УДК 004.056; 519.686.2 doi: 10.15827/0236-235X.150.374-380 2025. Т. 38. № 2. С. 374–380

Динамическая верификация промышленных средств защиты информации на основе формальных моделей управления доступом

А.К. Петренко [1, 2, 3]✉, Д.В. Ефремов [1], Е.В. Корныхин [1, 2], В.В. Кулямин [1, 2, 3], В.А. Семенов [1]

[1] Институт системного программирования имени В.П. Иванникова РАН, г. Москва, 109004, Россия
[2] Московский государственный университет имени М.В. Ломоносова, г. Москва, 119991, Россия
[3] НИУ Высшая школа экономики, г. Москва, 101978, Россия

Ссылка для цитирования
Петренко А.К., Ефремов Д.В., Корныхин Е.В., Кулямин В.В., Семенов В.А. Динамическая верификация промышленных средств защиты информации на основе формальных моделей управления доступом // Программные продукты и системы. 2025. Т. 38. № 2. С. 374–380. doi: 10.15827/0236-235X.150.374-380
Информация о статье
Группа специальностей ВАК: 2.3.5, 2.3.6
Поступила в редакцию: 18.01.2025 После доработки: 10.02.2025 Принята к публикации: 21.02.2025

Аннотация. Работа посвящена методике верификации средств защиты информации (СЗИ) на соответствие формальным моделям безопасности. Поддержка такой верификации требуется современными стандартами разработки ПО с высоким уровнем доверия, а именно стандартами группы «Защита информации. Формальная модель управления доступом» и проектом стандарта «Защита информации. Системы с конструктивной информационной безопасностью. Методология разработки». В статье уточняется, что СЗИ – это механизм реализации требований информационной безопасности в программных системах. Обеспечение корректности и надежности его функционирования в рамках базовых программных систем промышленного уровня, таких как операционные системы или СУБД, является самостоятельной и достаточно сложной задачей. Отмечается, что при проектировании подобных систем основные свойства СЗИ должны быть заранее продуманы и точно описаны. Создавать и эффективно использовать строгие описания требований к СЗИ позволяют формальные методы разработки и верификации программ, но в классическом виде эти методы не удается масштабировать для таких крупных и сложных систем, как операционные системы или СУБД. Для верификации промышленной СЗИ предлагается использовать технику динамической верификации, в ходе которой сопоставляются трассы выполнения верифицируемой СЗИ и ее формальной модели. Поскольку модель и ее реализация описаны на разных уровнях абстракции, сопоставление опирается на неклассическое отношение уточнения. Данное уточнение реализовано в виде специального медиатора, строящего последовательность событий в модели по набору событий в самой верифицируемой системе. В статье приводится алгоритм проверки полученной трассы событий на корректность в рамках формальной модели СЗИ.
Ключевые слова: кибербезопасность, формальные спецификации, тестирование на основе моделей, защищенные операционные системы, конструктивная информационная безопасность
Благодарности. Авторы выражают благодарность Лаборатории Касперского за поддержку исследований в области конструктивной информационной безопасности и сотрудникам ИСП РАН за полезные замечания, внесенные в ходе выполнения исследования и написания статьи, в первую очередь, А.В. Хорошилову и А.А. Карнову

Введение. Повышение защищенности программных и программно-аппаратных систем становится одной из важнейших задач в современном мире. В системном и базовом ПО, таком как операционные системы и СУБД, в программной инфраструктуре поддержки сетевых коммуникаций выделяется механизм *средств защиты информации* (СЗИ). Верификация самого механизма СЗИ – первоочередная задача в обеспечении кибербезопасности. Для ее решения необходимо применять весь арсенал доступных средств, включая явное моделирование политик безопасности, в частности, управление доступом, и различные виды верификации не только самих формальных моделей, но и кода реализации СЗИ, который должен отвечать требованиям, зафиксированным в таких моделях. Спецификация СЗИ и верификация как полученной модели, так и реализации СЗИ являются важными фазами жизненного цикла разработки безопасного ПО.

