Russian Technological Journal.
2024;12(2):16–27

ISSN 2782-3210 (Print)
ISSN 2500-316X (Online)

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

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.

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.

ВВЕДЕНИЕ

Условия причисления автоматизированных систем к определенным классам защиты определяются оценочными стандартами. В настоящее время требования к безопасности формулируются в терминах «Общих критериев оценки защищенности информационных технологий»[1] и включают функциональные требования и требования доверия [1]. Из анализа нормативных документов[2, 3, 4] следует, что существуют три важных категории условий, учитываемых при определении класса защиты: контроль «легальных» траекторий и сред распространения информации[5], контроль скрытых каналов, формальное доказательство эффективности реализованных механизмов защиты или безопасности вычислений.

На прикладном уровне проверка безопасности вычислений представляется наиболее сложной. На практике полноценное решение этой проблемы не достигнуто даже в контексте контроля «легальных» траекторий распространения информации. Траектории распространения информации в программном обеспечении (ПО), которые соответствуют правилам, предусмотренным его логикой, можно получить из графа потока управления при условии целостности потока управления. Для обеспечения конфиденциальности и целостности данных в ПО используется сочетание формальных, полуформальных и неформальных методов, которые применяются на разных стадиях разработки системы. Некоторые из этих методов включают динамический и статический анализ кода, символьное (косимвольное) выполнение, формальную верификацию и фаззинг-тестирование[6]. Кроме того, на уровне платформы и компилятора применяются дополнительные защитные меры, такие как рандомизация смещений динамической памяти, защита от исполнения на стеке, контроль целостности потока управления и др. Хотя проверки, реализуемые с использованием перечисленных методов, играют существенную роль в обеспечении защиты информации, они в основном не учитывают специфику обрабатываемых данных и бизнес-логику приложения. К формальным методам, которые учитывают специфику данных, относятся методы на основе контроля информационных потоков (КИП).

[1] ГОСТ Р ИСО/МЭК 15408-1-2002. Государственный стандарт Российской Федерации. *Информационная технология. Критерии оценки безопасности информационных технологий*. Часть 1. М.: ИПК Издательство стандартов; 2002. [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] Приказ ФСТЭК России от 14.03.2014 г. № 31. Об утверждении требований к обеспечению защиты информации в автоматизированных системах управления производственными и технологическими процессами на критически важных объектах, потенциально опасных объектах, а также на объектах, представляющих повышенную опасность для жизни и здоровья людей и для окружающей природной среды». https://fstec.ru/dokumenty/vse-dokumenty/prikazy/prikaz-fstek-rossii-ot-14-marta-2014-g-n-31. Дата обращения 12.03.2023. [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] ГОСТ Р 56939-2016. Национальный стандарт Российской Федерации. *Защита информации. Разработка безопасного программного обеспечения. Общие требования*. М.: Стандартинформ; 2018. [GOST R 56939-2016. National Standard of the Russian Federation. *Information protection. Secure software development. General requirements*. Moscow: Standartinform; 2018 (in Russ.).]

[4] ГОСТ Р 51583-2014. Национальный стандарт Российской Федерации. *Защита информации. Порядок создания автоматизированных систем в защищенном исполнении. Общие положения*. М.: Стандартинформ; 2018. [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] Под «легальными» траекториями и средами распространения информации понимаются траектории и среды, предусмотренные разработчиком системы для получения доступа пользователей к данным, а также обмена данными между пользователями и отдельными компонентами системы. [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] Техника тестирования программного обеспечения, заключающаяся в передаче приложению на вход неправильных, неожиданных или случайных данных. [Fuzzing testing is a software testing technique of giving incorrect, unexpected, or random data as input to an application.]

Принято различать четыре составляющих любой технологии анализа безопасности вычислений на основе КИП в ПО: алфавит ограничительных меток, формальные условия безопасности, механизм проверки условий безопасности, реализация. Полноценное внедрение КИП в информационную систему предполагает охват всех четырех составляющих.

Сложность описания политики безопасности на прикладном уровне и уровне специального ПО обусловлена двумя факторами: необходимостью учитывать правила управления доступом, реализованные на системном уровне, и необходимостью учитывать дополнительные ограничения, которые обычно проверяются непосредственно в коде программы. С исследованиями, посвященными описанию политики безопасности на уровне программного кода, можно познакомиться в [2].

В большей части исследований, касающихся условий безопасности вычислений [3–6], за основу принимается понятие информационного невлияния. Применительно к ПО суть сводится к проверке отсутствия влияния чувствительных (недоверенных – в контексте контроля целостности) входных значений на чувствительные (доверенные – в контексте контроля целостности) выходные значения. Необходимые формальные определения будут представлены далее. В литературе также известны иные подходы к определению безопасности вычислений: статичность знаний нарушителя [7], невыводимость [8] и др.

