DOI: 10.15514/ISPRAS-2021-33(5)-2

# Формирование методологии разработки безопасного системного программного обеспечения на примере операционных систем

[1] *П.Н. Девянин, ORCID: 0000-0003-2561-794X <pdevyanin@astralinux.ru>*
[1] *В.Ю. Тележников, ORCID: 0000-0002-6192-2856 <vtelezhnikov@astralinux.ru>*
[2,3,4,5] *А.В. Хорошилов, ORCID: 0000-0002-6512-4632 <khoroshilov@ispras.ru>*
[1] *ООО «РусБИТех-Астра»,*
*117105, г. Москва, Варшавское ш., д. 26, стр.11*
[2] *Институт системного программирования имени В.П. Иванникова РАН,*
*109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[3] *Московский государственный университет имени М.В. Ломоносова,*
*119991, Россия, Москва, Ленинские горы, д. 1*
[4] *НИУ Высшая школа экономики,*
*101978, Россия, г. Москва, ул. Мясницкая, д. 20*
[5] *Московский физико-технический институт,*
*141701, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*

**Аннотация.** Разработка безопасного системного программного обеспечения (ПО), на основе которого строятся сертифицированные средства защиты информации, достижение доверия к нему согласно требованиями нормативных документов отечественных регуляторов – крупная научно-техническая проблема. Возможным путем ее решения является формирование соответствующей методологии, которая должна включить передовые научные результаты в области информационной безопасности и системного программирования и отразить лучшие практики разработки такого ПО. В статье рассматриваются текущие результаты формирования этой методологии, которое осуществляется по следующим направлениям. Во-первых, это деятельность по развитию нормативной базы в области разработки и обеспечения доверия к безопасному системному ПО, включая создание профильных национальных стандартов. Во-вторых, разработка и верификация формальных моделей управления доступом, как основы механизмов защиты, входящих в состав системного ПО и составляющих его поверхность атаки. В-третьих, методы и технологии статического и динамического анализа программного кода системного ПО с учётом его специфики. В-четвертых, способы сбора и аналитической обработки данных, получаемых в ходе анализа программного кода системного ПО. Все направления формирования методологии иллюстрируются примерами ее практического применения и апробации при разработке ОС семейства Linux, в особенности сертифицированной по высшим классам защиты и уровням доверия операционной системы специального назначения Astra Linux Special Edition.

**Ключевые слова:** системное программное обеспечение; формальная модель управления доступом; верификация; статический и динамический анализ кода; операционная система; Astra Linux

**Для цитирования:** Девянин П.Н., Тележников В.Ю., Хорошилов А.В. Формирование методологии разработки безопасного системного программного обеспечения на примере операционных систем. Труды ИСП РАН, том 33, вып. 5, 2021 г., стр. 25-40. DOI: 10.15514/ISPRAS–2021–33(5)–2.





# Building a methodology for secure system software development on the example of operating systems

[1] *P.N. Devyanin, ORCID: 0000-0003-2561-794X <pdevyanin@astralinux.ru>*
[1] *V.Y. Telezhnikov, ORCID: 0000-0002-6192-2856 <vtelezhnikov@astralinux.ru>*
[2,3,4,5] *A.V. Khoroshilov, ORCID: 0000-0002-6512-4632 <khoroshilov@ispras.ru>*
[1] *RusBITech-Astra*
*26, Varshavskoe, Moscow, 117105, Russia*
[2] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
*25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[3] *Lomonosov Moscow State University,*
*GSP-1, Leninskie Gory, Moscow, 119991, Russia*
[4] *National Research University, Higher School of Economics*
*20, Myasnitskaya Ulitsa, Moscow, 101978, Russia*
[5] *Moscow Institute of Physics and Technology (State University),*
*9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russian Federation*

**Abstract.** System software is a cornerstone of any software system, so building secure system software in accordance with requirements of certification authorities and state-of-the-art practices is an important scientific and technical problem. One of possible approaches to cope with the problem is to build a methodology for secure system software development including advanced scientific technologies and industry best practices. The paper presents current results achieved in building such methodology in the following directions. The first one is regulatory framework improvement including development of GOST R specifications defining requirements to formal models of access control policies and their formal verification. The second direction is design and verification of formal models of corresponding security functional requirements. The third direction is application of new and well established technologies of static and run-time analysis of systems software. The considered technologies include static analysis, fuzzing, functional and unit testing as well as testing the system software against formal models of its functional security requirements. The forth direction is development of methods for acquisition of results of all kinds of the analysis and for its analytical processing. All the directions are illustrated by practical examples of application of the methodology to development of Astra Linux operating system distribution that is certified according to the highest evaluation assurance levels.

**Keywords:** security development lifecycle; access control; formal methods; verification; static analysis; fuzzing; operating systems; Astra Linux

**For citation:** Devyanin P.N., Telezhnikov V.Y., Khoroshilov V.V. Building a methodology for secure system software development on the example of operating systems**.** Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 5, 2021, pp. 25-40 (in Russian). DOI: 10.15514/ISPRAS–2021–33(5)–2

## *1. Введение*

Разработка безопасного системного программного обеспечения (ПО), достижение доверия к нему является самостоятельной крупной научно-технической проблемой. Это объясняется, во-первых, большими объемами кода, сложностью архитектуры и интерфейсов взаимодействия компонент такого ПО как внутри самой информационной системы, так и со средой ее функционирования. Во-вторых, системное ПО, например, операционные системы (ОС), по сути являются фундаментом, на котором строится безопасность и надежность работы всей системы в целом, в том числе ее прикладного ПО. В-третьих, на основе системного ПО создаются многие отечественные сертифицированные средства защиты информации (СЗИ), которые внедряются в рамках импортозамещения во многих информационных системах органов государственной власти, критической информационной инфраструктуры и коммерческих компаний Российской Федерации. Все это в свою очередь повышает ответственность всех участников процесса разработки системного ПО за

обеспечение его безопасности и доверия к нему, выбор используемых при этом технических решений, научных подходов и методов.

Непрерывно совершенствуемые с участием самих разработчиков ПО, представителей научного сообщества нормативные документы отечественных регуляторов в области разработки безопасного ПО и обеспечения доверия к нему также оказывают существенное влияние на процессы разработки системного ПО. Большое значение здесь имеют нормативные документы ФСТЭК России, в первую очередь базирующиеся на ГОСТ Р 56939-2016 [1] утвержденные Приказом № 76 от 02.06.2020 во второй редакции «Требования по безопасности информации, устанавливающие уровни доверия к средствам технической защиты информации и средствам обеспечения безопасности информационных технологий» (далее – Требования доверия) [2] совместно с детализирующей их утверждённой ФСТЭК России 25.12.2020 второй редакцией «Методики выявления уязвимостей и недекларированных возможностей в программном обеспечении» (далее – Методика выявления уязвимостей) [3].