В настоящее время завершается работа над проектом ГОСТ Р «Защита информации. Системы с конструктивной информационной безопасностью. Методология разработки». Одним из принципов, положенных в его основу, является обеспечение безопасности на начальных этапах разработки программ, чтобы во время формирования требований к системам, в том числе по кибербезопасности, и проектирования

не возникали потенциальные дефекты и уязвимости. Речь идет не об ошибках кодирования, а о дефектах, которые закладываются в систему на уровне замысла, при определении архитектуры системы, декомпозиции системы на компоненты и, соответственно, при проектировании интерфейсов между ними.

Одним из подходов, созвучных принципам конструктивной безопасности, является включение в жизненный цикл разработки программ создания функциональных требований в виде формальных моделей системы в целом или ее отдельных критически важных механизмов. Примером такого механизма являются СЗИ. В операционных системах и в СУБД основное назначение СЗИ – это управление доступом к таким объектам, как файлы, директории, таблицы, записи и др.

В 2021–2025 гг. были разработаны и введены в действие четыре части ГОСТ Р 59453 «Защита информации. Формальная модель управления доступом». Части 1–3 содержат рекомендации по разработке и верификации моделей управления доступом. В качестве удобного (но необязательного) языка моделирования рассматривается Event-B [1, 2]. Часть 4 стандарта содержит рекомендации для верификации реализации СЗИ на соответствие требованиям, заданным формальной спецификацией.

Несмотря на важность обеспечения информационной безопасности и появление стандартов, предписывающих их применение, до сих пор существуют лишь единичные примеры внедрения формальных методов разработки и верификации промышленных программных систем. Это объясняется как сложностью новых технологий, так и нехваткой удобных и эффективных инструментов, способствующих нахождению баланса между строгостью математического подхода и разумностью инженерных решений, позволяющих использовать эти технологии программистам-практикам. Основная тема представленного исследования – подход, который дает возможность применения формальных моделей и техник верификации на их основе в крупных и сложных системах ответственного назначения. Так, в отличие от задач разработки и верификации моделей управления доступом, поддерживаемых доступными и хорошо зарекомендовавшими себя инструментами, например Rodin [3], задача верификации промышленной СЗИ сложной системы (иногда размером в несколько миллионов строк кода) адекватной инструментальной поддержки пока не имеет. В связи с этим возникает потребность в инструментах, которые могли бы поддерживать процессы динамической верификации, в частности, описанные в части 4 ГОСТ Р 59453.

Известными подходами для динамического анализа на основе формальных моделей являются динамическая верификация (*Runtime Verification*, RV) [4] и тестирование на основе формальных моделей (*Model Base Testing*, MBT) [5, 6]. Оба термина используются при верификации систем, поведение которых специфицируется в форме некоторой формальной модели. В первом случае фокусом внимания обычно является проверка корректности (соответствия модели) трасс выполнения системы, полученных произвольным способом (из тестов, реальной работы или других источников). Во втором основной задачей является построение тестов для системы из формальной модели ее поведения.

Развитие формальных методов разработки программ, в частности, методов разработки формальных моделей и тестирования на основе этих моделей, концептуально близко к идеям конструктивной безопасности, так как модели СЗИ – это то, что должно появляться на фазе замысла программных систем ответственного назначения. Представленное исследование является одной из работ, которую ИСП РАН ведет совместно с Лабораторией Касперского по тематике конструктивной информационной безопасности.

Обзор существующих подходов

Примером использования подхода MBT является технология UniTESK [7], для поддержки которой были разработаны несколько инструментов для разных языков: Си, Java, C# и др. Однако механически перенести опыт UniTESK на задачу тестирования СЗИ нельзя, так как стартовой точкой процесса верификации СЗИ, который регламентируется в ГОСТ Р 59453, является модель управления доступом на достаточно абстрактном уровне, не связанном явно с имеющимися интерфейсами верифицируемой системы.

