DOI: 10.15514/ISPRAS-2024-36(5)-1

Конструирование программных систем, нацеленное на обеспечение безопасности

[1,2,3] *В.В. Кулямин, ORCID: 0000-0003-3439-9534 <kuliamin@ispras.ru>*
[1,2,3] *А.К. Петренко, ORCID: 0000-0001-7411-3831 <petrenko@ispras.ru>*
[4] *Е.А. Рудина, ORCID: 0000-0003-2944-162X <Ekaterina.Rudina@kaspersky.com>*
[1] *Институт системного программирования им. В.П. Иванникова РАН, Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.*
[2] *Московский государственный университет имени М.В. Ломоносова, Россия, 119991, г. Москва, Ленинские горы, д. 1.*
[3] *Национальный исследовательский университет «Высшая школа экономики», Россия, 101000, г. Москва, ул. Мясницкая, д. 20.*
[4] *АО «Лаборатория Касперского», Россия, 125212, г. Москва, Ленинградское шоссе, д. 39А, стр. 2.*

Аннотация. Конструктивная информационная безопасность — один из подходов, нацеленных на обеспечение надежности и безопасности программных систем, занимающий среди них достаточно важное место. Он развивается уже более 50 лет, однако широкие массы разработчиков не знакомы с его принципами и методами. Важной задачей в рамках популяризации этого подхода является создание типологии задач и техник обеспечения конструктивной информационной безопасности и определение приоритетных направлений их развития.

Ключевые слова: информационная безопасность; методы предотвращения атак; конструирование безопасных систем; архитектурные образцы и стили; усиление защищенности кода; моделирование архитектуры; анализ архитектуры; безопасные языки программирования.

Для цитирования: Кулямин В.В., Петренко А.К., Рудина Е.А. Конструирование программных систем, нацеленное на обеспечение безопасности. Труды ИСП РАН, том 36, вып. 5, 2024 г., стр. 7–16. DOI: 10.15514/ISPRAS–2024–36(5)–1.

Благодарности: В статье использовались материалы, которые были собраны сотрудниками Лаборатории Касперского и ИСП РАН. В первую очередь мы выражаем благодарность А. Бухвалову как инициатору данной работы, а также Д. Буздалову, С. Ерошкину, Н. Горелиц, А. Максимову, М. Кричанову, Е. Корныхину, В. Падаряну, А. Угненко, А. Хорошилову.

Software Security by Design

[1,2,3] *V.V. Kuliamin, ORCID: 0000-0003-3439-9534 <kuliamin@ispras.ru>*
[1,2,3] *A.K. Petrenko, ORCID: 0000-0001-7411-3831 <petrenko@ispras.ru>*
[4] *E.A. Rudina, ORCID: 0000-0003-2944-162X <Ekaterina.Rudina@kaspersky.com>*
[1] *V.P. Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*
[2] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia.*
[3] *National Research University «Higher School of Economy», 20, Myasnitskaya st., Moscow, 101000, Russia.*
[4] *AO Kaspersky Lab, 39A, str. 2, Leningradskoye sh., Moscow, 125212, Russia.*

Abstract. Security-by-Design is an important approach to ensure software security and reliability. It has been developing already for more than 50 years, but its principles and techniques are still not well known among wide society of software developers. To make the approach more familiar and popular we need to reestablish its goals and problems, to classify and explain its techniques, and formulate trends of its future development. This paper reformulates the main principles of Security-by-Design, provides some examples of security design patterns and anti-patterns, and also explores relations between the approach and software architecture analysis methods, hardening techniques, and safe programming languages.

Keywords: software security; attack prevention methods; secure software design; software architecture styles and patterns; hardening; software architecture modeling; software architecture analysis; safe programming languages.

For citation: Kuliamin V.V., Petrenko A.K., Rudina E.A. Software Security by Design. Trudy ISP RAN/Proc. ISP RAS, vol. 36, issue 5, 2024, pp. 7-16 (in Russian). DOI: 10.15514/ISPRAS-2024-36(5)-1.

Acknowledgements. The paper used materials provided by many employees of Kaspersky Lab and Institute for System Programming. We thank A. Buhvalov for the initiative to prepare this paper, we also thank D. Buzdalov, S. Eroshkin, H. Gorelits, A. Maksimov, M. Kritchanov, E. Kornykhin, V. Padaryan, A. Ugnenko, and A. Khoroshilov.