Эти нормативные документы, уже прошедшие несколько лет апробации при проведении сертификации СЗИ, включают широкий спектр требований, основанных на применении передовых результатов науки и реализующих их технологий. При этом востребованы научные теории, позволяющие с учётом сложности системного ПО моделировать и анализировать различные аспекты его безопасности.

В этом ПО большую роль играет управление доступом к контролируемым им ресурсам, например, для ОС это, как минимум, управление доступом субъектов доступа (процессов) к объектам доступа (файлам, каталогам и т. д.). Поэтому механизм управления доступом часто является основным из входящим в состав системного ПО механизмом защиты, формирующим поверхность атаки [3], и, следовательно, от того, как он реализован, во многом зависит безопасность всего системного ПО. В этой связи теория управления доступом и информационными потоками (скрытыми каналами) по памяти и по времени [4, 5] уже много лет является «локомотивом» разработки безопасного системного ПО. В рамках этой теории на математическом или формализованном (машиночитаемом) языках формируются и верифицируются вплоть до реализации в программном коде системного ПО формальные модели управления доступом [6], что полностью отвечает требованиям действующих нормативных документов.

Несомненно, важнейшей составляющей разработки любого безопасного ПО является обеспечение качества и доверия к его программному коду, которые достигаются в результате применения широкого спектра технологий его анализа и автоматизирующих их инструментальных средств, включая использование лучших международных практик по разработке безопасного ПО, таких как Secure Software Development Lifecycle (SSDL) [7]. Вместе с тем важно рассматривать эти практики с учётом специфики создания и применения системного ПО в основе сертифицированных СЗИ, часто включения в это ПО значительных объемов программного кода, заимствованного из свободного ПО (Open source). В частности, необходимо провести приоритизацию компонентов, исходя из их роли в реализации функциональных требований безопасности и поверхности атаки СЗИ [3], и адаптировать уровень проводимого анализа в зависимости от установленных приоритетов. Наиболее значимыми компонентами для системного ПО, как правило, являются те, что отвечают за реализацию механизма управления доступом. Поэтому анализ программного кода в наиболее тщательном виде должен быть сконцентрирован именно на них. В рамках этого необходимо применение статического анализа (в том числе чувствительного к контексту и путям выполнения) и динамического анализа (включая генерационное и мутационное фаззинг-тестирование, анализ трасс, выявление утечки чувствительных данных) программного кода [1], вобравшего в себя передовые результаты теорий системного программирования, методов верификации, символьных вычислений и даже искусственного интеллекта.





При этом также важно использовать возможности внедрения в таком системном ПО, в применяемых для обеспечения доверия к нему методах и инструментальных средствах результатов научных исследований отечественных специалистов, возможности отразить все перечисленное в новых или актуализируемых нормативных документах.

Причем для системного ПО важен комплексный подход в применении рассматриваемых технологий и методов обеспечения его безопасности, так как фрагментарность здесь (с учётом специфики и сложности такого ПО) приведет к наличию в нем уязвимостей и тем самым сведет на нет все прилагаемые усилия по достижению к нему доверия. Кроме того, значительные объемы кода и сложность системного ПО, как правило, приводят к большим объемам результатов его анализа, обработка которых за приемлемое время и приведение их к виду, позволяющему использовать эти результаты для оперативного исправления выявляемых ошибок, также является проблемой, для решения которой создаются соответствующие технологии и методы. Таким образом, уместно говорить о формировании у нас в стране методологии разработки безопасного системного ПО, направленной на комплексное решение практических проблем в этой предметной области в контексте выполнения требований соответствующих нормативных документов регуляторов.

Участие в этом процессе, с одной стороны, осуществляют представители научного сообщества, ведущих научных центров компетенций в области системного программирования и разработки безопасного ПО. С другой стороны, успех формирования методологии невозможен без использования опыта, технологий и компетенций отечественных разработчиков системного ПО, в том числе ОС и СУБД. При этом данный опыт будет тем ценнее, чем большему составу требований доверия удовлетворяет создаваемое системное ПО, и чем больше его разработчик самостоятельно внедрил и развивает необходим~~ые~~ для этого научно обоснованные технические решения, методы и реализующие их инструментальные средства.

В связи с этим в настоящей статье анализируются результаты совместных усилий по формированию и апробации рассматриваемой методологии специалистами Института системного программирования Российской академии наук им. В.П. Иванникова (ИСП РАН) и ООО «РусБИТех-Астра» (ГК Astra Linux) – разработчика сертифицированной по высшему 1 классу защиты (1 уровню доверия) в системе сертификации ФСТЭК России (а также сертифицированной по высоким классам защиты в системах сертификации ФСБ, Минобороны России и Республики Беларусь) операционной системы специального назначения (ОССН) Astra Linux Special Edition [8, 9].

Статья организована в соответствии с основными направлениями формирования методологии, непосредственное участие в реализации которых принимают авторы, и в первую очередь применительно к разработке сертифицированных ОС. В следующем разделе описывается являющаяся важной частью методологии деятельность по развитию нормативной базы в области разработки и обеспечения доверия к безопасному системному ПО, в том числе по созданию соответствующих национальных стандартов (ГОСТ). В разд. 3 приводится описание основных приемов и методов по разработке и верификации формальных моделей управления доступом, как основы механизмов защиты, входящих в состав системного ПО и составляющих его поверхность атаки. В разд. 4 анализируются особенности применения методов и технологий статического и динамического анализа программного кода системного ПО с учётом его специфики. В разд. 5 рассматривается ряд способов сбора и аналитической обработки данных, получаемых в ходе анализа программного кода системного ПО. Все разделы включают примеры практического применения и апробации формируемой методологии при разработке ОС семейства Linux, в особенности ОССН Astra Linux Special Edition. Заключение завершает статью, в нем приводятся возможные направления дальнейшего совершенствования методологии.

## 2. Развитие нормативной базы

Основополагающим национальным стандартом в области разработки безопасного ПО, в том числе системного, является ГОСТ Р 56939-2016 [1], в котором определены применяемые в этой области базовые технологии и методы. Для содержательного их наполнения, создания условий для эффективного использования на практике либо уже утверждены, либо готовится еще ряд национальных стандартов и нормативных документов (в том числе руководства по реализации мер по разработке безопасного ПО, по проведению статического или динамического анализа ПО и др.). Эта деятельность организуется техническим комитетом по стандартизации «Защита информации» (ТК 362), в состав которого наряду с другими лицензиатами ФСТЭК России, испытательными лабораториями входят ИСП РАН и ГК Astra Linux.