В моделях управления доступом описание механизмов контроля доступа может выполняться для набора абстрактных операций до-

ступа субъектов к объектам. Это позволяет сосредоточить внимание на основных особенностях используемых алгоритмов управления доступом, а не на их применении в конкретных компонентах системы. В рамках UniTESK предполагается моделирование поведения системы на основе программных контрактов ее реальных интерфейсных операций, которые могут быть достаточно далеки от абстрактных действий по получению доступа к тем или иным объектам. На основании этого легко сгенерировать тестовые оракулы (компоненты тестов, выносящие вердикт о согласовании поведения системы с моделью) для проверки работы реальных операций верифицируемой системы.

Выбор уровня абстракции спецификаций порождает проблему вынесения вердикта. По сути приходится сравнивать две трассы событий. Первая трасса порождается событиями, которые происходят в целевой СЗИ вследствие либо выполнения некоторого тестового сценария, либо работы СЗИ в режиме мониторинга, вторая – при проигрывании последовательности обращений к операциям, описанным в модели. Такая процедура проигрывания на модели иногда называется анимацией модели. Эти две трассы не совпадают друг с другом, хотя логически представляют поведение схожих программных систем – целевой СЗИ и ее модели. В работах по теоретической информатике для представления похожести поведения автоматов или систем размеченных переходов (LTS) часто используются понятия, связанные с трассовой эквивалентностью [8]. Но эти понятия плохо приспособлены для сравнения поведения программ, в которых фигурируют данные сложной структуры и к тому же описанных на разных уровнях абстракции. В данном случае сначала необходимо построить в модели трассу, соответствующую произошедшей в проверяемой системе последовательности событий, при этом само соответствие между событиями системы и операциями модели нетривиально (некоторым событиям в системе нет соответствия в модели, некоторым соответствует цепочка событий, иногда набору событий в системе соответствует одно в модели). Затем нужно проверить, допускает ли модель выполнение полученной трассы: если да, будем считать, что вердикт положительный, то есть поведение СЗИ отвечает требованиям ее спецификации, если нет, значит, поведение СЗИ отклонилось от требований модели.

Нетривиальность сопоставления трасс обусловлена тем, что корректная реализация СЗИ формально должна быть уточнением [9, 10] используемой модели, но отношение уточнения в данном случае является неклассическим и существенно более сложным, чем при последовательной разработке и верификации многоуровневых моделей в рамках классических формальных методов [11, 12]. Следует отметить, что свежих публикаций по практическому использованию неклассических методов уточнения крайне мало, что можно объяснить сложностью внедрения формальных методов в процессы промышленной разработки программ.

Уточнение в общем случае является соответствием между формальными моделями, поведение которых схоже с точностью до некоторых деталей. Различают доказательные и симуляционные виды уточнения. Доказательное уточнение [13] определяется так, чтобы при его наличии доказательство некоторых инвариантов уточняемой модели можно было перенести в уточненную без изменений (и, соответственно, эти инварианты выполнялись бы в уточненной модели автоматически). Доказательные уточнения, например [12], используются в формальных методах, делающих основной упор на строгое доказательство корректности моделей при их последовательном развитии. Симуляционное уточнение определяется таким образом, чтобы в некоторых контекстах можно было заменить уточняемую модель уточненной без изменения наблюдаемого в этом контексте поведения. Симуляционные уточнения используются в рамках методов, фокусирующихся на последовательном проектировании за счет насыщения моделей деталями, с итоговой проверкой исходных требований с помощью тестирования или динамической верификации. Они часто используются в формальных методах на основе языков VDM [14] и RSL [15]. Некоторые виды уточнений [11] являются одновременно доказательными и симуляционными.

Для целей данного исследования требуется более сложное уточнение симуляционного типа между насыщенными данными автоматными моделями, близкое к описанным в [10]. Достаточно подробно соответствующая ему схема повышения уровня абстракции в трассе, собранной во время тестирования или мониторинга СЗИ, описана в работе [16].