Механизмы проверки чаще всего реализуются на языковом уровне в виде отдельных видов статического (динамического) анализа. Особую популярность приобрели методы статического анализа информационных потоков на основе безопасных систем типов [9, 10]. Они позволяют применить принцип безопасной композиции, что существенно в виду большого объема проверяемого кода промышленных приложений.

К известным реализациям КИП можно отнести JIF [11], Joana [12], Paragon [13].

Комплексный обзор уже достигнутых результатов в рассматриваемой предметной области представлен в [2–4].

Следует отметить, что, несмотря на долгую историю, КИП в ПО остается лишь предметом академических исследований. Причиной является сложность процедур разметки исходного кода метками безопасности и интерпретации полученных результатов (предупреждений).

1. МЕТОДОЛОГИЧЕСКАЯ ОСНОВА

В работе выдвигается гипотеза о том, что для переноса исследований методов КИП в практическую плоскость требуется вынесение функции анализа информационных потоков с привязкой к заданным в системе ограничениям доступа на отдельный этап жизненного цикла разработки ПО. Добиться этого можно, обратившись к теории формальной верификации свойств ПО.

Перед тем как перейти к описанию предложенной технологии, отметим, что идея использования теории формальной верификации программ в области КИП не является новой. Значимые результаты получены Кларксоном и др. [14]. В частности, авторами расширены аппараты линейной темпоральной логики (linear temporal logic, LTL) и логики ветвящегося времени (computational tree logic, CTL) кванторами над траекториями вычислений. В результате удалось сформулировать свойства безопасности информационных потоков (на основе понятия информационного невлияния), которые, по сути, являются гиперсвойствами. В работах также намечен подход к проверке указанных свойств с применением инструментов проигрывания моделей. Суть подхода сводится к описанию структуры Крипке для проверяемой программы, описанию свойств безопасности с использованием формул HyperLTL – расширения LTL и последующей генерации и проверке автоматной модели[7]. Описание промышленного приложения в виде структуры Крипке на практике представляется трудновыполнимой задачей. Сами авторы исследования и соответствующего прототипа отмечают невозможность масштабирования области применения разработанного инструментария до систем средней сложности (число состояний не превышает 1000). В [15] предлагается интересный способ представления свойств безопасности информационных потоков как стандартных свойств надежности (safety properties). Данный способ основан на идее трансформации проверяемой программы с использованием «собственной композиции» и некоторых стандартных правил вывода системы безопасных типов. К ограничениям здесь следует отнести некоторую сложность интерпретации результатов проверки, поскольку процедура требует модификации оригинальной программы, и сохраняющуюся трудоемкость анализа программ с большим числом состояний.

Чтобы преодолеть ограничения формальной верификации, в настоящем исследовании также заимствован ранее отмеченный подход на основе безопасных систем типов, однако, в отличие от [14] и [15], при моделировании вычислений предлагается переход от операционной семантики к упрощенной абстрактной семантике информационных потоков. Кроме того, вводится ограничение, связанное с соблюдением разработчиками принципа минимизации поверхности атаки, который,

[7] В указанных исследованиях за основу принимаются автоматы Бюхи. [In these studies, Büchi automata are taken as a basis.]

в т.ч. предполагает компактное хранение чувствительной информации (в ограниченном наборе таблиц) и стремление четко разделить критичные и некритичные сервисы системы. Физическое разделение сервисов на основе уровней конфиденциальности и целостности обрабатываемых данных в настоящее время становится возможным благодаря отказу от монолитной архитектуры бизнес-приложений и широкому развитию платформ, поддерживающих модульную разработку. Правила вывода типов в нашем исследовании, как можно предположить, заменяются правилами абстрактной семантики информационных потоков. Отказ от операционной семантики в пользу абстрактной семантики при моделировании вычислений позволяет добиться существенного сокращения числа состояний автоматной модели и сформулировать свойства безопасности информационных потоков в виде стандартных свойств надежности. Все это сделало возможным реализацию механизма проверки свойств безопасности ПО (в смысле информационных потоков) на базе широко применяемых на практике средств создания программных спецификаций и проигрывания моделей TLA+[8] и TLC (TLA checker).

Под нарушителем в настоящем исследовании понимается любой пользователь системы, не являющийся администратором. Нарушителю известен исходный код, он может взаимодействовать с программой, ему доступны выходные значения, генерируемые на всех этапах выполнения и обладающие меткой безопасности, соответствующей его собственной метке. Полагаем также, что злоумышленник не способен эксплуатировать скрытые вероятностные информационные каналы и каналы по времени.