Это позволяет включать в новые нормативные документы положения, основанные на передовых научных исследованиях и при этом апробированные в результате разработки промышленного системного ПО. Однако здесь имеются трудности, которые требуют устранения в рамках формируемой методологии. Они заключаются в необходимости находить оптимальное сбалансированное сочетание таких положений. Во-первых, в нормативные документы необходимо включать требования, отражающие политику регуляторов в области разработки безопасного ПО, стимулирующие развитие отрасли. Во-вторых, целесообразно, чтобы эти положения основывались на лучших практиках в данной области и актуальных научных разработках. В-третьих, важно, чтобы положения нормативных документов проходили всестороннюю проверку на возможность их применения в полном объеме и с учётом специфики системного ПО, имеющего широкое внедрение в реальных информационных системах, включая СЗИ, а не только на макетах такого ПО или только в части стандартизируемых положений.

В этой связи для примера рассмотрим положения ГОСТ 59453.1-2021 «Защита информации. Формальная модель управления доступом. Часть 1. Общие положения» [10] и ГОСТ 59453.2-2021 «Защита информации. Формальная модель управления доступом. Часть 2. Рекомендации по верификации формальной модели управления доступом» [11], непосредственное участие в разработке которых принимали авторы.

Подготовка этих национальных стандартов стала необходимой после включения в 2017 г. компоненты доверия ADV_SPM.1 «Формальная модель политики безопасности» в утвержденные ФСТЭК России профили защиты ОС [12], начиная с 2 класса защиты, и в дальнейшем в 2018 г. включения требований о разработке формальной модели управления доступом и ее верификации с применением инструментальных средств, соответственно, в 4 и 3 уровни доверия первой редакции Требований доверия [2].

На первом шаге разработки этих национальных стандартов были проанализированы лучшие практики использования формальных моделей при реализации механизмов управления доступом в реальных СЗИ, в том числе ОС [9, 14]. Также были рассмотрены актуальные технологии верификации формальных моделей управления доступом с применением инструментальных средств [6, 15]. В результате было принято решение, что в ГОСТ 59453.1-2021 должны быть стандартизованы не конкретная модель или требования, которым должны удовлетворять все формальные модели, а с учётом текущего уровня развития технологий и опыта создания СЗИ, реализующих политики управления доступом, стандартизованы только критерии, которым должны соответствовать описания формальных моделей управления доступом, на основе которых разрабатываются эти СЗИ. При этом в ГОСТ 59453.2-2021 должны быть стандартизованы рекомендации по верификации с применением инструментальных средств формальных моделей управления доступом.

Далее были проанализированы классические и современные формальные модели управления доступом (например, изложенные в [4, 5]), подходы к их описанию. Это позволило определить научные основы разрабатываемых стандартов. В том числе, в ГОСТ 59453.1-2021 указано, что содержание описания формальной модели управления доступом должно быть дано на





математическом или формализованном (машиночитаемом) языках и включать описание состояний и правил перехода между состояниями абстрактного автомата, соответствующего политикам управления доступом, реализуемым моделируемым СЗИ. Это описание должно быть дано как минимум с использованием терминов: объект доступа (объект, контейнер, сущность), учётная запись пользователя, субъект доступа, доступ, право доступа, информационный поток. Также должны быть описаны условия безопасности, выполнение которых в абстрактном автомате указывает на реализацию заданных политик управления доступом. Уверенность в корректности формальной модели управления доступом должна быть достигнута математическим (формальным) доказательством того, что в ней не содержится противоречий, т. е. в абстрактном автомате выполняются условия безопасности.

Поскольку в существующих нормативных документах, технической и научной литературе часто приводятся несогласованные определения терминов, используемых при моделировании управления доступом (например, объект доступа, субъект доступа, учётная запись пользователя и др.), существенных для корректного изложения положений ГОСТ 59453.1-2021, то в этом стандарте потребовалось дать определения большинства таких терминов. При чем определения ряда терминов, например, политика мандатного контроля целостности или объект доступа, ассоциированный функционально с субъектом доступа, были стандартизованы впервые.

Для того чтобы, с одной стороны, дать критерии, которым должны соответствовать описания большинства формальных моделей управления доступом, а, с другой стороны, привести более детальные критерии для описания моделей для СЗИ, реализующих наиболее востребованные на практике виды политик управления доступом, структура ГОСТ 59453.1-2021 была сформирована следующим образом. Каждый раздел стандарта (посвященный, соответственно, описанию в рамках формальной модели управления доступом состояний абстрактного автомата, описанию правил его перехода из состояний в состояния и доказательству выполнения условий безопасности во всех его состояниях и при всех переходах из состояний в состояния) был разбит на пять частей. Первая часть такого раздела предназначалась для изложения критериев, которым должны удовлетворять описания всех формальных моделей управления доступом, а последующие части – для критериев описания моделей для СЗИ, реализующих, соответственно, дискреционное, ролевое, мандатное управление доступом и мандатный контроль целостности.

В свою очередь при изложении в ГОСТ 59453.2-2021 рекомендаций по верификации с применением инструментальных средств формальных моделей управления доступом наибольшее внимание было уделено описанию выполняемых при этом действий, в том числе:

- разработка и применение критериев выбора инструментальных средств верификации;
- перевод (при необходимости) описания модели из математического в формализованное (машиночитаемое) описание;
- автоматическое доказательство непротиворечивости формальной модели и выполнения заданных в ее рамках условий безопасности (условий верификации).

Для придания положениям ГОСТ 59453.1-2021 и ГОСТ 59453.2-2021 большей ясности и обеспечения большего удобства их использования при разработке и верификации формальных моделей управления доступом в текст первого стандарта почти к каждому пункту был добавлен пример, а во втором стандарте примеры собраны в приложении. В этих примерах на классическом математическом языке и на формализованном языке, поддерживаемом формальным методом Event-B [15], поясняются положения обоих стандартов.

Оба стандарта прошли апробацию на примере описания и верификации мандатной сущностно-ролевой ДП-модели управления доступом и информационными потоками в ОС семейства Linux (МРОСЛ ДП-модели) [5, 9], являющейся научной основой механизма управления доступом ОССН Astra Linux Special Edition.

Таким образом, при разработке ГОСТ 59453.1-2021 и ГОСТ 59453.2-2021, по мнению авторов, удалось, руководствуясь политикой регуляторов, учесть лучшие практики при описании и верификации формальных моделей управления доступом, а также современные научные теории и решения в этой области. Выработанные при этом апробированные подходы являются частью формируемой методологии разработки безопасного системного ПО.