Схема анализа трасс в инструменте динамической верификации СЗИ

Система, подлежащая верификации, тестируется в некотором окружении, в которое включена программа-монитор. Монитор фиксирует последовательность операций СЗИ с их аргументами и результатами (в частности, была ли операция успешна, доступ разрешен, или неуспешна, доступ отклонен). Фиксируемая информация должна быть полностью воспроизводимой, то есть, кроме последовательности операций, содержать описание начального состояния тестируемой системы. Последовательность записей операций будем называть системной трассой.

Собранную системную трассу нужно привести к виду, позволяющему выполнить ее проверку на формальной модели Event-B. Для этого значения состояний тестируемой системы, параметры вызывавшихся операций и их результаты приводятся к виду, в котором эти сущности представлены в модели. Параметрами события (как минимум) должны быть аргументы системных операций и их результаты, включая существенные изменения в состоянии системы. Еще один результат системных операций – успешность операции, то есть по сути булевское значение, которое свидетельствует о том, что система разрешает или нет доступ к определенным объектам. Признаком успешности в формальной модели на Event-B будет истинность всех охранных условий события.

Чтобы узнать, правильно ли СЗИ приняла решение об успешности операции, нужно проверить охранное условие соответствующего события. Для этого необходимо построить параметры события по аргументам и результатам системной операции из трассы. Данные системы и модельные данные могут отличаться по структуре, что делает невозможным автоматическое преобразование. В этом случае требуется разработка компонента-медиатора, адаптирующего системную трассу к формальной модели (он реализует используемое отношение уточнения). Полученную модельную трассу проверяет компонент-оракул.

Процедуру проверки трассы на соответствие формальной модели СЗИ описывает алгоритм CheckTrace:

```
procedure CheckTrace(initial, operations)
machine ← CreateMachine()
  for all event ∈ TranslateInitial(initial)
    value ← ComputeEvent(event, machine)
    assert IsTrue(value)
    machine ← ExecuteActions(value, machine)
  end for
  for all op ∈ operations
    if not IsSuccess(op)and NotEnoughResource(op)
      continue
    end if
    event ← TranslateOperation(op)
    value ← ComputeEvent(event, machine)
    if not (IsTrue(value) ⇔ IsSuccess(op))
      output обнаружено расхождение трассы с моделью
      break
    end if
    UpdateCoverage(value)
    if IsTrue(value)
    machine ← ExecuteActions(value, machine)
    end if
  end for
end procedure
```

Параметрами процедуры CheckTrace являются две части системной трассы – начальное состояние (initial) и последовательность операций (operations).

Функция CreateMachine создает объект-машину. Выполняет ее компонент-оракул.

Функция TranslateInitial возвращает последовательность событий модели с параметрами, которая перемещает объект-машину в состояние, соответствующее начальному состоянию трассы. Функцию выполняет компонент-медиатор.

Функции ComputeEvent, ExecuteActions и IsTrue, а также процедура UpdateCoverage выполняются компонентом-оракулом. Функция ComputeEvent вычисляет значения всех охранных условий событий в текущем состоянии объекта-машины; IsTrue проверяет истинность всех охранных условий; процедура UpdateCoverage обновляет покрытие по элементам формальной модели. Функция ExecuteActions вычисляет состояние объекта-машины после выполнения действий события.

Функции IsSuccess и NotEnoughResources выполняются адаптером к монитору, IsSuccess проверяет успешность завершения системной операции, а NotEnoughResources – завершение системной операции с ошибкой нехватки ресурсов. Такие операции нужно пропустить при проверке.

Описание алгоритма показывает разграничение ролей компонентов системы динамической верификации.

