политику управления доступом к данным. Современным направлением решения проблемы является контроль информационных потоков. Несмотря на большое количество проведенных исследований, механизмы контроля информационных потоков в ПО пока не находят широкого применения на практике, поскольку обладают значительной сложностью и диктуют повышенные требования к разработчикам. Целью работы является перенос контроля информационных потоков с языкового уровня на уровень формальной верификации и выделение функции контроля целостности и конфиденциальности данных в ПО в самостоятельную задачу, решаемую аналитиками информационной безопасности.
Методы. Исследование опирается на общие формальные методы безопасности компьютерных систем и методы формальной верификации. Разработанный автором алгоритм проверки спецификаций использует аппарат темпоральной логики действий.
Результаты. Представлена технология, предполагающая поэтапное решение частных задач: проектирование базы данных (БД) для хранения и обработки подлежащей защите информации, анализ зависимостей и выделение релевантного множества программных блоков БД, генерация спецификаций TLA+ выделенных программных блоков БД, разметка спецификаций в соответствии с правилами глобальной политики безопасности и дополнительными ограничениями, применение алгоритма проверки спецификаций и устранение нарушений инварианта безопасности с внесением рекомендаций для разработчиков ПО, применение процедуры анализа помеченных данных для контроля распространения выходных значений верифицированных программных блоков БД во внешних программных модулях.
Выводы. Представленная технология не требует от разработчиков внесения избыточных аннотаций, описывающих правила политики безопасности. Функция анализа информационных потоков с привязкой к заданным в системе ограничениям доступа выносится на отдельный этап жизненного цикла разработки ПО.

Ключевые слова: информационные потоки, контроль информационных потоков, формальная верификация, языковая платформа, политика безопасности, абстрактная семантика, информационное невлияние

• **Поступила:** 11.05.2023 • **Доработана:** 03.10.2023 • **Принята к опубликованию:** 07.02.2024

Для цитирования: Тимаков А.А. Технология анализа безопасности информационных потоков в программном обеспечении, реализующем бизнес-логику с использованием хранимых программных блоков баз данных. *Russ. Technol. J.* 2024;12(2):16–27. https://doi.org/10.32362/2500-316X-2024-12-2-16-27

Прозрачность финансовой деятельности: Автор не имеет финансовой заинтересованности в представленных материалах или методах.

Автор заявляет об отсутствии конфликта интересов.

INTRODUCTION

The conditions for assigning automated systems to certain protection classes are determined by evaluation standards. At present, such security requirements are formulated in terms of “General Criteria for Assessing the Security of Information Technologies”[1] and include functional and trust requirements [1]. Based on an analysis of the regulatory documents,[2, 3, 4] it follows that there are three important categories of conditions to be taken into account when determining the protection class: control of legal trajectories and media of information distribution[5]; control of hidden channels; and formal proof of the effectiveness of the implemented protection mechanisms or security of computing.

At the application level, verifying computing security is most difficult. In practice, a complete solution to this problem has not been achieved, even in the context of controlling legal information dissemination trajectories. Information dissemination trajectories in software (software) which conform to the rules provided by its logic can be obtained from the control flow graph. This is provided that the control flow has integrity. In order to ensure data confidentiality and integrity in software, a combination of formal, semi-formal and informal techniques is used at different stages of system development. Some of these techniques include dynamic and static code analysis, symbolic (cosymbolic) execution, formal verification, and fuzzing testing.[6] Additional protective measures such as randomization of dynamic memory offsets, stack execution protection, control-flow integrity control, among others, can be applied at the platform and compiler level. Although the checks implemented using the above methods play an essential role in ensuring information protection, they mostly do not take into account the specifics of processed data and business logic of the application. Formal methods which take into account the specifics of data include methods based on information flow control (IFC).

Any technology of computing security analysis based on instrumentation in software (IFC) has four commonly accepted components: alphabet of restrictive labels; formal security conditions (semantics security); security condition checking (enforcing mechanism) mechanism; and implementation. The full-fledged implementation of the instrumentation in an information system implies coverage of all four components.

The difficulty of describing security policy at the application and special software level is due to two factors: firstly, the need to take into account the access control rules implemented at the system level, and secondly, the need to take into account additional restrictions, usually checked directly in the program code. Studies devoted to the description of security policy at the level of program code can be found in [2].

Most of the studies concerning computational security (IFC) [3–6], take the concept of information noninterference as a basis. In the case of software, the essence is reduced to verifying the absence of influence of sensitive (untrusted—in the context of integrity control) input values on sensitive (trusted—in the context of integrity control) output values. The necessary formal definitions will be presented below. The literature also describes other approaches to the definition of computational security: staticity of intruder knowledge [7], non-inference [8], and others.

Verification methods are most often realized at the language level in the form of separate types of static (dynamic) analysis. The methods of static analysis of information flows based on safe (secure type system) type systems have gained special popularity [9, 10]. They enable the principle of safe composition to be applied which is essential in view of the large size of the verified code of industrial applications.

Well-known implementations of IFC include JIF [11], Joana [12], and Paragon [13].

A comprehensive review of the results already achieved in this subject area is presented in [2–4].

Despite its long history, IFC in software remains only a subject of academic research. The reason is the complexity of procedures for marking source code (source code markup) with safety (security) labels and interpreting the results obtained (warnings).

1. METHODOLOGICAL BASIS

The hypothetical basis of this article is that in order to transfer the research of instrumentation methods (IFC) into practice, the function of analyzing

[1] GOST R ISO/IEC 15408-1-2002. State Standard of the Russian Federation. *Information technology. Security techniques. Evaluation criteria for IT security*. Part 1. Moscow: IPK Izdatelstvo standartov; 2002 (in Russ.).