## 3. Разработка и верификация формальных моделей управления доступом

Как уже было отмечено, управление доступом к контролируемым ресурсам является одной из ключевых функций системного ПО. Поэтому реализация механизма управления доступом с использованием соответствующей формальной модели согласно требованиям нормативных документов регуляторов становится неотъемлемой частью разработки безопасного системного ПО. При этом, как правило, интерфейсы этого механизма составляют поверхность атаки, а, значит, при его реализации следует применять весь спектр средств анализа его программного кода. Это в совокупности предоставляет возможность выработки и апробации комплексных научно обоснованных технологий и методов обеспечения доверия как к самому механизму управления доступом, так и распространения этого доверия на все системное ПО в целом.

Очевидно, что первым шагом при разработке формальной модели управления доступом должен быть анализ известных моделей, их теоретических основ, опыта реализации в системном ПО, в том числе СЗИ, с целью обоснования возможности использования какой-либо существующей модели, комбинации таких моделей, либо необходимости формирования новой модели.

В соответствии с ГОСТ 59453.1-2021 следующим шагом, когда разрабатывается новая формальная модель, является определение видов реализуемых СЗИ политик управления доступом. Это, с одной стороны, задает все дальнейшее содержание модели, с другой стороны, поскольку в большинстве реальных СЗИ используются несколько политик (например, часто дискреционное или ролевое управление доступом сочетается с мандатным управлением доступом), то важно при объединении в рамках модели нескольких политик сделать это не «механически», т.е. в способах такого объединения должны учитываться свойства каждой из политик, чтобы при реализации модели в СЗИ одни политики не противоречили другим. Примером проявления такого противоречия являются запрещенные политикой мандатного управления доступом информационные потоки (скрытые каналы) по времени от сущностей с большим уровнем конфиденциальности к сущностям с меньшим уровнем конфиденциальности, возникающие за счет использования параметров дискреционного или ролевого управления доступом [5].

Дальнейшим шагом является собственно разработка с использованием ГОСТ 59453.1-2021 самой формальной модели управления доступом и ее верификация с применением инструментальных средств согласно ГОСТ 59453.2-2021. Схема этого процесса и рекомендации по его выполнению изложены в указанных стандартах, а также в научной и учебной литературе (например, в [4-6]), вобравшей в себя уже почти полувековой опыт создания и верификации формальных моделей вплоть до их реализации в программном коде СЗИ. Самым трудным здесь является найти баланс между выразительностью формальной модели, обеспечением ее адекватности реальному СЗИ и чрезмерным усложнением модели, что сделает технически сложными ее верификацию и доказательство в ее рамках условий безопасности. Кроме того, при этом следует учитывать, что, как и моделируемое СЗИ, которое в большинстве случаев будет дорабатывается, так и формальная модель должна будет развиваться, поэтому потребуют многократного повторения и доказательства, и верификация.

Для примера рассмотрим опыт формирования МРОСЛ ДП-модели, на основе которой в подсистеме безопасности PARSEC [9] реализуется механизм управления доступом ОССН Astra





Linux Special Edition (в ОССН не применяются аналогичные заимствованные иностранные механизмы защиты, в том числе SELinux и AppArmor). В начале разработки этой модели в 2012 г. анализ существовавших на тот момент формальных моделей управления доступом показал отсутствие готовой модели, которую можно было использовать без существенной переработки. Поэтому было принято решение на основе семейства ДП-моделей и ряда классических моделей создать новую ДП-модель и объединить в ней мандатное управление доступом, мандатный контроль целостности и ролевое управление доступом. Целесообразность первой политики следовала из области применения ОССН как СЗИ, которое может обрабатывать информацию различных уровней конфиденциальности. Необходимость мандатного контроля целостности обосновывалась его эффективностью при защите целостности программной среды ОС, что подтверждалось практикой его внедрения с 2007 г. в ОС семейства Microsoft Windows. Ролевое управление доступом рассматривалось как мощная, гибкая и перспективная политика, с помощью которой легко выражается традиционное для ОС семейства Linux дискреционное управление доступом. При этом все три политики сочетались в МРОСЛ ДП-модели с учётом их свойств, не противореча друг другу, в том числе, не создавая условий для возникновения запрещенных информационных потоков по памяти или по времени.

Практически сразу МРОСЛ ДП-модель формировалась в двух нотациях: математической и формализованной на языке метода Event-B [6, 15]. Модель в первой нотации кроме использования опыта разработки классических моделей управления доступом, предоставляет возможность объективного анализа её корректности любым специалистом в области информационной безопасности, знакомым с языком математики. Например, может быть проверена корректность доказательств безопасности моделируемой системы при выполнении заданных в рамках модели условий безопасности. В формализованной нотации МРОСЛ ДП-модель верифицируется с применением инструментальных средств дедуктивно (с помощью Rodin) и по методу проверки моделей (с помощью ProB) [16], что также обеспечивает уверенность в её корректности.

МРОСЛ ДП-модель в обеих нотациях имеет иерархическое представление, состоящее из восьми уровней – четырех уровней для моделирования управления доступом непосредственно в ОССН Astra Linux Special Edition (первый – ролевого управления доступом, второй – мандатного контроля целостности, третий – мандатного управления доступом с информационными потоками по памяти, четвертый – мандатного управления доступом с информационными потоками по времени) и четырех уровней для решения аналогичной задачи в штатной для ОССН СУБД PostgreSQL, что обеспечивает согласованность механизмов управления доступом в ОССН и СУБД, а также возможность анализа безопасности информационных потоков (скрытых каналов) по памяти и по времени между сущностями ОССН и СУБД.

Переход к иерархическому представлению (первоначально модель формировалась монолитно без разделения её описания по видам политик управления доступом) был важным шагом в разработке модели, так как создавал предпосылки, во-первых, для работы с большим объемом описания модели (сейчас в математической нотации оно составляет более 500 страниц текста). Во-вторых, для удобства её представления и верификации в формализованной нотации, где для этого применялись техники пошагового уточнения (refinement) Event-B, давшие возможность разделить модель объемом более 32 тысяч строк кода на последовательность из восьми связанных между собой спецификаций, соответствующих уровням модели в математической нотации. В-третьих, помимо дедуктивной верификации поступательно по уровням спецификаций использовать метод проверки моделей, который из-за проблемы «комбинаторного взрыва» чрезвычайно чувствителен к объему верифицируемой модели. В-четвертых, разделение модели на уровни соответствует подходу по её реализации непосредственно в программном коде подсистемы безопасности PARSEC ОССН Astra Linux Special Edition. В итоге, метод основанный на применении иерархического представления и использовании для этого одновременно математической и формализованной нотаций не только

оправдал себя на примере ОССН, его целесообразно рекомендовать для разработки других сложных формальных моделей управления доступом.