2. ТЕХНОЛОГИЯ АНАЛИЗА БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ПОТОКОВ

Как уже отмечалось, целью разработанной технологии служит выявление и устранение запрещенных информационных потоков в ПО автоматизированных информационных систем уровня предприятия. Упомянутые ранее реализации КИП в основном опираются на систему безопасных типов и статический анализ (статический анализ при этом является отдельным этапом компиляции). Безопасные типы в таких платформах задают правила политики безопасности наряду с правилами преобразования данных и ограничениями на объем выделяемой памяти, диктуемыми стандартными типами. Например, расширенный тип переменной может выглядеть так: int x {Alice –> Bob}. Это означает, что владельцем данных, хранимых в переменной x, является Alice, чтение разрешено Alice, Bob, и любым иным пользователям, действующим от их имени. Таким образом, упомянутые известные реализации КИП предполагают, что на разработчика ПО возлагаются дополнительные функции по разметке исходного кода и интерпретации предупреждений безопасности, генерируемых статическим анализатором. В основу предложенной технологии положена идея автоматической генерации спецификаций на основе исходного кода программных блоков баз данных (сервисов) с последующей их верификацией специалистами в области безопасности – аналитиками [16].

Этапы анализа представлены на рис. 1.

Хранимые программные блоки БД (блоки *PL/SQL*) в контексте технологии представляют собой удобный механизм реализации бизнес-логики. Данные модули, как правило, характеризуются малым объемом кода, отсутствием избыточных вычислений, направленностью на работу с данными.

В описании технологии используется подмножество языка *PL/SQL*, далее представлена его BNF-грамматика[9]:

(values) v ::= $n \mid b$

(declarations) d ::= x **number** | $x_1\, x_2$ | $d_1;d_n$ | **type** x_r **is object** ($x_{f1}\, x_1,...x_{fn}\, x_n$) | **type** x_t **is table of** x | **exception** x | **procedure** $x_p(x_1,...x_n)$ **as** d **begin** c_1 [**exception**] c_2 **end;** | **function** $x_{fn}(x_1,...x_n)$ **return** x_{rt} **as** d **begin** c_1 [**exception** c_2] **end;** | ...

(expressions) e ::= $v \mid x \mid x_1.x_2 \mid e_1 \odot e_2 \mid x(e_1,...e_2)$ | ...

(conditions) cnd ::= $e_1 * e_2 \mid cnd_1 \circledast cnd_2$

(statements) c ::= $x := e \mid c_1;c_2 \mid x_f(x_1 \rightarrow e_1,...x_n \rightarrow e_n)$ | **if** e **then** c_1 **else** c_2 | **while** e **do** c | **end if** | **end while** | $c_1 \vee c_2$ | **select** $e_1,...e_n$ **into** $x_1,...x_n$ **from** $x_{t1},...x_{tn}$ **where** cnd | **insert into** $x_t\,(x_1,...x_n)$ **values** $(e_1,...e_n)$ | **update** x_t **set** $x_1 = e_1,...x_n = e_n$ **where** cnd | **delete from** x_t **where** cnd | **throw** x_{exc} | **when** x_{exc} **then** c | **null** | **return**($x_{out} \rightarrow e$) |...

(program) p ::= **declare** d **begin** c_1 [**exception** c_2] **end;**

[8] TLA – temporal logic of actions, темпоральная логика действий.

[9] Backus–Naur form, форма Бэкуса – Наура – формальная система описания синтаксиса. [Backus–Naur formisa formal system for describing syntax.]