[2] Order of the Federal Service for Technical and Export Control of Russia No. 31 dated March 14, 2014. “On Approval of Requirements for Information Protection in Automated Control Systems for Production and Technological Processes at Critically Important Facilities, Potentially Hazardous Facilities, as well as Facilities Presenting an Increased Risk to Human Life and Health and to the Natural Environment” (in Russ.). https://fstec.ru/dokumenty/vse-dokumenty/prikazy/prikaz-fstek-rossii-ot-14-marta-2014-g-n-31. Accessed March 12, 2023.

[3] GOST R 56939-2016. National Standard of the Russian Federation. *Information protection. Secure software development. General requirements*. Moscow: Standartinform; 2018 (in Russ.).

[4] GOST R 51583-2014. National Standard of the Russian Federation. *Information protection. Sequence of protected operational system formation. General provisions*. Moscow: Standartinform; 2018 (in Russ.).

[5] Legal trajectories and environments for information dissemination mean the trajectories and environments envisioned by the system developer for user access to data, as well as data exchange between users and individual system components.

[6] Fuzzing testing is a software testing technique of giving incorrect, unexpected, or random data as input to an application.

information flows with reference to access restrictions set in the system needs to be positioned at a separate stage of the software development life cycle. This can be achieved by turning to the theory of formal verification of software properties.

Before proceeding to a description of the proposed technology, we note that the idea of using the theory of formal program verification in the field of instrumentation (IFC) is not new. Significant results were obtained by Clarkson et al. [14]. In particular, the authors extended the apparatuses of linear temporal logic (LTL) and branching time logic (computational tree logic, CTL) by quantizers (quantifiers) over computation trajectories. As a result, properties of information flow security were formulated (based on the notion of information non-influence noninterference), which, in fact, are hyperproperties. The papers also outline an approach to verifying these properties using model playback (model checking tools) tools. The essence of the approach involves a description of the Kripke structure for the program to be verified, description of security properties using HyperLTL formulas—LTL extension and subsequent generation and verification of the automata model.[7] Describing an industrial application in the form of a Kripke structure may in practice be a difficult task. The authors of this study and the corresponding prototype themselves note the impossibility of scaling the application area of the developed toolkit to systems of medium complexity (the number of states does not exceed 1000). In [15], an interesting way of representing safety (security) properties of information flows as standard safety properties is proposed. This method is based on the idea of transforming the program being checked using an own composition and certain standard (inference rules) rules of inference of the system of safe types (secure type system). The limitations here include a certain difficulty in interpreting the verification results, since the procedure requires modification of the original program, and the remaining labor-intensity of analyzing programs with a large number of states.

In order to overcome the limitations of formal verification, the present study also borrows the previously mentioned approach based on secure type systems. However, in contrast to [14] and [15], a transition from operational semantics to a simplified abstract semantics of information flows is proposed for modeling computations. In addition, we introduce a restriction related to developers' compliance with the principle of minimizing the attack surface. This, inter alia, implies compact storage of sensitive information (in a limited set of tables) and the desire to clearly separate critical and non-critical services of the system. The physical separation of services based on the levels of confidentiality and integrity of processed data is now becoming possible. This is due to the rejection of monolithic architecture of business applications and the wide development of platforms supporting modular development. The type inference rules in our study are assumed to be replaced by rules of abstract information flow semantics. The rejection of operational semantics in favor of abstract semantics in modeling computations allows a significant reduction in the number of states of the automata model to be achieved, and the security properties of information flows in the form of standard reliability (safety) properties to be formulated. All this enables a mechanism for checking software security properties (in the sense of information flows) to be implemented on the basis of widely used in practice tools for creating program specifications and playing TLA+ and TLC models (TLA checker).

In this study, an intruder is defined as any system user who is not an administrator. The intruder knows the source code, can interact with the program, and has access to output values generated at all stages of execution and possessing a security label corresponding to his own label. We also believe that the intruder is unable to exploit hidden probabilistic information channels and time channels.

2. METHOD OF INFORMATION FLOW SECURITY ANALYSIS

As already mentioned, the purpose of the technology developed is to detect and eliminate prohibited information flows in the software of enterprise-level automated information systems. The previously mentioned MAC (IFC) implementations mainly rely on the system of safe types (secure type systems) and static analysis (static analysis is a separate compilation step). Safe (secure) types in such platforms define security policy rules along with rules of data conversion and restrictions on the amount of allocated memory dictated by standard types. For example, an extended variable type might look like this: int x {Alice −> Bob}. This means that the owner of the data stored in variable x is Alice, and reading is allowed to Alice, Bob, and any other users acting on their behalf. Thus, the known IFC implementations already mentioned assume that the software developer is assigned additional functions of marking up the source code and interpreting security warnings generated by the static analyzer. The proposed technology is based on the idea of automatic generation of specifications based on the source code of database (DB) program blocks (services) with their subsequent verification by security specialists—analysts [16]. The stages of the analysis are presented in Fig. 1.

[7] In these studies, Büchi automata are taken as a basis.