Также важно отметить, что с использованием описания МРОСЛ ДП-модели в формализованной нотации для демонстрации корректности её реализации непосредственно в программном коде ОССН Astra Linux Special Edition на языке ACSL разрабатываются спецификации функций подсистемы безопасности PARSEC, после чего происходит их дедуктивная верификация с применением инструментального средства Frama-C [17]. В результате корректность наиболее существенной для безопасности ОССН части её программного кода доказывается математически, а уже далее, как это будет показано в последующих разделах настоящей статьи, этот код обрабатывается средствами статического и динамического анализа. Таким образом, делается переход от разработки формальной модели с применением в основном математических методов к использованию методов системного программирования для анализа и обеспечения доверия к программному коду, реализующему в ОССН поверхность атаки, что является важной частью формируемой методологии разработки безопасного системного ПО.

## *4. Статический и динамический анализ программного кода*

Согласно ГОСТ Р 56939-2016 [1] статический и динамический анализ программного кода определены как основные меры, используемые при разработке безопасного ПО. Конкретное описание того, как эти меры должны применяться для достижения доверия к такому ПО, изложено в Методике выявления уязвимостей [3]. В том числе, это автоматизированные технологии и методы статического анализа, чувствительного к контексту и путям выполнения и реализующего метод статического символьного выполнения, а для динамического анализа – это в первую очередь технологии и методы мутационного (включая реализацию генетических алгоритмов) и генерационного (включая динамическое символьное выполнение) фаззинг-тестирования, а также анализ трасс выполнения тестируемого ПО и выявление уязвимостей, связанных с утечкой чувствительных данных (например, паролей).

Для системного ПО, такого как ОС или СУБД, ввиду сложности его архитектуры, интерфейсов взаимодействия его компонент, объема кода, часто исчисляемого десятками миллионов строк, адаптация технологий и методов анализа программного кода системного ПО, организация соответствующего технологического процесса имеют существенное значение.

В Методике выявления уязвимостей при осуществлении статического и динамического анализа требуется акцентировать внимание на программном коде, реализующем поверхность атаки. В случае системного ПО определение адекватной поверхности атаки чрезвычайно важно. С одной стороны, включение в нее максимального состава компонент системного ПО и реализуемых ими интерфейсов может сделать задачу полного анализа их программного кода не выполнимой на практике. С другой стороны, необоснованное уменьшение поверхности атаки наверняка приведет к пропуску существенных уязвимостей в таком ПО. В связи с этим для определения поверхности атаки системного ПО следует исходить из всестороннего исследования архитектуры системного ПО, его механизмов защиты и формирования соответствующей модели нарушителя. Здесь обоснованным научно и подтвержденным практикой можно считать подход, предполагающий определение механизма управления доступом, других связанных с ним механизмов защиты (например, направленных на создание изолированной программной среды) и их интерфейсов в качестве основы поверхности атаки системного ПО. Это подтверждается тем, что прежде чем нарушитель сможет проэксплуатировать уязвимость в некоторой компоненте реализующего развитые механизмы защиты системного ПО, ему в большинстве случаев потребуется преодолеть их, т. е. «пробить» поверхность атаки. Аналогично, если даже нарушителю удастся воспользоваться уязвимостью функционирующего в среде системного ПО прикладного ПО, то для дальнейшего распространения атаки на всю систему, захвата контроля над ней, иных атакующих действий,





например, реализации утечки конфиденциальных данных, ему, как правило, потребуется преодолеть механизмы защиты системного ПО.

Рассмотрим опыт применения статического и динамического анализа программного кода при разработке безопасного системного ПО на примере ОССН Astra Linux Special Edition. Для автоматизации её процессов сборки, тестирования и верификации используется практика непрерывной интеграции (continuous integration) в инструментальной среде GitLab. В этой среде развёрнут специальный компьютерный полигон, названный «Стендом доверия», являющийся комплексом современных инструментальных средств применения научно обоснованных технологий безопасной разработки и обеспечения доверия к ОССН, а также сбора и аналитической обработки выявленных ошибок в программном коде и результатов их устранения.

В качестве основных инструментальных средств статического анализа кода ОССН на «Стенде доверия» применяются cppcheck [18], Clang Static Analyzer [19] и разработанное ИСП РАН инструментальное средство Svace [20].

С учётом специфики реализации основного механизма управления доступом PARSEC в пространстве ядра ОССН для средств статического анализа специалистами ГК Astra Linux уточняются существующие и создаются новые модули тестирования (checkers). Так, например, принимаются во внимание соглашения о вызовах и передаче параметров в функции как непосредственно в ядре ОССН, так и при обращении к модулям ядра, в том числе PARSEC, что значительно сокращает количество «условно ложных» предупреждений об ошибках вида «разыменование нулевого указателя» и им подобных. При этом делаются проверки отсутствия нарушающих данные соглашения путей выполнения, приводящих к вызову потенциально небезопасной функции в условиях отсутствия внутри нее проверок корректности переданных аргументов. Для обеспечения качества проводимого анализа системного ПО, сопоставимого с результатами анализа прикладного ПО, для соответствующих модулей тестирования разрабатываются спецификации функций работы с памятью в пространстве ядра ОССН. Например, в набор спецификаций инструментального средства Svace добавлены описания специализированных функций класса kvmalloc (kvfree), что позволяет отслеживать использование динамически выделяемой памяти и выявлять ошибки при работе с ней, в том числе приводящие к ее «утечке».

При выборе средств динамического анализа ОССН Astra Linux Special Edition для достижения наибольшей эффективности фаззинг-тестирования учитываются условия функционирования каждого из её тестируемых компонентов, в первую очередь подсистемы безопасности PARSEC. Поэтому в качестве основного инструментального средства на «Стенде доверия» используется Syzkaller [21], с применением которого открытым сообществом выявлено множество уязвимостей в различных версиях ядер ОС семейства Linux.