1. Введение

В последние десятилетия значимость и сложность проблем обеспечения безопасности программно-аппаратных систем значительно повысилась в связи с действием следующих факторов.

- Рост количества и сложности решаемых такими системами задач из-за увеличивающихся требований со стороны бизнеса и органов государственного управления, а также из-за высокой скорости изменений самих требований.
- Создание с использованием разнородных технологий все более сложных систем из гетерогенных компонентов с целью охватить как можно больше пользователей, относящихся к различным социальным слоям и распределенных географически.
- Обострение конкурентной борьбы между бизнес-группами и между государствами и совершенствование технологий атак, препятствующих функционированию программных систем и нарушающих их информационную безопасность.

Проблема обеспечения информационной безопасности является комплексной и многоаспектной, у нее нет простых и линейных решений. Систематические и тщательно продуманные подходы к выработке возможных решений предполагают использование разнообразных методов предотвращения, обнаружения и устранения проблем безопасности на всех стадиях создания, сопровождения и эксплуатации программно-аппаратных систем, причем ни один элемент этой триады не может быть проигнорирован без значительного

повышения рисков. В частности, без систематической работы над предотвращением проблем безопасности трудоемкость поддержки систем становится чрезмерно дорогой из-за постоянной потребности в закрытии все новых и новых обнаруживаемых уязвимостей.

Одним из наиболее значимых подходов к предотвращению проблем информационной безопасности является конструирование систем, с самого начала нацеленное на ее обеспечение [1,2]. В России этот подход получил название Конструктивная информационная безопасность (КИБ). Конечно же, предотвратить все возможные проблемы безопасности никогда не удается, но можно существенно уменьшить разнообразие потенциальных атак и риски их успешности, тем самым значительно снижая стоимость поддержания системы в высоко защищенном состоянии и повышая уровень доверия к ней. Для этого необходимо с самого начала проектирования не оставлять проблемы защиты системы на более поздние этапы, а одновременно решать задачи реализации основных функций системы и защиты этой системы как от внешних, так и от внутренних факторов, ведущих к нештатному функционированию или отказам.

Подход к конструированию, нацеленному на обеспечение безопасности, пока остается достаточно аморфным, разные авторы и группы выделяют в нем немного отличающиеся списки элементов. Но, в целом, все согласны, что в него входит систематическое использование нескольких принципов и правил, а также ряда учитывающих эти правила образцов (паттернов) проектирования и низкоуровневых паттернов организации данных и взаимодействия.

Конструктивная информационная безопасность не противопоставляется другим подходам и технологиям, нацеленным на повышение надежности и защищенности программ. Более того, данный подход предполагает, что жизненный цикл программ обеспечен, например, такими технологиями как SSDL (Secure Software Development Lifecycle) или РБПО (Разработка Безопасного Программного Обеспечения). То есть перечисленные подходы и технологии дополняют друг друга, и их следует применять совместно.

Принципы не всегда легко трансформируются в понятные разработчикам технологии, инструкции, техники разработки безопасного ПО, а в рамках конструктивного подхода они явно нужны. Далее мы сначала остановимся на принципах, а потом на тех конструктивных составляющих, которые должны развить и обогатить арсенал средств конструктивной информационной безопасности.

Важным вопросом в рамках популяризации конструктивной информационной безопасности является создание типологии задач и техник, присущих этому подходу, и определение приоритетных направлений их развития. Исследованию данного вопроса посвящен совместный проект Лаборатории Касперского и Института системного программирования. Данная статья является публикацией результатов исследований первой фазы этого проекта.

2. Принципы построения безопасных программных систем

Отдельные идеи, связанные с конструированием систем, нацеленным на обеспечение безопасности, (Secure by Design, Secure by Construction) [1] возникают достаточно давно, первые из них развиваются в рамках методов создания систем, корректных по построению (Correctness by Construction) [3-5]. Такие методы развиваются уже более 50-ти лет, но на практике применяются лишь к системам небольшого размера или в сочетании с другими более масштабируемыми технологиями. Тем не менее уже первые работы в этом направлении указали на важность успешного предотвращения многих проблем на этапе проектирования системы.