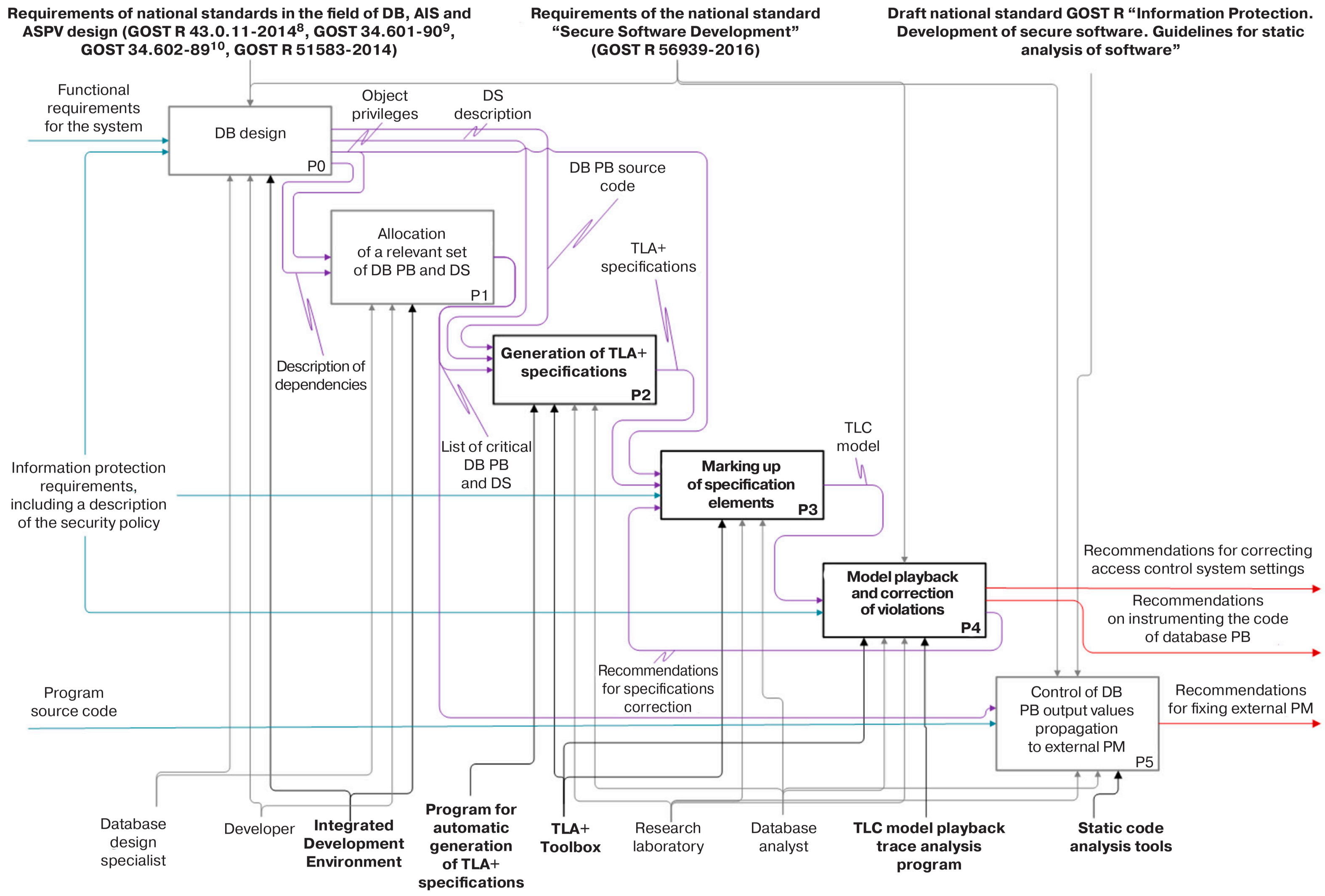
Requirements of national standards in the field of DB, AIS and ASPV design (GOST R 43.0.11-2014[8], GOST 34.601-90[9], GOST 34.602-89[10], GOST R 51583-2014)
Requirements of the national standard "Secure Software Development" (GOST R 56939-2016)
Draft national standard GOST R "Information Protection. Development of secure software. Guidelines for static analysis of software"
Functional requirements for the system
DB design
P0
Object privileges
DS description
DB PB source code
Allocation of a relevant set of DB PB and DS
P1
TLA+ specifications
Description of dependencies
Generation of TLA+ specifications
P2
List of critical DB PB and DS
TLC model
Information protection requirements, including a description of the security policy
Marking up of specification elements
P3
Model playback and correction of violations
P4
Recommendations for correcting access control system settings
Recommendations on instrumenting the code of database PB
Recommendations for specifications correction
Program source code
Control of DB PB output values propagation to external PM
P5
Recommendations for fixing external PM
Database design specialist
Developer
Integrated Development Environment
Program for automatic generation of TLA+ specifications
TLA+ Toolbox
Research laboratory
Database analyst
TLC model playback trace analysis program
Static code analysis tools

Fig. 1. Stages of analysis. PM—program module, PB—program block, DS—data source, AIS—automated information system, ASPV—automated system in a protected version

[8] GOST R 43.0.11-2014. National Standard of the Russian Federation. *Informational ensuring of equipment and operational activity. Database in technical activities*. Moscow: Standartinform; 2018 (in Russ.).

[9] GOST 34.601-90. Interstate Standard. *Information technology. Set of standards for automated systems. Automated systems. Stages of development*. Moscow: Standartinform; 2009 (in Russ.).

[10] GOST 34.602-89. Interstate Standard. *Information technology. Set of standards for automated systems. Technical directions for automated system making*. Moscow: Standartinform; 2009 (in Russ.).