Syzkaller обладает достаточно широкими возможностями и реализует эффективные методы (алгоритмы) фаззинг-тестирования: генерационные алгоритмы для составления начальных целевых тестовых данных на основе предоставленных ему описаний системных вызовов на специализированном языке; различные алгоритмы мутации сгенерированных тестовых значений с целью достижения большего покрытия; генетические алгоритмы для отбора наиболее подходящих («интересных») экземпляров набора сгенерированных значений (корпусов входных данных), например тех, которые либо расширяют покрытие, в том числе в заданном направлении, либо чаще приводят к «падениям». Помимо этого, Syzkaller также обеспечивает возможность минимизации корпуса с целью сокращения содержащихся в нем данных, что значительно повышает эффективность фаззинг-тестирования и позволяет сосредоточиться на участках кода (модулях), непосредственно составляющих поверхность атаки. Благодаря этим возможностям апробированы подходы по уточнению существующих и описанию дополнительных прототипов для генерации на их основе начальных данных фаззинг-тестирования, нацеленных непосредственно на подсистему безопасности PARSEC, что позволило добиться существенного увеличения покрытия (до 70% по базовым блокам)

наиболее «чувствительного» кода с одновременным кратным увеличением сгенерированных для этого наборов тестовых данных.

Для реализации метода регрессионного фаззинг-тестирования авторами развивается инструмент Syzcrawler, который обеспечивает автоматизированное тестирования обнаруженных ранее ошибок (падений) на стендах фаззинга с новыми (доработанными) ядрами ОССН Astra Linux Special Edition с целью подтверждения их успешного устранения. Помимо тестирования ошибок, найденных с помощью Syzkaller, для которых фаззером были построены файлы для воспроизведения с помощью встроенной в него утилиты syz-repro, Syzcrawler поддерживает воспроизведение ошибок по файлам журнала, которые Syzkaller составляет для каждого обнаруженного падения. Также в рамках работы над повышением эффективности регрессионного тестирования и его автоматизацией проходит апробацию метод обнаружения ошибок по шаблонам, составленным на основе уже выявленных в результате фаззинга падений, что позволяет находить типовые ошибки, не дожидаясь результатов динамического анализа.

С целью фаззинг-тестирования отдельных драйверов или приложений пользовательского пространства, в том числе входящих в подсистему безопасности PARSEC, на «Стенде доверия» применяются положительно зарекомендовавшие себя средство Amrican Fuzzy Lop [22] (далее – AFL) и комплекс динамического анализа программ Crusher [23], разрабатываемый ИСП РАН. AFL – фаззер с открытым исходным кодом, в основе которого лежат инструментация кода на этапе компиляции программы и применение генетических и мутационных алгоритмов для поиска новых тестовых случаев и расширения покрытия анализируемого кода. Crusher – это инструментальное средство, реализующее фаззинг-тестирование с использованием генерационных и генетических алгоритмов и динамического символьного выполнения.

Для фаззинг-тестирования приложений пользовательского пространства подсистемы безопасности PARSEC для AFL и Crusher были написаны дополнительные «программы-обертки» (далее – обертки), которые в свою очередь передают полученные от фаззера данные в целевую программу. При этом важно, чтобы обертка принимала любые искаженные значения и имела узкую цель, по этой причине при тестировании было принято решение не использовать одну обертку для целого набора приложений, разделяя между ними входной буфер, а создавать для каждого тестируемого приложения свою. Данный подход позволил значительно повысить эффективность фаззинга и скорость обнаружения новых путей исполнения, что в сочетании с применением составленных для каждого тестируемого компонента словарей входных данных обеспечило высокие результаты покрытия за короткое время работы программного комплекса тестирования на «Стенде доверия». В результате апробации данных инструментальных средств покрытие приложений (библиотек) пользовательского пространства, входящих в PARSEC, достигает 60-80% (в отдельных случаях и выше), а параллельное применение AFL и Crusher обеспечивают повышенную эффективность тестирования и большую уверенность в качестве и достоверности полученных результатов. Следует отметить, что в соответствии с Методикой выявления уязвимостей достижение такого покрытия может характеризовать процедуру фаззинг-тестирования как эффективную.

Для требуемого согласно Методике выявления уязвимостей анализа помеченных данных (taint-анализа) применяется созданное ИСП РАН инструментальное средство Блесна. Оно, например, эффективно позволяет осуществлять поиск путей несанкционированной передачи между программными модулями ОССН пароля пользователя (вместо его затирания в памяти).

Таким образом, применение на «Стенде доверия» совокупности рассмотренных методов, технологий и реализующих их инструментальных средств статического и динамического анализа кода ОССН Astra Linux Special Edition к составляющим поверхность атаки модулям её подсистемы безопасности PARSEC, позволяют обеспечивать высокий уровень доверия к ней на протяжении всего жизненного цикла её разработки, устраняя большинство программных ошибок на его ранних этапах. Это создает условия для распространения доверия далее на весь программный код ОССН.





## *5. Аналитическая обработка результатов анализа*

С учетом специфики системного ПО при проведении анализа его программного кода одной из основных проблем является аналитическая обработка и интерпретация результатов такого анализа. На объёмах программного кода, типичных для системного ПО, а это миллионы строк, инструментальные средства выдают тысячи, а иногда, десятки тысяч предупреждений о возможных ошибках, большая часть из которых, как правило, оказываются ложными или не учитывающими особенности анализируемого кода, что наиболее актуально для модулей или драйверов ядра ОС. Возложение обязанностей по анализу этих предупреждений об ошибках непосредственно на программистов-разработчиков системного ПО резко снизит эффективность их работы. В связи с этим необходимы технологии и методы, реализующие их инструментальные средства, автоматизирующие обработку этих предупреждений, чтобы программистам-разработчикам информация предоставлялась с учётом степени критичности ошибки в виде, максимально точно указывающем на место ее возникновения.

Поскольку большинство инструментальных средств анализа программного кода имеют индивидуальные часто несовместимые форматы представления результатов своей работы, то, конечно, для дальнейшей аналитической обработки этих результатов требуется реализация средств их приведения к единому формату в некоторой общей базе данных. При этом для повышения эффективности аналитической обработки желательно оснащение этой базы данных интерфейсами, позволяющими визуализировать полученные результаты анализа. Например, проводить разметку результатов анализа, сопоставление таких результатов, полученных либо в разное время, либо от разных инструментальных средств.

При организации аналитической обработки следует иметь ввиду, что для системного ПО (особенно ОС) проведение динамического анализа программного кода и главное интерпретация его результатов является более трудоёмкой и технически сложной задачей, чем для статического анализа кода. С одной стороны, общее число сообщений об ошибках (сбоях в работе системного ПО), выдаваемых инструментальными средствами (например, при фаззинг-тестировании) за приемлемое время тестирования, существенно меньше, чем при проведении статического анализа кода. С другой стороны, практически весь дальнейший анализ этих ошибок с большим трудом автоматизируется и в большинстве случаев предполагает непосредственную их обработку («ручной» анализ) аналитиком. Именно поэтому средства динамического анализа чаще всего применяются непосредственно после выхода системного ПО на стадию тестирования, когда ошибки, обнаруженные статическими анализаторами устранены.