Можно утверждать, что основе методов и технологий конструирования программных систем лежат некоторые фундаментальные принципы, в том числе принципы безопасности. Поскольку само понятие безопасности зависит от назначения и условий применения системы, то принципы могут иметь разные приоритеты и применяться с различной степенью

строгости. Важно, что принципы применяются практически на всех этапах жизненного цикла систем, при этом они реализуются разными методами и средствами. Приведем несколько примеров принципов конструктивной информационной безопасности [2,6]:

- Принцип изоляции — нужно стараться свести данные, доступные одновременно нескольким компонентам или передаваемые между ними, к минимуму, необходимому для обеспечения решения ими своих задач.
- Принцип минимизации поверхности атаки — нужно стараться свести к минимуму количество компонентов, напрямую взаимодействующих с внешними системами или пользователями и получающих извне (возможно, через посредников) необработанные или непроверенные данные.
- Принцип минимизации привилегий — нужно стараться выдавать каждому компоненту или процессу минимальные привилегии, необходимые ему для решения его задач, и избегать неоправданного расширения привилегий.
- Принцип эшелонированной защиты — нужно стараться обеспечить определенный уровень защиты при каждом переходе управления и/или данных между слоями/уровнями функциональности системы, не доверяя полностью защите каждого отдельного компонента или сервиса и защищаясь от возможного внедрения злоумышленника в любой из них.
  При этом понятно, что каждый слой защиты приводит к дополнительному снижению как производительности, так и удобства использования и сопровождения компонентов, поэтому организовывать защиту стоит не между каждыми двумя взаимодействующими компонентами, а между естественно возникающими в структуре системы их слоями.
- Принцип обеспечения безопасности по умолчанию — установленные по умолчанию настройки, значения данных и алгоритмы работы должны обеспечивать безопасное функционирование системы, даже если пользователи или администраторы никак не будут вмешиваться в ее работу.
- Принцип обеспечения безопасности сбоев — нужно стараться предусмотреть возможные сбои и ошибки в различных компонентах системы (включая возможные успешные атаки на них) и сделать возможным безопасное продолжение, или, при невозможности такового, безопасное завершение, работы системы при их возникновении, включая реакцию на такие события и аккуратное протоколирование происходящего для последующего анализа.
- Принцип прозрачности решений по защите — нужно стараться аккуратно определять возможные атаки и в полном и однозначном виде описывать принимаемые проектные решения для противодействия им, избегая неопределенных возможных угроз (от которых невозможно защититься достаточно надежно) и невнятных решений по защите (которая в этом случае не будет обладать значимым уровнем доверия).

3. Паттерны и антипаттерны безопасной архитектуры

Принципы не определяют проектные решения, но задают набор правил, которым стоит следовать при их выработке, а также позволяют оценить принимаемые решения как следующие им в достаточной мере, или, наоборот, имеющие потенциально небезопасные последствия.

Образцы проектирования, используемые при конструировании, нацеленном на обеспечение безопасности, задают более конкретные элементы решений по защите системы, хотя их доработка, конфигурирование и расширение остаются возможными [7-9].

Примером может служить образец монитор защиты (reference monitor) [10]. Монитор защиты определяет, как можно реализовать унифицированное применение единой политики безопасности во всех компонентах системы, при этом внесение изменений в политику может выполняться в одном месте и сразу становится актуальным для всей системы. Другим примером архитектурного образца, нацеленного на повышение безопасности, является виртуализация [11]. Она позволяет аккуратно реализовать принцип изоляции сразу для большого числа компонентов.

Важным подходом к обеспечению защиты процессов от нежелательного взаимного влияния является концепция ядра разделения [12]. Роль ядра безопасности заключается в воссоздании в рамках одной общей машины такого окружения, которое поддерживает различные компоненты системы и обеспечивает каналы связи между ними таким образом, что отдельные компоненты системы не могут отличить эту общую среду от физически распределенной.

Ядро с гарантиями изоляции [13] является своего рода ядром разделения для интегрированной модульной авионики и даёт гарантии изоляции в основном согласно схеме разделения ARINC 653 [14].

Ядра разделения в операционных системах предоставляют функции для обеспечения изоляции процессов и управления информационными потоками. Они должны быть достаточно компактными, чтобы обеспечить возможность формальной верификации корректности их работы. Функциональность может незначительно отличаться в зависимости от области применения системы, созданной с использованием такого ядра.

Ядро разделения не обязательно совпадает с ядром операционной системы.