The stored database program blocks (*PL/SQL* blocks) in the context of technology represent a convenient mechanism for implementing business logic. These modules, as a rule, are characterized by small code size, absence of redundant calculations, and focus on working with data.

A subset of the *PL/SQL*, language is used in the description of the technology, and its BNF-grammar[11] is presented below:

(values)	$v ::=$	$n \mid b$
(declarations)	$d ::=$	x **number** $\mid x_1\ x_2 \mid d_1;d_n \mid$ **type** x_r **is object** $(x_{f1}\ x_1,\dots x_{fn}\ x_n) \mid$ **type** x_t **is table of** $x \mid$ **exception** $x \mid$ **procedure** $x_p(x_1,\dots x_n)$ **as** d **begin** c_1 **[exception]** c_2 **end;** $\mid$ **function** $x_{fn}(x_1,\dots x_n)$ **return** x_{rt} **as** d **begin** c_1 **[exception** c_2**] end;** $\mid\dots$
(expressions)	$e ::=$	$v \mid x \mid x_1.x_2 \mid e_1 \odot e_2 \mid x(e_1,\dots e_2) \mid \dots$
(conditions)	$cnd ::=$	$e_1 * e_2 \mid cnd_1 \circledast cnd_2$
(statements)	$c ::=$	$x := e \mid c_1;c_2 \mid x_f(x_1 \rightarrow e_1,\dots x_n \rightarrow e_n) \mid$ **if** e **then** c_1 **else** $c_2 \mid$ **while** e **do** $c \mid$ **end if** $\mid$ **end while** $\mid c_1 \vee c_2 \mid$ **select** $e_1,\dots e_n$ **into** $x_1,\dots x_n$ **from** $x_{t1},\dots x_{tn}$ **where** $cnd \mid$ **insert into** $x_t\ (x_1,\dots x_n)$ **values** $(e_1,\dots e_n) \mid$ **update** x_t **set** $x_1 = e_1,\dots x_n = e_n$ **where** $cnd \mid$ **delete from** x_t **where** $cnd \mid$ **throw** $x_{exc} \mid$ **when** x_{exc} **then** $c \mid$ **null** $\mid$ **return**$(x_{out} \rightarrow e) \mid\dots$
(program)	$p ::=$	**Declare** d **begin** c_1 **[exception** c_2**] end;**

2.1. Database design

The first stage of analysis is database design. It should be performed in such a way that confidential data is placed compactly—in a limited set of tables. This requirement does not contradict the generally accepted principles of secure development, while at the same time allows us to isolate critical calculations from the general *PL/SQL* code more effectively.

2.2. Allocation of a relevant set of database program blocks and data sources

The next step is to select *PL/SQL* procedures and functions which implement critical computations, i.e., computations over confidential data and data requiring a high level of trust. An integral function of modern databases is the management of direct and transitive dependencies.[12, 13, 14] They are used by the system kernel to check the states of objects before their calls and to avoid critical errors at runtime.[15] Such mechanisms work in approximately the same way. In *Oracle* DBMS, in order to obtain the direct and indirect dependencies associated with some table T, the following commands may be executed:

execute deptree_fill('TABLE', 'T');
select * **from** deptree.

The next three steps: generation, markup, and application of the specification validation algorithm, are the key and most labor-intensive ones.

2.3. Generation of TLA+ specifications

Let *PC* be the security label of the instruction counter (program counter). It defines the implicit information flows occurring in conditional **if** statements and **while** loops; *c* is the current instruction in the process associated with the user session; *M* is the abstract state of the computing environment, which defines the mapping of variables (local and global), input and output flows to their corresponding restrictive labels; *n* is the total number of user sessions. Then the execution of program blocks in the DBMS environment can be described by the system of state transitions (state transition system) in the following form:

$$\left\langle \left\langle \left\langle PC1, c1 \right\rangle \dots \left\langle PCn, cn \right\rangle \right\rangle, M \right\rangle.$$

The generation of specifications describing the behavior of such a system is performed using the "*Generation of TLA+ specifications based on database program blocks*" software tool, in accordance with the abstract semantics of information flows [2].

As an example, consider the rule of calculating an abstract expression and the assignment rule. The result of an abstract expression $\odot$, is calculated as the minimum (least) upper bound of the labels of the operands included in it:

[11] Backus–Naur formisa formal system for describing syntax.

[12] https://docs.oracle.com/cd/A84870_01/doc/server.816/a76965/c19depnd.htm. Accessed March 12, 2023.

[13] https://www.postgresql.org/docs/current/catalog-pg-depend.html. Accessed March 12, 2023.

[14] https://learn.microsoft.com/en-us/sql/ssms/object/object-dependencies?view=sql-server-ver16.AccessedMarch12, 2023.

[15] Database program environment is not monolithic, and in the process of database instance operation separate objects: tables, program blocks, views, etc., may undergo changes that are critical for the operation of other dependent objects.

$$\text{(E-OPER)}\frac{\langle e1, M\rangle \Downarrow p1 \quad \langle e2, M\rangle \Downarrow p2}{\langle e1 \odot e2, M\rangle \Downarrow p1 \sqcup p2}.$$

The resulting labels for comparison expressions * and logical expressions ⊛ are calculated in the same way. As a result of the assignment operation, the abstract state of the computing environment changes according to the label of the value to be assigned and the label of the instruction counter in the current session:

$$\text{(C-ASSIGN)}\frac{\langle e,M\rangle\Downarrow p}{\begin{gathered}\langle\langle\langle\ldots\rangle\ldots\langle PC,x{:=}e\rangle\ldots\langle\ldots\rangle\rangle,M\rangle\rightarrow\\ \rightarrow\langle\langle\langle\ldots\rangle\ldots\langle PC,\text{null}\rangle\ldots\langle\ldots\rangle\rangle,M[x\mapsto p\sqcup pc1\ldots pcn]\rangle\end{gathered}}.$$