Кроме того, в случае, когда большие объемы программного кода системного ПО являются заимствованными из свободного ПО (например, когда оно разрабатывается на основе ОС семейства Linux), имеет большое значение применение технологий и методов статистической обработки результатов анализа. Это позволит выявлять аномальные изменения в таком заимствованном коде, что может указывать на необходимость более детального анализа выявленных в нем ошибок.

Применительно к ОССН Astra Linux Special Edition в рамках рассмотренного в предыдущем разделе специального компьютерного полигона – «Стенда доверия» развернута база данных, названная «Базой данных доверия». В ней собираются данные от всех инструментальных средств анализа программного кода, осуществляется их аналитическая обработка, для чего специалистами ГК Astra Linux разрабатываются соответствующие средства автоматизации, в том числе включающие интерфейсы визуализации результатов такой обработки.

Для обеспечения возможности реализации единого интерфейса работы с предупреждениями, полученными от различных инструментальных средств статического, динамического и других видов анализа, с привязкой к исходному коду тестируемых компонентов, осуществляется автоматизированное сопоставление полученных результатов по значительному числу параметров, включая тип и шаблон описания ошибки, уровень достоверности, место сработки,

последовательность вызовов, приводящих к ошибке, а так же другие значимые характеристики, перечень которых постоянно расширяется. Помимо этого, в функциональные возможности «Базы данных доверия» входит применение алгоритмов машинного обучения для определения уровня достоверности получаемых результатов анализа. Обучение проводится на достаточно объемных наборах данных, предварительно обработанных аналитиками на предмет выявления ложных срабатываний и корректировки, при необходимости, степени критичности предупреждений, действительно способных привести к возникновению ошибок. Такой подход обеспечивает возможность применения накопленных результатов при анализе как еще не затронутых компонентов прикладного и системного ПО, так и новых версий ранее протестированных пакетов, в исходных кодах которых возможны существенные изменения, как например при переходе на новую версию ядра ОССН.

В ходе аналитической обработки при определении уровня критичности выявленных инструментальными средствами ошибок в программном коде (и, как следствие, приоритетности их устранения) учитывается наличие собственных реализованных в ОССН средств защиты в её подсистеме безопасности PARSEC, формирующей поверхность атаки. Т.е. такие ошибки в PARSEC проходят через самый тщательный анализ. Вместе с тем в «Базе данных доверия» осуществляется сопоставление ошибок, одновременно выявленных средствами статического и динамического анализа, что упрощает процесс оценивания их достоверности и критичности. Для некоторых типов ошибок апробируются подходы, позволяющие на основе результатов, полученных статическими анализаторами, осуществлять автоматизированную генерацию наборов входных данных для средств динамического анализа, выполняющих дополнительную проверку воспроизводимости найденных потенциальных ошибок.

При этом разрабатываются подходы по обработке результатов статического анализа программного кода подсистемы безопасности PARSEC в сочетании с методами и технологиями дедуктивной верификации этого кода с применением инструментального средства Frama-C. В особенности той части кода, которая непосредственно реализует функции управление доступом согласно МРОСЛ ДП-модели. Задание спецификаций таких функций на ACSL и дальнейшая их верификация взаимно дополняют и развивают технологии и методы, используемые при обработке результатов статического анализа.

В итоге программисты-разработчики ОССН Astra Linux Special Edition получают только достоверную, верифицированную аналитиками информацию об ошибках с разъяснением причин возникновения, и, возможно, с предложением способов их устранения. Таким образом, по сути требование отсутствия (минимизации до определённой степени критичности) выявляемых инструментальными средствами статического и динамического анализа ошибок в программном коде становится для разработчика ОССН стандартом качества.

### *6. Заключение и направления развития*

В настоящей статье проанализированы основные направления формирования методологии разработки безопасного системного ПО, в том числе деятельность по развитию соответствующей нормативной базы, включая профильные национальные стандарты, создание и верификация формальных моделей управления доступом, методы и технологии статического и динамического анализа программного кода системного ПО, аналитической обработки данных, получаемых в ходе анализа программного кода системного ПО. Этот процесс формирования методологии еще далек от своего завершения, требуют совершенствования многие составляющие ее технологии и методы, в особенности направленные на определение глубины поверхности атаки и на анализ программного кода системного ПО с учётом его объемов и сложности. Кроме того, требует расширения спектр поддерживаемых инструментальными средствами анализа кода языков программирования и аппаратных





платформ, в первую очередь базирующихся на отечественных процессорах таких, как «Эльбрус» и «Байкал».

В то же время уже имеющийся опыт успешной апробации методологии при создании СЗИ на основе ОС семейства Linux, включая ОССН Astra Linux Special Edition, подтверждает правильность выбранных направлений её формирования и востребованность в дальнейшем при разработке безопасного системного ПО и обеспечении доверия к нему.

## Список литературы / References


[1] ГОСТ Р 56939-2016. Защита информации. Разработка безопасного программного обеспечения. Общие требования. / GOST R 56939-2016. Information protection. Secure Software Development. General requirements. Federal Agency for Technical Regulation and Metrology, 2016 (in Russian).

[2] Выписка из Требований по безопасности информации, утвержденных приказом ФСТЭК России от 2 июня 2020 г. N 76 / Excerpts from Requirements for information security approved by FSTEK Russia order #76 of 2nd June 2020. Available at: https://fstec.ru/tekhnicheskaya-zashchita-informatsii/dokumenty-po-sertifikatsii/120-normativnye-dokumenty/2126-vypiska-iz-trebovanij-po-bezopasnosti-informatsii-utverzhdennykh-prikazom-fstek-rossii-ot-2-iyunya-2020-g-n-76, accessed 14.11.2021 (in Russian).

[3] Информационное сообщение ФСТЭК России от 10.02.2021 № 240/24/647 / Informational message of FSTEK Russia of 10th February 2021 #240/24/647. Available at: https://fstec.ru/normotvorcheskaya/informatsionnye-i-analiticheskie-materialy/2171-informatsionnoe-soobshchenie-fstek-rossii-ot-10-fevralya-2021-g-n-240-24-647, accessed 14.11.2021 (in Russian).

[4] Bishop M. Computer Security: art and science. Addison-Wesley Professional, 2002. 1084 p.

[5] Девянин П.Н. Модели безопасности компьютерных систем. Управление доступом и информационными потоками. Учебное пособие для вузов. М., Горячая линия – Телеком, 2020 г.. 352 стр. / P.N. Devyanin. Security models of computer systems. Control for access and information flows. Hotline-Telecom, 2013, 338 p. (in Russian).