Заключение

В области кибербезопасности наблюдается активное развитие технологий и нормативно-правовой базы, регламентирующей разработку и сертификацию программных систем ответственного назначения. Иногда разработка стандартов опережает технологии, что свидетельствует о высоких требованиях к информационной безопасности в ИТ-отрасли. Такая ситуация сейчас сложилась в направлениях конструктивной информационной безопасности, верификации моделей управления доступом и самих средств управления доступом на соответствие формальным моделям. Данная работа представляет расширенный алгоритм схемы анализа трасс работы промышленных СЗИ на соответствие формальным моделям управления доступом.

Предложенный подход уже реализован в экспериментальном образце программных средств, опробованном в промышленных проектах. В частности, он используется в жизненном цикле разработки специальных версий операционной системы AstraLinux. Инструментальные средства, создающиеся для поддержки новых стандартов в области информационной безопасности, входят в арсенал средств поддержки разработки безопасного ПО.

Список литературы

1. Девянин П.Н., Ефремов Д.В., Кулямин В.В., Петренко А.К., Хорошилов А.В., Щепетков И.В. Моделирование и верификация политик безопасности управления доступом в операционных системах. М.: Горячая линия–Телеком, 2019. 214 с.

2. Abrial J.-R. Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge, NY, 2010, 586 p.

3. Abrial J.-R., Butler M., Hallerstede S., Hoang T.S., Mehta F., Voisin L. Rodin: An open toolset for modelling and reasoning in Event-B. Int. J. on Software Tools for Technology Transfer, 2010, vol. 12, no. 6, pp. 447–466. doi: 10.1007/s10009-010-0145-y.

4. Bartocci E., Falcone Y. Lectures on runtime verification. Introductory and advanced topics. In: LNCS, 2018, vol. 10457, 233 p.

5. Broy M., Jonsson B., Katoen J.-P., Leucker M., Pretschner A. Model-based testing of reactive systems. Advanced lectures. In: LNCS, 2005, vol. 3472, 659 p.

6. Utting M., Legeard B. Practical Model-Based Testing: A Tools Approach. Morgan-Kaufmann Publ., 2007, 456 p. doi: 10.1016/B978-0-12-372501-1.X5000-5.

7. Bourdonov I., Kossatchev A., Kuliamin V., Petrenko A. UniTesK test suite architecture. In: LNCS. Proc. FME, 2002, vol. 2391, pp. 77–88. doi: 10.1007/3-540-45614-7_5.

8. van Glabbeek R.J. The linear time – brachning time spectrum. In: LNCS. Proc. CONCUR, 1990, vol. 458, pp. 278–297. doi: 10.1007/BFb0039066.

9. Schellhorn G. Verification of ASM refinements using generalised forward simulation. J.UCS, 2001, vol. 7, no. 11, pp. 952–979.

10. Börger E. ASM refinement method. Formal Aspects of Computing, 2003, vol. 15, no. 2-3, pp. 237–257. doi: 10.1007/s00165-003-0012-7.

11. Bolton C., Davies J. Refinement in object-Z and CSP. In: LNCS. Proc. IFM, 2002, vol. 2335, pp. 225–244. doi: 10.1007/3-540-47884-1_13.

12. Abrial J.-R., Hallerstede S. Refinement, decomposition, and instantiation of discrete models: Application to event-B. Fundamenta Informaticae, 2007, vol. 77, no. 1-2, pp. 1–28.

13. de Roever W.P., Engelhardt K. Data Refinement: Model-Oriented Proof Methods and their Comparison. Cambridge University Press, Cambridge, 1999, 436 p.

14. Fitzgerald J.S., Larsen P.G., Mukherjee P., Plat N., Verhoef M. Validated Designs for Object-oriented Systems. Springer-Verlag Publ., London, 2005, 404 p. doi: 10.1007/b138800.

15. George C. The RAISE specification language. A tutorial. In: LNCS. Proc. VDM, 1991, vol. 552, pp. 238–319. doi: 10.1007/BFb0019998.

16. Efremov D., Shchepetkov I. Runtime verification of linux kernel security module. In: LNPSE. Proc. FM Int. Workshops, 2019, vol. 12233, pp. 185–199. doi: 10.1007/978-3-030-54997-8_12.