Rules of type EXT are used to check information flows associated with the elements-stocks possessing stationary security labels. These include output streams, attributes of relations, which can be accessed outside the context of calls of the checked program blocks. *Inv* conditions correspond to the security (safety) invariant:

$$\text{(C-ASSIGN-EXT)}\frac{\langle e,M\rangle\Downarrow p \quad \langle x,M\rangle\Downarrow px \quad \text{inv: } p\sqcup pc1\ldots pcn\sqsubseteq px}{\begin{gathered}\langle\langle\langle\ldots\rangle\ldots\langle PC,x{:=}e\rangle\ldots\langle\ldots\rangle\rangle,M\rangle\rightarrow\\ \rightarrow\langle\langle\langle\ldots\rangle\ldots\langle PC,\text{null}\rangle\ldots\langle\ldots\rangle\rangle,M\rangle\end{gathered}}.$$

Transitions between parallel sessions are represented as equally probable. This is a fair assumption for the chosen intruder model and the corresponding information non-influence (noninterference) scheme. They are described by global rules in the following form:

$$\text{(GLOB-1)}\frac{O(d)=ci \qquad \langle PCi,ci,M\rangle\rightarrow\langle PCk,ck,M'\rangle}{\langle\langle\langle PC1,c1\rangle\ldots\langle PCn,cn\rangle\rangle,M\rangle\underset{1/n}{\overset{i}{\rightarrow}}\langle\langle\langle PC1,c1\rangle\ldots\langle PCi-1,ci-1\rangle\langle PCk,ck\rangle\ldots\langle PCn,cn\rangle\rangle,M'\rangle},$$

$$\text{(GLOB-2)}\frac{O(d)=ci \qquad \langle PCi,ci,M\rangle\rightarrow\langle PCi,M\rangle}{\langle\langle\langle PC1,c1\rangle\ldots\langle PCn,cn\rangle\rangle,M\rangle\underset{1/n}{\overset{i}{\rightarrow}}\langle\langle\langle PC1,c1\rangle\ldots\langle PCi-1,ci-1\rangle\langle PCi+1,ci+1\rangle\ldots\langle PCn,cn\rangle\rangle,M\rangle}.$$

2.4. Layout (Marking up) of specification elements

Specification markup is performed on the basis of the described security policy requirements and valid access restrictions. In order to perform this step, the modified language for describing security policies *Paralocks* [13] is used. Its main advantages include the ability to embed data declassification conditions into policy expressions (or labels), flexibility, and the ability to integrate with various access control systems (role-based, mandate-based, mandatory, etc.). Arguments regarding the necessity of interfacing the IFC mechanisms implemented at the level of special (application) software and system-level access control mechanisms, as well as a comparative analysis of known security policy description languages that justify the choice of *Paralocks*, are given in [2].

Formally, the security label P_k is defined by a first-order predicate logic formula of the following form: $P_k \triangleq C_1 \wedge C_2 \wedge \ldots$, here C_n is a separate expression of information flow admissibility: $\forall x_1,\ldots,x_m.l_1(\cdot)\wedge l_2(\cdot)\wedge l_3(\cdot)\ldots\Rightarrow Flow(u)$, $Flow(u)$ is a predicate denoting the data flow to u (u is a bound variable or constant denoting a user), $l_1 \ldots l_n$ are conditions (or locks) whose fulfillment is required for $Flow(u)$ to be true. Predicates $l_1 \ldots l_n$ can be parametric or non-parametric. The set of possible locks depends on the access restrictions set at the system level and additional restrictions realized by the application software.

As an example, let us consider the policy requirement: “A flow to an arbitrary user *x* is possible if: (a) the *guest* role is set for *x* and the *w_hours* lock is open—the attempt is made during business hours, or (b) the *account* role is set for *x*.” Logically it can be represented as:

$$\begin{gathered}\forall x.\left(w\_hours \wedge guest(x) \Rightarrow Flow(x)\right)\wedge\\ \wedge\left(account(x)\Rightarrow Flow(x)\right).\end{gathered}$$

For the convenience of analyzing the traces leading to the occurrence of prohibited information flows, a separate software tool with a graphical user interface was developed: “*Analysis of TLC model playback traces built on the basis of TLA+ specifications of database program blocks and leading to the violation of the invariant of information flows security*.” Labels are displayed in accordance with the simplified notation, in which the example under consideration appears as follows:

$$\boxed{\begin{array}{l}\text{x : account(x)}\\ \text{x : guest(x), t\_expire}\end{array}}$$

P_1	P_2	$P_1 \sqsubseteq P_2$
x: manager(x)	x: reviewer(x)	FALSE
x: reviewer(x)	x: manager(x)	FALSE
x: manager(x)	x: manager(x), t_expire	TRUE
x: manager(x)	bob: manager(bob)	TRUE

P_1	P_2	$P_1 \sqsubseteq P_2$
x: manager(x)	x: reviewer(x)	x: manager(x)
	x: reviewer(x)	alice: reviewer(alice)
x: manager(x) x: reviewer(x)	x: t_expire	x: manager(x), t_expire x: reviewer(x), t_expire
bob	alice: t_expire	T

Fig. 2. Work of the label comparison operator and calculation of the minimum upper boundary

A partial order relation $\sqsubseteq$ on the set of security labels is defined as follows:

$$P_1 \sqsubseteq P_2, \text{ if } \forall c_2 \in P_2 : \exists c_1 \in P_1 : c_1 \sqsubseteq c_2. \quad (1)$$

From a logical point of view $P_1 \sqsubseteq P_2$ can be interpreted as

$$P_1 \vDash P_2. \quad (2)$$

In [17] it is shown that condition (1) is necessary and sufficient for the truth of expression (2). The comparison of security label proposals in [17] is described algorithmically, through a set of rules.