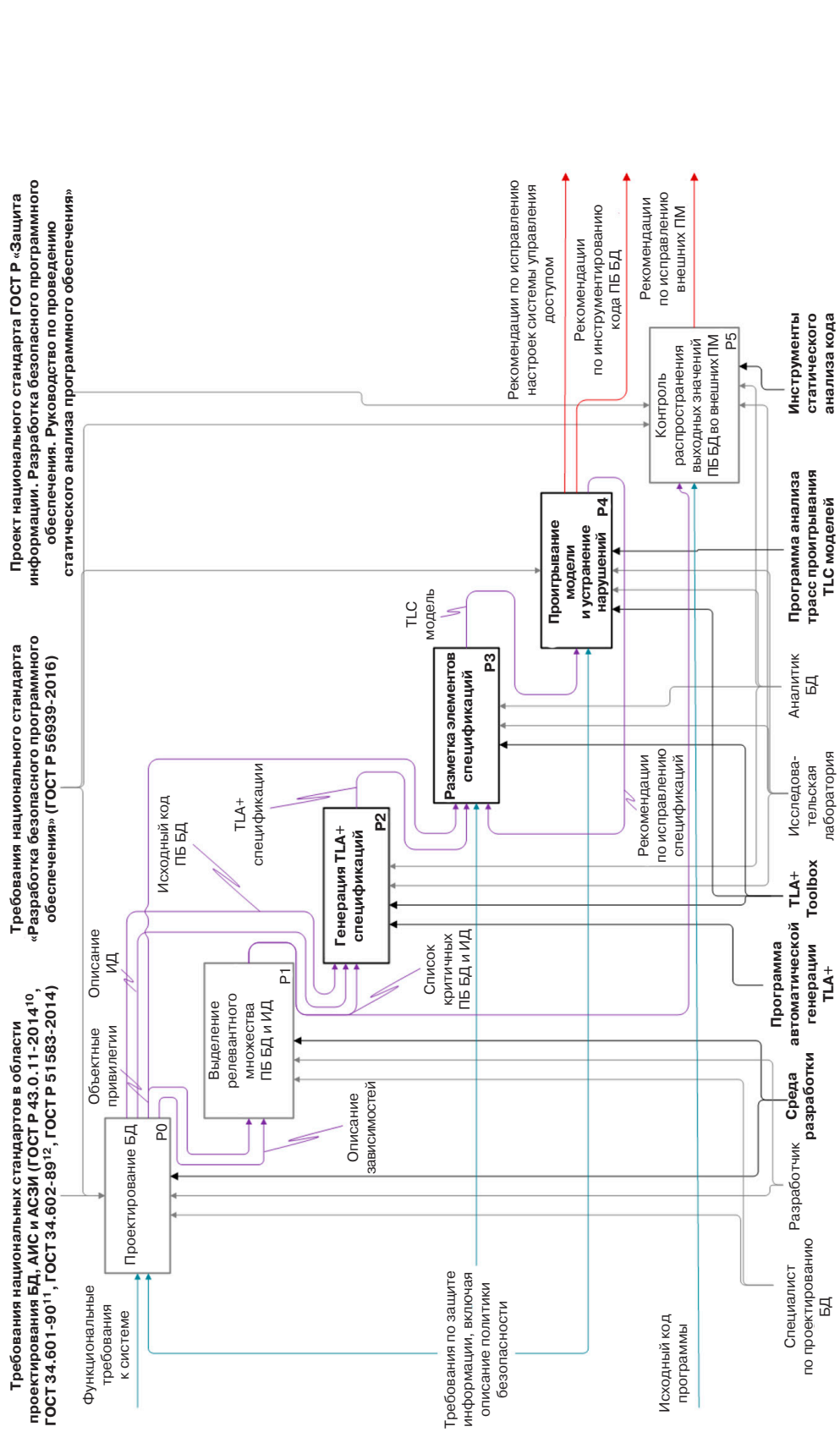

Рис. 1. Этапы анализа. ПМ – программный модуль, ПБ – программный блок, БД – база данных, ИД – источник данных, АИС – автоматизированная информационная система, АСЗИ – автоматизированная система в защищенном исполнении

[10] ГОСТ Р 43.0.11-2014. Национальный стандарт Российской Федерации. *Информационное обеспечение техники и операторской деятельности. Базы данных в технической деятельности.* М.: Стандартинформ; 2018. [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.).]

[11] ГОСТ 34.601-90. Межгосударственный стандарт. *Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания.* М.: Стандартинформ; 2009. [GOST 34.601-90. Interstate Standard. *Information technology: Set of standards for automated systems. Automated systems. Stages of development.* Moscow: Standartinform; 2009 (in Russ.).]

[12] ГОСТ 34.602-89. Межгосударственный стандарт. *Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы.* М.: Стандартинформ; 2009. [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.).]

2.1. Проектирование БД

Первым этапом анализа является проектирование БД. Его необходимо осуществлять таким образом, чтобы конфиденциальные данные размещались компактно – в ограниченном наборе таблиц. Данное требование не противоречит общепринятым принципам безопасной разработки и при этом позволяет более эффективно изолировать критичные вычисления от общего кода *PL/SQL*.

2.2. Выделение релевантного множества программных блоков БД и источников данных

На следующем этапе осуществляется выделение процедур и функций *PL/SQL*, которые реализуют критичные вычисления, т.е. вычисления над конфиденциальными данными и данными, требующими высокого уровня доверия. Стоит отметить, что неотъемлемой функцией современных БД является управление прямыми и транзитивными зависимостями[13, 14, 15]. Они используются ядром системы для проверки состояний объектов перед их вызовами и позволяют избегать критичных ошибок на этапе выполнения[16]. Работают подобные механизмы примерно одинаково. В системе управления базами данных *Oracle* для получения прямых и косвенных зависимостей, ассоциированных с некоторой таблицей T, можно выполнить следующие команды:

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

Следующие три этапа: генерация, разметка, применение алгоритма проверки спецификаций, являются ключевыми и наиболее трудозатратными.