[6] Девянин П.Н., Ефремов Д.В. и др. Моделирование и верификация политик безопасности управления доступом в операционных системах. М., Горячая линия – Телеком, 2019 г., 214 стр. / P.N. Devyanin, D.V. Efremov et al. Modeling and verification of access control access policies in operating systems. Hotline-Telecom, 2019, 214 p. (in Russian).

[7] Microsoft Security Development Lifecycle. Available at: https://www.microsoft.com/en-us/securityengineering/sdl, accessed 14.11.2021.

[8] Операционная система специального назначения Astra Linux Special Edition / Astra Linux Special Edition operating system. Available at: https://astralinux.ru/products/astra-linux-special-edition, accessed 14.11.2021.

[9] П.В. Буренин, П.Н. Девянин и др. Безопасность операционной системы специального назначения Astra Linux Special Edition. Учебное пособие для вузов. М., Горячая линия – Телеком, 2019 г., 404 стр. / P.V. Burenin, P.N. Devyanin et al. Information security with Astra Linux Special Edition. Hotline-Telecom, 2019, 404 p. (in Russian).

[10] ГОСТ Р 59453.1-2021. Защита информации. Формальная модель управления доступом. Часть 1. Общие положения. / GOST R-59453.1-2021. Information protection. Formal access control model. Part 1. General principles (in Russian).

[11] ГОСТ Р 59453.2-2021. Защита информации. Формальная модель управления доступом. Часть 2. Рекомендации по верификация формальной модели управления доступом / GOST R-59453.2-2021 «Information protection. Formal access control model. Part 2. Recommendations on verification of formal access control model» (in Russian).

[12] Документы по сертификации средств защиты информации и аттестации объектов информатизации по требованиям безопасности информации / Documents for certification of information security software and attestation of information systems according to information security requirements. Available at: http://fstec.ru/tekhnicheskaya-zashchita-informatsii/dokumenty-po-sertifikatsii/120-normativnye-dokumenty, accessed 14.11.2021 (in Russian).

[13] Информационное сообщение ФСТЭК России от 29.03.2019 № 240/24/1525 / Informational message of FSTEK Russia 29th March 2021 #240/24/1525. Available at: https://fstec.ru/component/attachments/download/2286, accessed 14.11.2021 (in Russian).

[14] Буренков В.С., Кулагин Д.А. Модель мандатного контроля целостности в операционной системе KasperskyOS. Труды ИСП РАН, том 32, вып. 1, 2020 г, стр. 27-56 / Burenkov V.S., Kulagin D.A. A Mandatory Integrity Control Model for the KasperskyOS Operating System. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 1, 2020. pp. 27-56 (in Russian). DOI: 10.15514/ISPRAS-2020-32(1)-2.
[15] Abrial J.-R., Butler M. et al. Rodin: An Open Toolset for Modelling and Reasoning in Event-B. International Journal on Software Tools for Technology Transfer, vol. 12, no. 6. 2010, pp. 447-466.
[16] Девянин П.Н., Леонова М.А. Приёмы по доработке описания модели управления доступом ОССН Astra Linux Special Edition на формализованном языке метода Event-B для обеспечения её автоматизированной верификации с применением инструментов Rodin и ProB. Прикладная дискретная математика, no. 52, 2021 г., стр. 83-96 / Devyanin P.N., Leonova M.A. The techniques of formalization of OS Astra Linux Special Edition access control model using Event-B formal method for verification using Rodin and ProB. Applied discrete mathematics, no. 52, 2021, pp. 83–96 (In Russian).
[17] Kirchner F., Kosmatov N. et al. Frama-C: a software analysis perspective. Formal Aspects of Computing, vol. 27, issue 3, 2015, pp. 573-609.
[18] Cppcheck. A tool for static C/C++ code analysis. Available at: http://cppcheck.sourceforge.net, accessed 14.11.2021.
[19] Clang Static Analyzer. Available at: https://clang-analyzer.llvm.org/, accessed 14.11.2021.
[20] Статический анализатор Svace. URL: http://www.ispras.ru/technologies/svace / SVACE static analyzer. Available at: https://www.ispras.ru/en/technologies/svace/, accessed 14.11.2021.
[21] Syzkaller project. Available at: https://github.com/google/syzkaller, accessed 14.11.2021.
[22] American Fuzzy Loop. Available at: https://github.com/google/AFL, accessed 14.11.2021.
[23] Комплекс динамического анализа программ Crusher. URL: https://www.ispras.ru/technologies/crusher / ISP Crusher: a dynamic analysis toolset. Available at: https://www.ispras.ru/en/technologies/crusher/, accessed 14.11.2021.

## Информация об авторах / Information about authors

Петр Николаевич ДЕВЯНИН – член-корреспондент Академии криптографии России, доктор технических наук, профессор, научный руководитель ООО "РусБИТех-Астра" (ГК Astra Linux). Область интересов: теория информационной безопасности, формальные модели безопасности компьютерных систем.

Petr Nikolaevich DEVYANIN – Doctor of Technical Sciences, corresponding member of Russian Academy of Cryptography, professor, scientific director in RusBITech-Astra (Astra Linux). Field of Interest: information security theory, formal security models of computer systems.

Владимир Юрьевич ТЕЛЕЖНИКОВ – кандидат технических наук, начальник отдела научных исследований ООО "РусБИТех-Астра" (ГК Astra Linux). Область интересов: формальные модели безопасности компьютерных систем, методы статического и динамического анализа программного кода, операционная система Linux.

Vladimir Iurevich TELEZHNIKOV, Ph.D in Technical Science, Head of Research Department of the RusBITech-Astra (Astra Linux). Field of Interest: formal security models of computer systems, methods of static and dynamic analysis of program code, Linux operating system.

Алексей Владимирович ХОРОШИЛОВ, ведущий научный сотрудник, кандидат физико-математических наук, директор Центра верификации ОС Linux в ИСП РАН, доцент кафедр системного программирования МГУ, ВШЭ и МФТИ. Основные научные интересы: методы проектирования и разработки ответственных систем, формальные методы программной инженерии, методы верификации и валидации, тестирование на основе моделей, методы анализа требований, операционная система Linux.

Alexey Vladimirovich KHOROSHILOV, Leading Researcher, Ph.D. in Physics and Mathematics, Director of the Linux OS Verification Center at ISP RAS, Associate Professor of System Programming Departments at Moscow State University, the Higher School of Economics, and Moscow Institute of Physics and Technology. Main research interests: design and development





methods for critical systems, formal methods of software engineering, verification and validation methods, model-based testing, requirements analysis methods, Linux operating system.