Некоторые из низкоуровневых функций, реализованных ядром операционной системы, также могут поддерживать изоляцию процессов (разделение доменов), но, как правило, оно также предоставляет множество функций, отсутствующих в ядре разделения. С другой стороны, одной из задач ядра разделения может быть обеспечение выполнения политик управления информационными потоками. В настоящее время ядро разделения – это не “ядро” в общепринятом смысле, а комбинация аппаратных технологий и программных компонентов, которые гарантированно обеспечивают изоляцию процессов [15].

Ядра разделения обеспечивают пространственную и временную изоляцию процессов, но позволяют им взаимодействовать контролируемым образом. Степень изоляции, обеспечиваемая каждым из этих типов разделения, зависит от назначения системы и конкретного архитектурного решения.

4. Усиление защиты кода (hardening)

Методы, нацеленные на повышение безопасности за счет раннего обнаружения возможных ошибок в архитектуре, коде ПО, а также препятствующие техникам выполнения известных атак, принято объединять под термином hardening [16]. Одним из примеров такой техники является окаймление выделяемых участков памяти специальным образом заполненными массивами, что позволяет быстро выявить возможные выходы за границы буферов. В действительности hardening объединяет большое количество разнообразных приемов, нацеленных на предупреждение вероятных угроз. Набор таких угроз и методов их парирования постоянно обновляется, поэтому для практического использования нужны удобные технические средства для проведения защитных мероприятий в ходе разработки и эксплуатации систем на регулярной основе.

5. Инструментальная поддержка архитектурных моделей и каталоги уязвимостей

С ростом размера и сложности программной или программно-аппаратной системы возрастает потребность в инструментах работы с архитектурными моделями. Наиболее известным инструментом архитектурного моделирования, по-видимому, можно назвать SysML, который можно рассматривать как надъязык языка UML. Другим известным языком в этом классе является AADL (Architecture Analysis & Design Language) [17]. AADL в отличие от SysML имеет стандартизованное текстовое представление и строго описанную семантику, что делает его более удобным для разработчиков инструментов анализа архитектурных моделей.

Инструменты моделирования программных систем распространены не очень широко, еще меньше круг пользователей средств архитектурного моделирования. В связи с этим набор отработанных сценариев использования архитектурного моделирования и, в частности, моделирования, направленного на повышение информационной безопасности, невелик. Примерами видов архитектурного анализа и способов использования архитектурных моделей интересных с позиций конструктивной безопасности являются: поиск уязвимостей архитектурных решений, анализ отказоустойчивости и самовосстановления, автоматизированный анализ поверхности атаки, моделирование поведения системы в агрессивном окружении, моделирование управления конфигурациями, тестирование конфигураций и др.

Надо отметить, что большая часть инструментов архитектурного моделирования поддерживает лишь небольшое число сценариев использования, и нет ни одного, который поддерживал хотя бы перечисленный набор. Одной из причин, почему пока еще нет удобных и эффективных инструментов для конструирования архитектур безопасных программных систем, является отсутствие добротного каталога шаблонов безопасных архитектурных решений и уязвимостей архитектуры. Лучшим на сегодняшний день собранием архитектурных уязвимостей является каталог CAWE (Common Architectural Weakness Enumeration) [18]. В этом каталоге потенциальные уязвимости снабжены указанием архитектурной тактики и методов возможного устранения уязвимостей. Авторы разработали интерактивное веб-приложение для архитекторов и разработчиков для доступа к каталогу. Каталог CAWE содержит 224 уязвимости, связанные с безопасностью. Пока этот каталог далек от завершенности и логической стройности и, к сожалению, как минимум с 2020 года каталог CAWE не обновляется.

В качестве ориентира для развития CAWE можно использовать общую копилку уязвимостей в программном и аппаратном обеспечении CWE (Common Weakness Enumeration) [19] и список актуальных уязвимостей, который ведется в базе CVE [20].

В рамках Microsoft Security Development Lifecycle [21] составлен свой каталог потенциальных уязвимостей. Более того, в Microsoft разработали среду моделирования Microsoft Threat Modeling Tool [22] для описания дизайна и автоматической проверки его на угрозы из этого каталога. Следует обратить внимание работы, посвященные классификации потенциальных уязвимостей и техникам их формализованного описания для дальнейшего использования при анализе проектов [23-26].