2.3. Генерация TLA+ спецификаций

Пусть *PC* – метка безопасности счетчика команд (program counter), который определяет неявные информационные потоки, возникающие в условных операторах **if** и циклах **while**; *c* – текущая инструкция в процессе, ассоциированном с пользовательским сеансом; *M* – абстрактное состояние среды вычислений, которое задает отображение переменных (локальных и глобальных), входных и выходных потоков на соответствующие им ограничительные метки; *n* – общее количество сеансов пользователей. Тогда выполнение программных блоков в среде системы управления базами данных можно описать системой переходов состояний вида:

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

Генерация спецификаций, описывающих поведение такой системы, осуществляется с помощью программного средства «*Генерация TLA+ спецификаций на основе программных блоков баз данных*» в соответствии с разработанной абстрактной семантикой информационных потоков [2].

В качестве примера рассмотрим правило вычисления абстрактного выражения и правило присваивания. Результат абстрактного выражения $\odot$ вычисляется как минимальная верхняя грань меток включенных в него операндов:

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

Аналогично рассчитываются результирующие метки для выражений сравнения * и логических выражений $\circledast$. В результате выполнения операции присваивания абстрактное состояние среды вычислений изменяется в соответствии с меткой присваиваемого значения и меткой счетчика команд в текущем сеансе:

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

Правила типа «EXT» применяются для проверки информационных потоков, ассоциированных с элементами-стоками, обладающими стационарными метками безопасности. К ним относятся выходные потоки, атрибуты отношений, доступ к которым может быть предоставлен вне контекста вызовов проверяемых программных блоков. Условия *inv* соответствуют инварианту безопасности:

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

Переходы между параллельными сеансами представляются равновероятными, что является справедливым допущением для выбранной модели нарушителя и соответствующей схемы информационного невлияния. Они описываются глобальными правилами вида:

[13] https://docs.oracle.com/cd/A84870_01/doc/server.816/a76965/c19depnd.htm. Дата обращения 12.03.2023. / Accessed March 12, 2023.

[14] https://www.postgresql.org/docs/current/catalog-pg-depend.html. Дата обращения 12.03.2023. / Accessed March 12, 2023.

[15] https://learn.microsoft.com/en-us/sql/ssms/object/object-dependencies?view=sql-server-ver16. Дата обращения 12.03.2023. / Accessed March 12, 2023.

[16] Программная среда БД не является монолитной, и в процессе работы экземпляра БД отдельные объекты: таблицы, программные блоки, представления и др., могут подвергаться изменениям, критичным для работы иных – зависимых объектов. [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.]

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

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

2.4. Разметка элементов спецификаций

Разметка спецификаций осуществляется на основе описанных требований политики безопасности и действующих ограничений доступа. Для выполнения этого шага применяется модифицированный язык описания политик безопасности *Paralocks* [13]. К основным его преимуществам следует отнести возможность встраивания в выражения политик (или меток) условий декласс-ификации данных, гибкость и возможность интеграции с различными системами управления доступом (ролевыми, мандатными и др.). Аргументированные доводы в отношении необходимости сопряжения механизмов КИП, реализуемых на уровне специального (прикладного) ПО, и механизмов управления доступом системного уровня, а также сравнительный анализ известных языков описания политик безопасности, обосновывающий выбор *Paralocks*, приведен в [2].

Формально метка безопасности P_k определяется формулой логики предикатов первого порядка вида: $P_k \triangleq C_1 \wedge C_2 \wedge \ldots$, здесь C_n – отдельное выражение допустимости информационного потока: $\forall x_1, \ldots, x_m . l_1(\cdot) \wedge l_2(\cdot) \wedge l_3(\cdot) \ldots \Rightarrow Flow(u)$, $Flow(u)$ – предикат, обозначающий поток данных к u (u – связанная переменная или константа, обозначающая пользователя), $l_1 \ldots l_n$ – условия (или блокировки), выполнение которых требуется для истинности $Flow(u)$. Предикаты $l_1 \ldots l_n$ могут быть параметрическими или непараметрическими. Множество возможных блокировок зависит от заданных на системном уровне ограничений доступа и дополнительных ограничений, реализуемых прикладным ПО.