Figure 2, using simplified notation, shows several examples of how the comparison operator works on a set of labels and calculates the least upper bound.

The confidentiality level of the data used by the program may not only decrease, but also increase in the process of their processing. In order to take this into account, the *Paralocks* language syntax is extended with a one-parameter *Unknown* lock. It permits rules of classification to be specified by content (*what*)[16] for separate elements. *Unknown* lock can be used in the left part (premise) of a policy clause of some element, if in the course of calculations the policy of the element should become stricter when some additional information is disclosed.[17] For example, a policy of the form "Salary (of an employee) can be read only by a specialist of the financial department" provided that the attributes identifying the employee are: *emp_id*, *email*, *lname*, can be expressed as:

$$\forall x.(Unknown(emp\_id) \wedge$$
$$\wedge\, Unknown(email) \wedge Unknown(lname) \Rightarrow$$
$$\Rightarrow Flow(x)) \wedge account\_emp(x) \Rightarrow Flow(x).$$

Using TLA+ formulas, finite sets of possible proposals of security policies (labels) and policies themselves can be defined.[18] Using the TLA PS (proof system), we prove that the set of policies with a partial order relation defined on it forms a complete algebraic lattice.

2.5. Model playback and correction of irregularities of the safety invariants

The property of progress-dependent information noninterference (PDIN) [3] is adopted as a formal condition of computational security. In contrast to strict information noninterference, intermediate states are subject to verification. No restrictions are imposed on the values of internal variables (inaccessible for observation).

Definition 1. *Program P satisfies the progress-dependent information noninterference property for initial and final mappings of a set of variables to a set of security labels M_1 and M_2—*$\mathrm{PDIN}(P)_{M_1,M_2}$*, if for any two states S_1 and S_2 of the computation environment consisting of a low equivalence relation with respect to some security level L at mapping M_1:* (a) *each computation step is accompanied by the generation of the same observed values with respect to level L or leads to a divergence for both states;* (b) *corresponding final*

[16] It is assumed that data classification is characterized by the same aspects: *when*, *who*, *what*, *where*, as for declassification.

[17] It is also allowed the approach when the classification conditions are not taken into account, i.e., the strictest policy $(account\_emp(x) \Rightarrow Flow(x))$ is initially selected for the element. At the same time, in case of false positives (when there are no grounds for classification), the reverse declassification procedure is applied.

[18] https://github.com/timimin/plif. Accessed March 12, 2023.

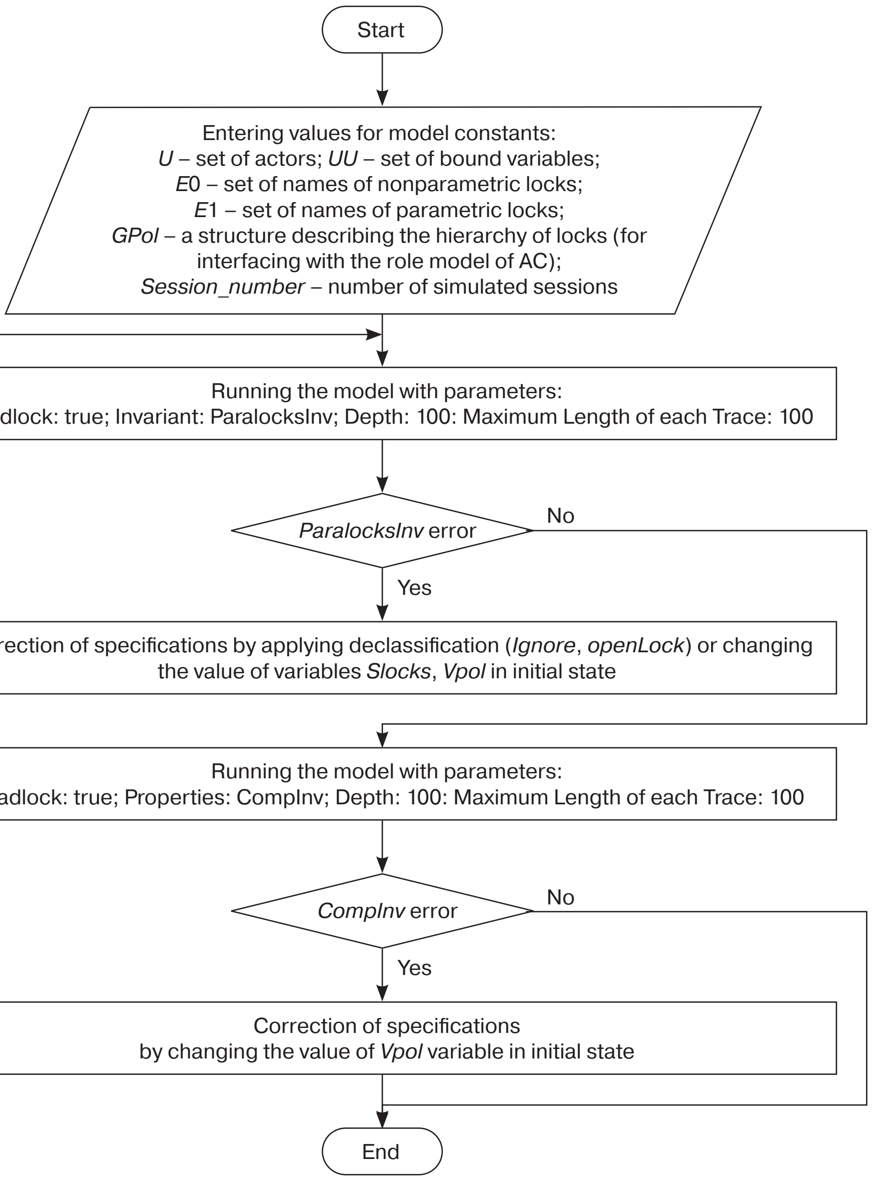
Start
Entering values for model constants:
U – set of actors; UU – set of bound variables;
E0 – set of names of nonparametric locks;
E1 – set of names of parametric locks;
GPol – a structure describing the hierarchy of locks (for interfacing with the role model of AC);
Session_number – number of simulated sessions
Running the model with parameters:
Deadlock: true; Invariant: ParalocksInv; Depth: 100: Maximum Length of each Trace: 100
ParalocksInv error
No
Yes
Correction of specifications by applying declassification (Ignore, openLock) or changing the value of variables Slocks, Vpol in initial state
Running the model with parameters:
Deadlock: true; Properties: CompInv; Depth: 100: Maximum Length of each Trace: 100
CompInv error
No
Yes
Correction of specifications by changing the value of Vpol variable in initial state
End