6. Безопасные языки программирования

Еще одним инструментом конструктивного подхода к обеспечению безопасности программных систем являются так называемые языки безопасного программирования. Частным примером таких языков являются memory safe языки. В действительности в последнее время появилось много языков, обеспечивающих защиту от разнообразных угроз информационной безопасности. Такими угрозами могут быть гонки по данным (data race), некорректная работа со сложными, часто динамическими, типами данных. Сейчас вышли на

уровень промышленной зрелости некоторые функциональные языки, например, Haskell. Имеется совсем новое направление функционального программирования языки с так называемыми зависимыми данными (dependent types). Интересным представителем этого направления является Idris [27].

При промышленном применении, помимо прямого эффекта от использования безопасных языков, который состоит в предотвращении уязвимостей определенного рода, будет и дополнительный эффект, обусловленный отсутствием необходимости в соответствующих возможностях в инструментах статического и динамического анализа, а это в свою очередь, позволит выделять больше ресурсов на поиск дефектов и уязвимостей других видов.

7. Заключение

В данной статье мы концентрируемся на тех проблемах, которые в большей степени связаны с просчетами в архитектурных решениях, нежели с дефектами и уязвимостями, которые по многим другим причинам возникают в программном коде. Чего же не хватает для того, чтобы как позитивный, так и негативный опыт проектирования накапливался и эффективно использовался?

Во-первых, нужно отдавать себе отчет, что исследования в области конструктивной информационной безопасности и внедрение результатов этих исследований в практику требуют тщательно продуманных действий одновременно в разных направлениях. Нужны будут совместные усилия по исследованиям и разработкам, по созданию и распространению учебных курсов и учебно-методических материалов, созданию новых и корректировке имеющихся нормативно-правовых документов, проведению пилотных проектов внедрения технологий и процессов КИБ. Во-вторых, видно, что в настоящее время наименее исследованы методы и технологии поддержки ранних фаз жизненного цикла разработки ПО, в частности, методы анализа свойств и уязвимостей архитектурных решений еще на фазе проектирования, а также методы выявления связей проектных решений с проблемами информационной безопасности, которые возникают в процессе разработки и эксплуатации ПО.

Отставание в этой сфере (оно наблюдается как в нашей стране, так и за рубежом) объясняется не только отсутствием удобных и эффективных средств архитектурного моделирования, но и тем, что в практике на фазах разработки и особенно сопровождения ПО про архитектуру часто забывают – она была важна на фазе замысла, а когда код разработан, она, как бы, уже не нужна. Но это не так. В действительности, после того как ошибка проявилась, найдено место в программе, куда вносится исправление, «работа над ошибками» еще не доведена до конца. Полезно взглянуть на архитектуру системы еще раз и проанализировать возможные недоработки в структуре системы и интерфейсах ее компонентов. То есть, в ходе сопровождения и развития программной системы к анализу и, возможно, коррекции проекта, и, соответственно, к накоплению знаний о шаблонах «правильных» и «неправильных» архитектурных решений нужно возвращаться регулярно.

Нередко ошибки в программах обусловлены тем, что интерфейсы подталкивают разработчиков к небезопасному программированию. Например, если в интерфейсах не предусмотрена возможность проверки работоспособности того или иного компонента-сервера, многие компоненты-клиенты будут продолжать обращаться к нему в расчете, что соответствующие операции выполняются. В некоторых ситуациях это приводит к катастрофическим результатам, и основная ответственность здесь не на программистах-разработчиках, а на проектировщиках.

Заметим, что действенным инструментом повышения защищенности программ является организация баз данных уязвимостей. Такие коллекции служат нескольким целями. Во-первых, это форма накопления опыта для обучения и использования многочисленным сообществом программистов; во-вторых, это систематизированная база для создания check-

list'ов, используемых, например, при инспекциях кода; в-третьих, это основа для разработки различных инструментов анализа кода.

В широкой практике программной инженерии, к сожалению, нет аналогичных репозиториев для архитектурных ошибок, а они, очевидно, появятся, если работу над анализом и исправлением найденных ошибок в коде доводить до конца, то есть выявлять те архитектурные уязвимости, которые привели к ошибкам в программном коде или, по крайней мере, повысили риски, связанные с определенными программными решениями. В настоящее время инструментов, поддерживающих такой сценарий работы нам не известно. Самые близкие к этой теме работы опубликованы в [28-29]. С другой стороны, когда появятся такие репозитории, возникнет заинтересованность в программных инструментах для работы с архитектурными моделями и поиска архитектурных уязвимостей на всех фазах жизненного цикла. Это одно из важных направлений развития технологий конструктивной информационной безопасности.