В качестве примера рассмотрим требование политики: «Поток к произвольному пользователю x возможен, если: а) для x задана роль *guest* и открыта блокировка *w_hours* – попытка совершается в рабочее время, или б) для x задана роль *account*». Логически оно может быть представлено как:

$$\forall x.\big(w\_hours \wedge guest(x) \Rightarrow Flow(x)\big) \wedge \\ \wedge \big(account(x) \Rightarrow Flow(x)\big).$$

Для удобства анализа трасс, приводящих к возникновению запрещенных информационных потоков, было разработано отдельное программное средство с графическим пользовательским интерфейсом – «*Анализ трасс проигрывания TLC моделей, построенных на основе TLA+ спецификаций программных блоков БД и приводящих к нарушению инварианта безопасности информационных потоков*». Отображение меток в нем происходит в соответствии с упрощенной нотацией, в которой рассматриваемый пример выглядит так:

```
x: account(x)
x: guest(x), t_expire
```

Отношение частичного порядка $\sqsubseteq$ на множестве меток безопасности задается следующим образом:

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

С точки зрения логики $P_1 \sqsubseteq P_2$ можно интерпретировать, как

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

В [17] показано, что условие (1) является необходимым и достаточным для истинности выражения (2). Сравнение предложений меток безопасности в [17] описывается алгоритмически, через набор правил.

На рис. 2 с использованием упрощенной нотации показано несколько примеров работы оператора сравнения на множестве меток и вычисления минимальной верхней грани.

Уровень конфиденциальности используемых программой данных может не только понижаться, но и повышаться в процессе их обработки. Для учета данного обстоятельства синтаксис языка *Paralocks* расширен однопараметрической блокировкой *Unknown* – «Неизвестно». Она позволяет задавать для отдельных элементов правила классификации по содержимому (*what*)[17]. Блокировка *Unknown* может использоваться в левой части (посылке) предложения политики некоторого элемента, если в ходе вычислений при раскрытии некоторых дополнительных сведений политика элемента должна становится

[17] Предполагается, что для классификации данных характерны те же аспекты: *when*, *who*, *what*, *where*, что и для декласс-ификации. [It is assumed that data classification is characterized by the same aspects: *when*, *who*, *what*, *where*, as for declassification.]

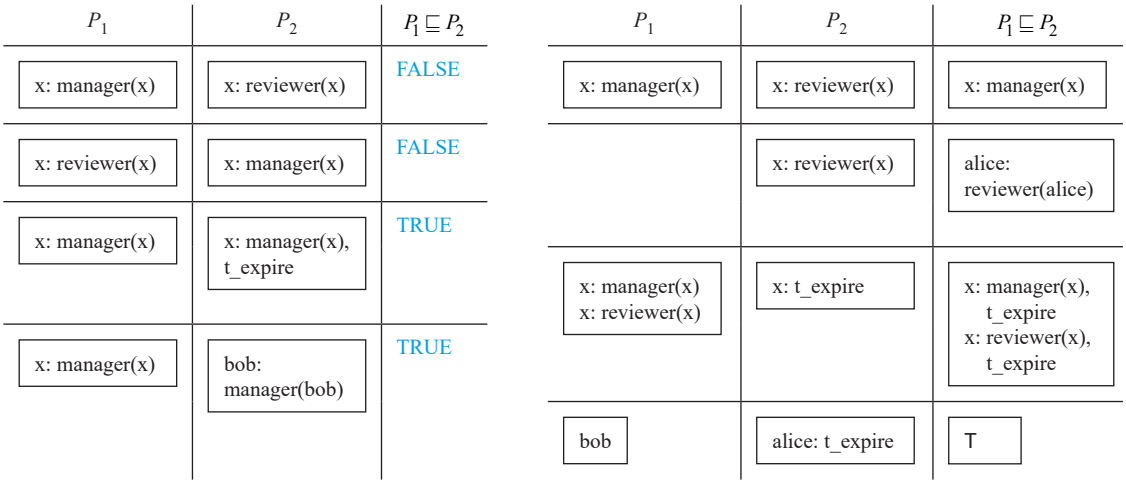

Рис. 2. Работа оператора сравнения меток и вычисления минимальной верхней грани

строже[18]. Например, политика вида «Зарплата (сотрудника) может быть прочитана только специалистом финансового отдела» при условии, что идентифицирующими сотрудника атрибутами выступают: *emp_id, email, lname*, может быть выражена как:

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

С использованием формул TLA+ определены конечные множества возможных предложений политик (меток) безопасности и самих политик[19]. С использованием логической системы вывода TLA PS (proof system) приводится доказательство того, что множество политик с заданным на нем отношением частичного порядка образует полную алгебраическую решетку.

2.5. Проигрывание модели и устранение нарушений инвариантов безопасности

В качестве формального условия безопасности вычислений принято свойство прогресс-зависимого информационного невлияния (ПЗИН) [3]. В отличие от строгого информационного невлияния, проверке подвергаются промежуточные состояния, при этом ограничения на значения внутренних переменных (недоступных для наблюдения) не накладывается.