Fig. 3. Specification verification algorithm. AC—access control

states S_1' and S_2' of the computation environment consist of a low equivalence relation with respect to some security level L at mapping M_2:

$$\mathrm{PDIN}(P)_{M_1,M_2} \triangleq \forall L, S_1, S_2, o_1, o_2 : S_1 \eqsim_{M_1,L} S_2 \wedge$$
$$\wedge\, P(S_1)\downarrow\left\langle S_1', o_1\right\rangle \wedge P(S_2)\downarrow\left\langle S_2', o_2\right\rangle \Rightarrow$$
$$\Rightarrow o_1 \approx_{L(d)} o_2 \wedge S_1' \eqsim_{M_2,L} S_2'.$$

The expression $o_1 \approx_{L(d)} o_2$ denotes the equivalence of the observed behaviors with respect to the level L (taking into account the declassification [18]). Two states of the computation environment are assumed to be in a low equivalence relation with respect to a given privacy level L, if all pairs of similarly labeled elements with label not exceeding L have the same values. Condition (b) is easily verified for individual expressions and *PL/SQL* commands using the abstract semantics rules defined for them, and is important for formal proof of the security of computation in general under flow-sensitive conditions and the absence of restrictions on the number of executable blocks in one session.

Algorithm. The preliminary step of the algorithm (Fig. 3) defines constants which specify the number of simulated sessions, the set of user names, the set of user (bound) variables, the set of single-parameter locks, and the set of non-parameter locks, etc. When the model is replayed, the main security invariant *ParalocksInv* (guaranteeing that condition (a) of Definition 1 is met for individual commands and expressions) and the action property *CompInv* (guaranteeing that the initial and final mappings of the set of global variables to the set of labels are equivalent). If *ParalocksInv* is violated, a data declassification procedure may be applied. It usually implies code correction or changing the privileges of accessing database objects. In the event of an error caused by the violation of the *CompInv* property, the label of the corresponding global variable—table column—is incremented in the initial state.[19] Bringing the labels of global variables to stationary values allows the fulfillment of condition (a) of Definition 1 to be verified for individual *PL*/*SQL* commands and expressions for all possible initial and final abstract states M_1 and M_2. This, in turn, is required to prove the correctness of the computation model checking algorithm.

Proof of convergence of the algorithm and fulfillment of security conditions of infinite computations in an unlimited number of user sessions in accordance with the definition of PDIN at successful completion of the algorithm is given in [5].

2.6. Control of distribution of output values of verified database program blocks in external program modules

Control of the propagation of output values of verified procedures and functions in application software is performed using standard taint tracking analysis by means of tools such as *CodeQL* [19].

CONCLUSIONS

The key stages of the technology herein presented have been tested on training examples. Despite a certain loss of analysis accuracy, due to abstraction inevitable for formal verification, the possibility of graphical interpretation of problematic calculation traces using a utility entitled “*Analysis of TLC model playback traces built on the basis of TLA+ specifications of database program blocks and leading to violation of information flows security invariant*” developed with the author’s participation significantly simplifies the task of identifying false alarms.

The development of comprehensive methodological recommendations on the application of data reclassifi cation (declassification and classification) procedures is a promising area for further research. At the present time, the general principles of reclassification have been formulated and a number of schemes have been proposed. However, they require adaptation to the described technology. It is recommended that separate research on the last stage of the proposed procedure be conducted.

[19] Fulfillment of the *CompInv* property can be achieved in a finite number of iterations because the policy alphabet is a finite lattice and the transition functions of the global variable policy computation are monotonically increasing.