Конструирование систем, нацеленное на обеспечение безопасности — один из множества подходов к обеспечению безопасности, занимающий в этом наборе достаточно важное место. Он развивается уже более 50 лет, однако широкие массы разработчиков не очень хорошо знакомы с его элементами, разве что отдельные техники и образцы используются достаточно активно. Такое положение дел не только не способствует эффективному развитию области, связанной с обеспечением информационной безопасности, но и приводит к тому, что многократно приходится возвращаться к преодолению проблем, которые, казалось бы, уже достаточно хорошо изучены. Исследование способов предотвращения проблем информационной безопасности и разработка методов, инструментов и технологий конструктивной информационной безопасности, это еще один путь для повышения защищенность программных и программно-аппаратных систем.

Список литературы / References

[1]. Cavoukin, A. and Dixon, M. Privacy and security by design: An enterprise architecture approach. Information and Privacy Commissioner, Canada, Tech. Rep., 9, 2013.
[2]. Johnsson, D.B., Deogun, D., and Sawano, D. Secure By Design. Manning Publications, 2019. ISBN: 9781617294358.
[3]. Dijkstra, E.W.A Discipline of Programming. Prentice Hall, Upper Saddle River, 1976. ISBN: 9780132158718/
[4]. Gries, D. The Science of Programming. Springer, Heidelberg, 1987. DOI: 10.1007/978-1-4612-5983-1.
[5]. Kourie, D.G., Watson, B.W. The Correctness-by-Construction Approach to Programming. Springer, Heidelberg, 2012. DOI: 10.1007/978-3-642-27919-5.
[6]. OWASP Application Security Verification Standard, v. 4.0.3. 2021.
[7]. Dougherty, C., Sayre, K., Seacord, R.C., Svoboda, D., and Togashi, K. Secure Design Patterns. Technical Report CMU/SEI-2009-TR-010, Software Engineering Institute, 2009. DOI: 10.1184/R1/6583640.v1.
[8]. Fernandez-Buglioni, E. Security Patterns in Practice: Designing Secure Architectures Using Software Patterns, 2013. ISBN: 9781119998945.
[9]. Washizaki, H., Xia, T., Kamata, N., Fukazawa, Y., Kanuka, H., and Yamaoto, D. Taxonomy and Literature Survey of Security Pattern Research. Proc. of 2018 IEEE Conference on Application, Information and Network Security (AINS), pp. 87-92, 2018. DOI: 10.1109/AINS.2018.8631465.
[10]. Jaeger, T. Operating System Security. Synthesis Lectures on Information Security, Privacy, and Trust. Springer, 2008. DOI: 10.1007/978-3-031-02333-0.
[11]. Pearce, M., Zeadally, S., and Hunt, R. Virtualization: Issues, security threats, and solutions. ACM Computing Surveys, 45(2), Article 17, 2013. DOI: 10.1145/2431211.2431216.
[12]. Rushby, J. Design and Verification of Secure Systems. 8-th ACM Symposium on Operating System Principles, pp. 12-21, Asilomar, CA, December 1981. DOI: 10.1145/800216.80658.
[13]. Rushby, J. Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999.
[14]. Aeronautical Radio, Inc. (ARINC). Avionics Application Software Standard Interface, Part 0, Overview of ARINC 653, ARINC 653P0-2, August 2019.