Определение 1. *Программа P удовлетворяет свойству прогресс-зависимого информационного невлияния для начального и конечного отображений множества переменных на множество меток безопасности* M_1 *и* M_2 – $\text{ПЗИН}(P)_{M_1,M_2}$, *если для любых двух состояний* S_1 *и* S_2 *среды вычислений, состоящих в отношении низкой эквивалентности относительно некоторого уровня конфиденциальности L при отображении* M_1: (а) *каждый шаг вычислений сопровождается генерацией одинаковых наблюдаемых значений относительно уровня L или приводит к «расхождению» для обоих состояний;* (б) *соответствующие финальные состояния* S_1' *и* S_2' *находятся в отношении низкой эквивалентности относительно уровня L при отображении* M_2:

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

Выражение $o_1 \approx_{L(d)} o_2$ означает эквивалентность наблюдаемых поведений относительно уровня L (с учетом деклассификации [18]). Предполагается, что два состояния среды вычислений находятся в отношении низкой эквивалентности относительно заданного уровня конфиденциальности L, если все пары одноименных элементов с меткой, не превышающей L, обладают одинаковыми значениями. Условие (б) легко проверяется для отдельных выражений и команд *PL/SQL* с использованием заданных для них правил абстрактной семантики и является важным для формального

[18] Допускается также подход, при котором условия классификации не учитываются, т.е. для элемента изначально выбирается наиболее строгая политика $(account\_emp(x) \Rightarrow Flow(x))$. При этом в случае ложного срабатывания (когда основания для классификации отсутствуют) применяется обратная процедура деклассификации. [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.]

[19] https://github.com/timimin/plif. Дата обращения 12.03.2023. / Accessed March 12, 2023.