Software & Systems doi: 10.15827/0236-235X.150.374-380 2025, 38(2), pp. 374–380

Runtime verification of industrial information security tools based on formal access control models

Aleksandr K. Petrenko [1, 2, 3]✉, Denis V. Efremov [1], Eugeny V. Kornykhin [1, 2], Victor V. Kulyamin [1, 2, 3], Vitaly A. Semenov [1]

[1] Ivannikov Institute for System Programming of the RAS, Moscow, 109004, Russian Federation
[2] Lomonosov Moscow State University, Moscow, 119991, Russian Federation
[3] National Research University, Higher School of Economics, Moscow, 101978, Russian Federation

For citation
Petrenko, A.K., Efremov, D.V., Kornykhin, E.V., Kulyamin, V.V., Semenov, V.A. (2025) 'Runtime verification of industrial information security tools based on formal access control models', *Software & Systems*, 38(2), pp. 374–380 (in Russ.). doi: 10.15827/0236-235X.150.374-380
Article info
Received: 18.01.2025 After revision: 10.02.2025 Accepted: 21.02.2025

Abstract. The work focuses on the methodology of verifying information security tools for compliance with formal security models. Such verification is due to modern standards of software development with a high level of trust, namely the standards of the group GOST R 59453 "Information security. Formal access control model" and the draft GOST R "Information security. Constructive information security systems. Development methodology". Information security tools is a mechanism for implementing information security requirements in software systems. Ensuring the mechanism correctness and reliability in terms of basic industrial-grade software systems, such as operating systems or database management systems (DBMS), is an independent and rather complex task. Designing such systems implies that the main properties of the information security tools are premeditate and accurately described. Formal methods of software development and verification allow creating and effective using strict descriptions of requirements for information security systems. However, in their classical form, formal methods are non-scalable for such large and complex systems as operating systems or DBMS. To verify industrial information security tools, the authors propose using runtime verification, which compares execution traces of a verified information security system and its formal model. Since the model and its implementation are described at different abstraction levels, the comparison is based on a non-classical refinement relation. The relation is implemented as a special mediator that builds a sequence of events in the model based on a set of events in the verified system itself. The paper describes an algorithm for checking the obtained event trace for correctness within the framework of the formal model of an information security tool.

Keywords: cybersecurity, formal specifications, model-based testing, secure operating systems, Secure-by-Design
Acknowledgements. he authors wish to express their thanks to Kaspersky Lab for supporting research in the field of constructive information security and to the staff of the Institute for System Programming of the Russian Academy of Sciences for useful comments made during the research and writing the paper, primarily to A.V. Khoroshilov and A.A. Karnov

References

1. Devyanin, P.N., Efremov, D.V., Kulyamin, V.V., Petrenko, A.K., Khoroshilov, A.V., Shchepetkov, I.V. (2019) *Modeling and Verification of Access Control Security Policies in Operating Systems*. Moscow, 214 p. (in Russ.).
2. Abrial, J.-R. (2010) *Modeling in Event-B: System and Software Engineering*. Cambridge, NY: Cambridge University Press, 586 p.
3. Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L. (2010) 'Rodin: An open toolset for modelling and reasoning in Event-B', *Int. J. on Software Tools for Technology Transfer*, 12(6), pp. 447–466. doi: 10.1007/s10009-010-0145-y.
4. Bartocci, E., Falcone, Y. (2018) 'Lectures on runtime verification. Introductory and advanced topics', in *LNCS*, 10457, 233 p.
5. Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (2005) 'Model-based testing of reactive systems. Advanced lectures', in *LNCS*, 3472, 659 p.
6. Utting, M., Legeard, B. (2007) *Practical Model-Based Testing: A Tools Approach*. Morgan-Kaufmann Publ., 456 p. doi: 10.1016/B978-0-12-372501-1.X5000-5.
7. Bourdonov, I., Kossatchev, A., Kuliamin, V., Petrenko, A. (2002) 'UniTesK test suite architecture', in *LNCS. Proc. FME*, 2391, pp. 77–88. doi: 10.1007/3-540-45614-7_5.
8. van Glabbeek, R.J. (1990) 'The linear time – brachning time spectrum', in *LNCS. Proc. CONCUR*, 458, pp. 278–297. doi: 10.1007/BFb0039066.
9. Schellhorn, G. (2001) 'Verification of ASM refinements using generalised forward simulation', *J.UCS*, 7(11), pp. 952–979.
10. Börger, E. (2003) 'ASM refinement method', *Formal Aspects of Computing*, 15(2-3), pp. 237–257. doi: 10.1007/s00165-003-0012-7.