[15]. DeLong, R.J., Rudina, E. MILS Architectural Approach Supporting Trustworthiness of the IIoT Solutions. Whitepaper, Industrial Internet Consortium, 2021.
[16]. Patra, P.K., and Pradhan, P.L. Hardening of UNIX Operating System. International Journal of Computer and Communication Technology 1(1), Article 9, 2010. DOI: 10.47893/ijcct.2010.1008.
[17]. Open AADL. URL: http://www.openaadl.org/ (доступ 05.12.2024).
[18]. Santos, J.C.S., Tarrit, K., and Mirakhorli, M. A catalog of security architecture weaknesses. In 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), pp. 220-223, 2017. DOI: 10.1109/ICSAW.2017.25.
[19]. MITRE. Common Weakness Enumeration, 2022. URL: https://cwe.mitre.org/index.html (доступ 01.11.2024).
[20]. MITRE. Common Vulnerabilities and Exposures, 2024. URL: https://www.cve.org (доступ 01.11.2024).
[21]. Microsoft Security Development Lifecycle. URL: https://www.microsoft.com/en-us/securityengineering/sdl/ (доступ 01.11.2024).
[22]. Microsoft Threat Modeling Tool. URL: https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool (доступ 01.11.2024)
[23]. Almorsy, M., Grundy, J., and Ibrahim, A.S. Automated software architecture security risk analysis using formalized signatures. Proc. of 35-th International Conference on Software Engineering (ICSE), pp. 662-671, San Francisco, CA, USA, 2013. DOI: 10.1109/ICSE.2013.6606612.
[24]. Frydman, M., Ruiz, G., Heymann, E., César, E., and Miller, B.P. Automating Risk Analysis of Software Design Models. The Scientific World Journal, June 2014, pp. 248-259. DOI: 10.1155/2014/805856.
[25]. Nafees, T., Coull, N., Ferguson, I., and Sampson, A. Vulnerability Anti-Patterns: a Timeless Way to Capture Poor Software Practices (Vulnerabilities). Proc. of 24-th Conference on Pattern Languages of Programs. The Hillside Group, ACM, 23, 2017.
[26]. Seifermann, S., Heinrich, R., and Reussner, R. Data-Driven Software Architecture for Analyzing Confidentiality. Proc. of IEEE International Conference on Software Architecture (ICSA), pp. 1-10. Hamburg, Germany, 2019. DOI: 10.1109/ICSA.2019.00009.
[27]. IDRIS. URL: https://www.idris-lang.org/ (доступ 05.12.2024).
[28]. Siu, K., Moitra, A., Li, M., Durling, M., Herencia-Zapana, H., and Interrante, J. Architectural and Behavioral Analysis for Cyber Security. In 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC), San Diego, CA, USA, 2019, pp. 1-10. DOI: 10.1109/DASC43569.2019.9081652.
[29]. VERDICT Project. URL: https://ge-high-assurance.github.io/VERDICT/ (доступ 05.12.2024).

Информация об авторах / Information about authors

Виктор Вячеславович КУЛЯМИН – кандидат физико-математических наук, доцент кафедры Системного программирования ВМК МГУ, ведущий научный сотрудник ИСП РАН. Сфера научных интересов: программная инженерия, тестирование на основе моделей, формальные методы программной инженерии, формальные методы обеспечения безопасности.

Victor Viatcheslavovitch KULIAMIN – Cand. Sci. (Phys.-Math.), Associate Professor of System Programming Department, Faculty of Computational Mathematics and Cybernetics Moscow State University, Leading Researcher of the Institute for System Programming, Russian Academy of Sciences. Research interests: software engineering, model-based testing, formal methods of software engineering, formal methods of software security assurance.

Александр Константинович ПЕТРЕНКО — доктор физико-математических наук, профессор кафедры Системного программирования ВМК МГУ, заведующий отделом Технологий программирования ИСП РАН. Его научные интересы включают формальные методы программной инженерии, языки спецификаций и моделирования, их применение для поддержки разработки и верификации программного обеспечения.

Alexander Konstantinovich PETRENKO — Dr. Sci. (Phys.-Math.), Professor of System Programming Department, Faculty of Computational Mathematics and Cybernetics Moscow State University, Head of Software Engineering department of the Institute for System Programming, Russian Academy of Sciences. His research interests include formal methods of software

engineering, specification and modeling languages, and their use in software development and verification.

Екатерина Александровна РУДИНА – кандидат технических наук в области компьютерной и сетевой безопасности, аналитик. Работает в Департаменте перспективных технологий «Лаборатории Касперского» в области исследования угроз и оценки рисков для систем критической инфраструктуры. Принимает участие в разработке национальных стандартов и стандартов ISO, IEEE, рекомендаций ITU и Консорциума промышленного Интернета вещей. Научные интересы включают методы моделирования угроз информационной безопасности, подходы к проектированию и реализации систем с заданными свойствами безопасности и устойчивости, методы обнаружения компьютерных атак.

Ekaterina Alexandrovna RUDINA – Cand. Sci. (Tech.) in Computer and Network Security, the analyst who works for the Kaspersky in the scope of threat research and risk assessment for the systems of critical information infrastructure. Ekaterina is a contributor to ISO, IEEE, ITU, and Industrial Internet Consortium documents and to national standards. Her research interests cover the methods of threat modeling, approaches to secure and safe systems engineering, and intrusion detection methods.