REFERENCES

1. Devyanin P.N., Telezhnikiv V.I., Khoroshilov A.V. Building a methodology for secure system software development on the example of operating systems. *Trudy Instituta sistemnogo programmirovaniya RAN = Proceedings of the Institute for System Programming of the RAS* (*Proceedings of ISP RAS*). 2021;33(5):25–40 (in Russ.). https://doi.org/10.15514/ISPRAS-2021-33(5)-2
2. Timakov A.A. Information flow control in software DB units based on formal verification. *Program. Comput. Soft*. 2022;48(4):265–285. https://doi.org/10.1134/S0361768822040053
3. Hedin D., Sabelfeld A. A Perspective on Information-Flow Control. In: *Software Safety and Security*. 2012;33:319–347. https://doi.org/10.3233/978-1-61499-028-4-319
4. Kozyri E., Chong S., Myers A.C. Expressing Information Flow Properties. *Foundations and Trends® in Privacy and Security*. 2022;3(1):1–102. http://doi.org/10.1561/3300000008
5. Volpano D., Smith G. Probabilistic noninterference in a concurrent language. *Journal of Computer Security* (*JCS*). 1999;7(2):231–253. http://doi.org/10.3233/JCS-1999-72-305
6. Sabelfeld A., Sands D. Probabilistic noninterference for multi-threaded programs. In: *Proceedings 13th IEEE Computer Security Foundations Workshop* (*CSFW-13*). 2000. P. 200–214. https://doi.org/10.1109/CSFW.2000.856937
7. Askarov A., Chong S. Learning is Change in Knowledge: Knowledge-Based Security for Dynamic Policies. In: *Proceedings 25th IEEE Computer Security Foundations Symposium* (*CSF 2012*). 2012. P. 308–322. https://doi.org/10.1109/CSF.2012.31
8. Sutherland D. A model of information. In: *Proceedings of the 9th National Security Conference*. 1986. P. 175–183.
9. Volpano D., Irvine C., Smith G. Sound type system for secure flow analysis. *Journal of Computer Security* (*JCS*). 1996;4(2–3):167–187.
10. Mantel H., Sudbrock H. Types vs. PDGs in information flow analysis. In: Albert E. (Ed.). *Logic-Based Program Synthesis and Transformation. The 22nd International Symposium, LOPSTR 2012. Proceedings.* Springer. 2012. P. 106–121. https://doi.org/10.1007/978-3-642-38197-3_8
11. Myers A.C., Liskov B. A decentralized model for information flow control. *ACM SIGOPS Operating Systems Review.* 1997;5:129–142. https://doi.org/10.1145/268998.266669
12. Graf J., Hecker M., Mohr M., Snelting G. Checking applications using security APIs with JOANA. In: *8th International Workshop on Analysis of Security APIs*. *Proceedings.* 2015. P. 118–129.
13. Broberg N., van Delft B., Sands D. Paragon for practical programming with information-flow control. In: Shan C. (Ed.). *Programming Languages and Systems: The 11th Asian Symposium*, APLAS 2013. *Proceedings.* Springer. 2013. P. 217–232. https://doi.org/10.1007/978-3-319-03542-0_16
14. Clarkson M.R., Finkbeiner B., Koleini M., Micinski K.K., Rabe M.N., Sánchez C. Temporal logics for hyperproperties. In: Abadi M., Kremer S. (Eds.). *Principles of Security and Trust: The Third International Conference*, POST 2014. *Proceedings.* Berlin, Heidelberg: Springer; 2014. P. 265–284. https://doi.org/10.1007/978-3-642-54792-8_15
15. Terauchi T., Aiken A. Secure information flow as a safety problem. In: Hankin C., Siveroni I. (Eds.). In: *Static Analysis: The 12th International Static Symposium*, SAS 2005. *Proceedings*. Berlin, Heidelberg: Springer. 2005. P. 352–367. https://doi.org/10.1007/11547662_24
16. Timakov A.A. Scenario of Information Flow Analysis Implementation in PL/SQL Program Units with PLIF Platform. *Program. Comput. Soft.* 2023;49(4):215–231. https://doi.org/10.1134/S0361768823040114

[Original Russian Text: Timakov A.A. Scenario of Information Flow Analysis Implementation in PL/SQL Program Units with PLIF Platform. *Programmirovanie.* 2023;4:215–231 (in Russ.).]

17. Broberg N., Sands D. Paralocks: Role based information flow control and beyond. In: *Proceedings of the Conference Record of the Annual ACM Symposium on Principles of Programming Languages*. 2010. P. 431–444. https://doi.org/10.1145/1706299.1706349
18. Sabelfeld A., Sands D. Declassification: Dimensions and principles. *Journal of Computer Security* (*JCS*). 2009;17(5):517–548. http://doi.org/10.3233/JCS-2009-0352
19. Youn D., Lee S., Ryu S. Declarative static analysis for multilingual programs using CodeQL. *Software: Practice and Experience*. 2023;53(7):1472–1495. https://doi.org/10.1002/spe.3199

About the author

Aleksey A. Timakov, Cand. Sci. (Eng.), Associate Professor, Department of Information Security, Institute of Artificial Intelligence, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: timakov@mirea.ru. Scopus Author ID 57809572100, RSCI SPIN-code 3163-2170, https://orcid.org/0000-0003-4306-789X

Об авторе

Тимаков Алексей Анатольевич, к.т.н., доцент, доцент кафедры информационной безопасности, Институт искусственного интеллекта ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: timakov@mirea.ru. Scopus Author ID 57809572100, SPIN-код РИНЦ 3163-2170, https://orcid.org/0000-0003-4306-789X

Translated from Russian into English by Lyudmila O. Bychkova
Edited for English language and spelling by Dr. David Mossop