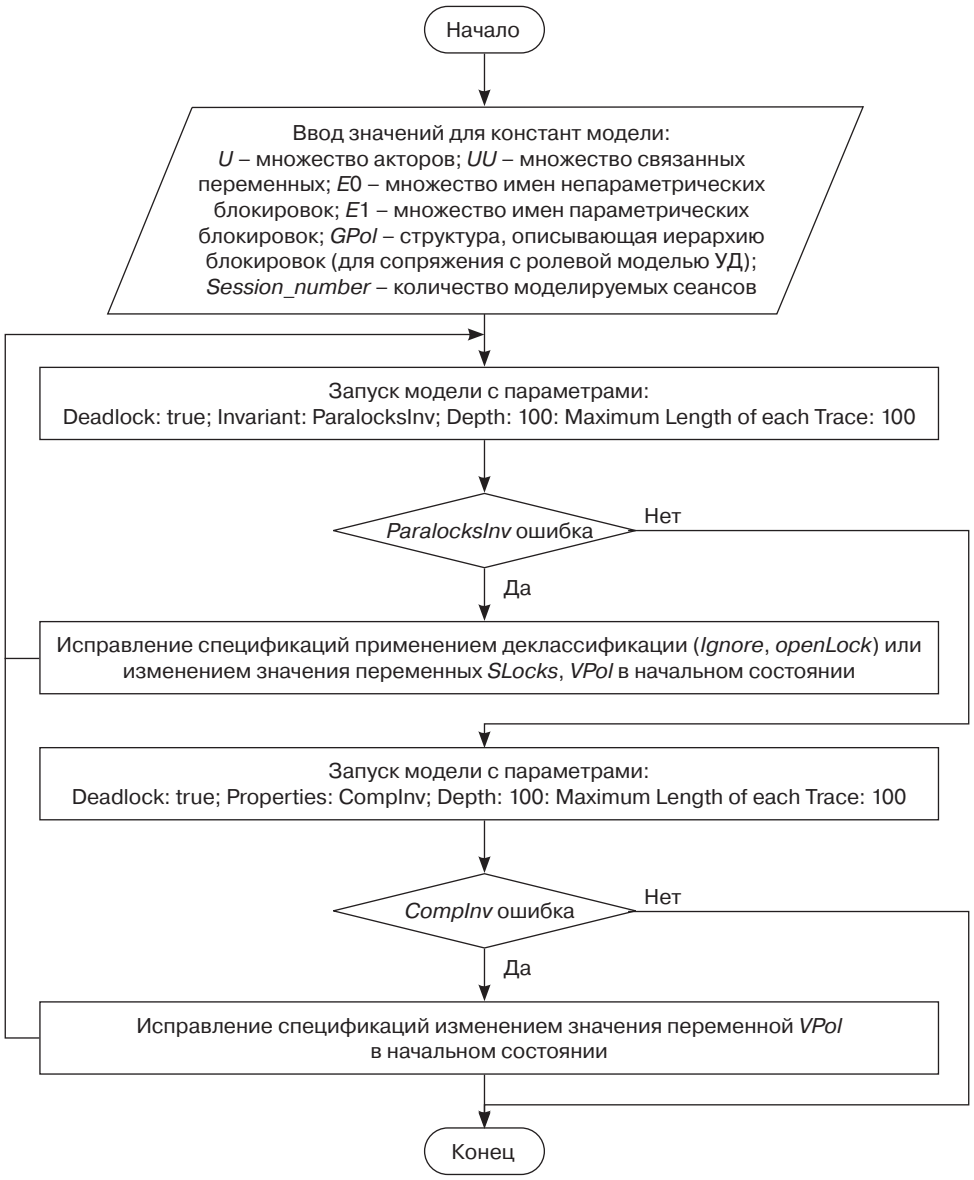

Рис. 3. Алгоритм проверки спецификаций. УД – управление доступом

доказательства безопасности вычислений в целом в условиях потокочувствительности и отсутствия ограничений на количество исполняемых блоков в одном сеансе.

Алгоритм. На предварительном шаге алгоритма (рис. 3) осуществляется определение констант, которые задают количество моделируемых сеансов, множество пользовательских имен, множество пользовательских (связанных) переменных, множество однопараметрических блокировок, множество непараметрических блокировок и т.д. При проигрывании модели проверяется главный инвариант безопасности *ParalocksInv* – гарантирует выполнение условия (а) Определения 1 для отдельных команд и выражений – и свойство перехода *CompInv* – гарантирует эквивалентность начального и конечного отображений множества глобальных переменных на множество меток. При нарушении *ParalocksInv* может применяться процедура деклассификации данных. Она, как правило, предполагает исправление кода или изменение привилегий на доступ к объектам БД. В случае возникновения ошибки, вызванной нарушением свойства *CompInv*, в начальном состоянии повышается метка соответствующей глобальной переменной – столбца таблицы[20]. Приведение меток глобальных переменных

[20] Выполнения свойства *CompInv* можно добиться за конечное количество итераций, поскольку алфавит политик представляет собой конечную решетку, а переходные функции вычисления политик глобальных переменных являются монотонно возрастающими. [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.]

к стационарным значениям позволяет проверить выполнение условия (а) Определения 1 для отдельных команд и выражений *PL/SQL* для всех возможных начальных и конечных абстрактных состояний M_1 и M_2, что, в свою очередь, требуется для доказательства корректности алгоритма проверки модели вычислений.

Доказательство сходимости алгоритма и выполнения условий безопасности бесконечных вычислений в неограниченном количестве пользовательских сеансов в соответствии с определением ПЗИН при успешном завершении алгоритма приведено в [5].

2.6. Контроль распространения выходных значений верифицированных программных блоков БД во внешних программных модулях

Контроль распространения выходных значений верифицированных процедур и функций в прикладном ПО осуществляется с использованием стандартного анализа помеченных данных (taint tracking) инструментальными средствами наподобие *CodeQL* [19].

ЗАКЛЮЧЕНИЕ

Ключевые этапы представленной технологии апробированы на учебных примерах. В условиях некоторой потери точности анализа, за счет неизбежного для формальной верификации абстрагирования, возможность графической интерпретации проблемных трасс вычислений с помощью разработанной с участием автора утилиты «*Анализ трасс проигрывания TLC моделей, построенных на основе TLA+ спецификаций программных блоков БД и приводящих к нарушению инварианта безопасности информационных потоков*» значительно упрощает задачу выявления «ложных» срабатываний.

К перспективным направлениям дальнейших исследований можно отнести разработку исчерпывающих методических рекомендаций по применению процедур реклассификации (деклассификации и классификации) данных. К настоящему времени сформулированы общие принципы реклассификации, предложены отдельные схемы, которые, однако, требуют адаптации к описанной технологии. Кроме того, целесообразно провести отдельные исследования в отношении последнего этапа предложенной процедуры.

СПИСОК ЛИТЕРАТУРЫ / REFERENCES

1. Девянин П.Н., Тележников В.Ю., Хорошилов А.В. Формирование методологии разработки безопасного системного программного обеспечения на примере операционных систем. *Труды Института системного программирования РАН*. 2021;33(5):25–40. https://doi.org/10.15514/ISPRAS-2021-33(5)-2
[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. Тимаков А.А. Вариант реализации процедуры анализа информационных потоков в программных блоках PL/SQL с использованием платформы PLIF. *Программирование*. 2023;4:39–57.
[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

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

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