11. Bolton, C., Davies, J. (2002) 'Refinement in object-Z and CSP', in *LNCS. Proc. IFM*, 2335, pp. 225–244. doi: 10.1007/3-540-47884-1_13.

12. Abrial, J.-R., Hallerstede, S. (2007) 'Refinement, decomposition, and instantiation of discrete models: Application to event-B', *Fundamenta Informaticae*, 77(1-2), pp. 1–28.

13. de Roever, W.P., Engelhardt, K. (1999) *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge: Cambridge University Press, 436 p.

14. Fitzgerald, J.S., Larsen, P.G., Mukherjee, P., Plat, N., Verhoef, M. (2005) *Validated Designs for Object-oriented Systems*. London: Springer-Verlag Publ., 404 p. doi: 10.1007/b138800.

15. George, C. (1991) 'The RAISE specification language. A tutorial', in *LNCS. Proc. VDM*, 552, pp. 238–319. doi: 10.1007/BFb0019998.

16. Efremov, D., Shchepetkov, I. (2019) 'Runtime verification of linux kernel security module', in *LNPSE. Proc. FM Int. Workshops*, 12233, pp. 185–199. doi: 10.1007/978-3-030-54997-8_12.

Авторы

Петренко Александр Константинович [1, 2, 3],
д.ф.-м.н., профессор, зав. отделом,
petrenko@ispras.ru
Ефремов Денис Валентинович [1],
старший научный сотрудник, efremov@ispras.ru
Корныхин Евгений Валерьевич [1, 2],
к.ф.-м.н., доцент,
старший научный сотрудник,
kornevgen@ispras.ru
Кулямин Виктор Вячеславович [1, 2, 3], к.ф.-м.н.,
доцент, ведущий научный сотрудник,
kuliamin@ispras.ru
Виталий Адольфович Семенов [1],
д.ф.-м.н., профессор, зав. отделом,
vital@ispras.ru

[1] Институт системного программирования
имени В.П. Иванникова РАН,
г. Москва, 109004, Россия
[2] Московский государственный университет
имени М.В. Ломоносова,
г. Москва, 119991, Россия
[3] НИУ Высшая школа экономики,
г. Москва,
101978, Россия

Authors

Aleksandr K. Petrenko [1, 2, 3],
Dr. Sci. (Physics and Mathematics),
Professor, Head of Department, petrenko@ispras.ru
Denis V. Efremov [1],
Senior Researcher, efremov@ispras.ru
Eugeny V. Kornykhin [1, 2],
Cand. of Sci. (Physics and Mathematics),
Associate Professor, Senior Researcher,
kornevgen@ispras.ru
Victor V. Kulyamin [1, 2, 3], Cand. of Sci. (Physics and
Mathematics), Leading Researcher,
kuliamin@ispras.ru
Vitaly A. Semenov [1],
Dr. Sci. (Physics and Mathematics),
Professor, Head of Department, vital@ispras.ru

[1] Ivannikov Institute for System Programming RAS,
Moscow, 109004,
Russian Federation
[2] Lomonosov Moscow State University,
Moscow, 119991,
Russian Federation
[3] National Research University,
Higher School of Economics,
Moscow, 101978, Russian Federation