DOI: 10.15514/ISPRAS-2026-38(5)-4

Способ выявления нарушений безопасности (на примере вирусов-шифровальщиков) ОС Astra Linux с использованием машинного обучения

И.О. Кемаев, ORCID: 0009-0005-6206-4119 <*ikemaev@astralinux.ru*>
П.Н. Девянин, ORCID: 0000-0003-2561-794X <*pdevyanin@astralinux.ru*>
ООО «РусБИТех-Астра»,
Россия, 127254, г. Москва, Огородный пр-д, д. 16/1с5.

Аннотация. Предложен использующий машинное обучение (МО) и учитывающий специфику работы средств защиты (в первую очередь механизма мандатного контроля целостности – МКЦ) ОС Astra Linux способ выявления нарушений ее безопасности на примере детектирования активности вирусов-шифровальщиков. Способ базируется на гранулярном анализе средствами МО потоков событий системных вызовов, сбор и обработка параметров которых (включая параметры МКЦ) осуществляется разработанной авторами системой мониторинга. В качестве входных данных используемой в этой системе модели МО применяются: типы системных вызовов, их аргументы и специфичные для МКЦ параметры (уровни целостности процессов, файлов, каталогов, их специальные флаги). При этом включенный в систему мониторинга синтезированный классификатор на базе двунаправленной рекуррентной нейронной сети BiGRU продемонстрировал высокую эффективность в выявлении признаков деструктивного поведения нарушителя, в том числе в сценариях попыток обхода МКЦ, сопровождаемых многочисленными отказами в доступе. Практическая апробация системы мониторинга подтвердила возможность оперативного обнаружения активности вирусов-шифровальщиков на ранних стадиях компрометации данных при минимальном уровне ложноположительных срабатываний. Результаты исследования показали перспективность применения методов МО для автоматизации процессов мониторинга и обеспечения безопасности защищенных отечественных ОС с учетом специфики их средств защиты.

Ключевые слова: машинное обучение; операционная система; мандатный контроль целостности; МРОСЛ ДП-модель; вирус-шифровальщик; Astra Linux.

Для цитирования: Кемаев И.О., Девянин П.Н. Способ выявления нарушений безопасности (на примере вирусов-шифровальщиков) ОС Astra Linux с использованием машинного обучения. Труды ИСП РАН, том 38, вып. 5, 2026, стр. 51–72. DOI: 10.15514/ISPRAS–2026–38(5)–4.

Method for detecting security violations (using the example of ransomware) in the Astra Linux OS using machine learning

I.O. Kemaev, ORCID: 0009-0005-6206-4119 <*ikemaev@astralinux.ru*>
P.N. Devyanin, ORCID: 0000-0003-2561-794X <*pdevyanin@astralinux.ru*>
RusBITech-Astra,
16/1c5, Ogorodnyy Proyezd, Moscow, 127254, Russia.

Abstract. This paper proposes a method for detecting security violations in the Astra Linux OS, leveraging machine learning (ML) and accounting for the specific features of its security tools, primarily the Mandatory Integrity Control (MIC) mechanism. The method focuses on ransomware activity detection and is based on a granular analysis of system call event streams. Data collection and processing, including MIC-specific parameters, are performed by an author-developed monitoring system. The model uses a comprehensive input dataset consisting of system call types, their arguments, and MIC-specific attributes, such as integrity levels for processes, files, and directories, as well as their associated security flags. The synthesized classifier, based on a Bidirectional Gated Recurrent Unit (BiGRU) neural network, demonstrated high efficiency in identifying destructive patterns, including scenarios of security violation attempts accompanied by multiple access denials. Practical evaluation of the monitoring system confirmed its capability for rapid ransomware detection at early stages of data compromise, while maintaining a minimal false positive rate. The research results prove the potential of applying ML methods to automate monitoring processes and enhance information security in secure domestic operating systems.

Keywords: machine learning; operating system; mandatory integrity control; MROSL DP-model; ransomware; Astra Linux.

For citation: Kemaev I.O., Devyanin P.N. Method for detecting security violations (using the example of ransomware) in the Astra Linux OS using machine learning. Trudy ISP RAN/Proc. ISP RAS, vol. 38, issue 5, 2026. pp. 51-72 (in Russian). DOI: 10.15514/ISPRAS-2026-38(5)-4.

1. Введение

Аудит событий информационной безопасности, выявление попыток ее нарушения традиционно являются основными задачами средств защиты информации (СЗИ), включая операционные системы (ОС). Для повышения эффективности используемых для этого технологий в настоящее время все активнее применяются методы машинного обучения (МО) или искусственного интеллекта (ИИ) [1-2], без которых уже трудно представить современные средства обнаружения и предотвращения вторжений (IPS/IDS) [3], сетевых атак [4] или антивирусной защиты [5]. Обобщающая способность моделей МО часто позволяет выявлять признаки вредоносной активности нарушителя без глубокого анализа содержимого собираемых в защищаемой системе данных, что критически важно для предотвращения, в том числе новых атак в режиме реального времени.

Вместе с тем применительно к ОС использование их СЗИ технологий, реализующих методы МО, помимо очевидных преимуществ сопряжено с рядом существенных трудностей. Архитектура большинства ОС, многообразие функционирующего в их среде системного и прикладного программного обеспечения (ПО), обрабатываемых им данных расширяют возможности нарушителя по маскировке своих действий на фоне штатной активности легального ПО, затрудняя использование МО для мониторинга событий информационной безопасности, фильтрации, разметки и обработки существенных для обнаружения атак нарушителя данных.

Большое значение в этом контексте имеет поиск путей преодоления отмеченных трудностей при применении МО в ОС. Так существуют системы обнаружения аномалий, например, модуль поведенческого анализа Behavioral Anomaly Detection в составе SIEM-системы MaxPatrol [6] или программный комплекс Ankey ASAP [7], ориентированные на широкий спектр ОС семейства Microsoft Windows или семейства Linux, на базе которого строятся многие

отечественные защищенные ОС, включая ОС Astra Linux (операционная система специального назначения Astra Linux Special Edition) [8-9]. Однако подобные системы обнаружения аномалий часто не учитывают специфику механизмов защиты, внедряемых в отечественные ОС.

Особенно это проявляется в случае ОС Astra Linux, в которой ее подсистемой безопасности PARSEC реализуются ряд собственных механизмов защиты, наиболее существенным из которых является мандатный контроль целостности (МКЦ), направленный на защиту целостности программной среды ОС (в том числе ПО, запускаемого с помощью средств контейнерной виртуализации [10]) и обрабатываемых в ОС пользовательских данных [11]. Большинство известных авторам систем обнаружения аномалий не учитывают особенности МКЦ и других механизмов защиты ОС Astra Linux (например, мандатного управления доступом), включая специфичные параметры проходящих через PARSEC системных вызовов (уровни целостности учетных записей пользователей, процессов, файлов, каталогов, сокетов и др., их специальные флаги SSI, IRELAX, IINH, SILEV), возможность изоляции с помощью МКЦ потенциально опасного ПО или содержащих его контейнеров («песочниц») на выделенных промежуточных неиерархических или отрицательных линейных уровнях целостности. Помимо этого, поскольку ОС Astra Linux сертифицирована по установленным нормативными документами ФСТЭК России [12] требованиям высших классов защиты и уровней доверия целесообразно безопасное внедрение в архитектуру ОС реализующих методы МО программных компонент, чтобы их потенциальные уязвимости сами не становились угрозой ее безопасности. Тем самым следует обеспечить доверие к соответствующим технологиям МО [13], что будет полностью соответствовать применяемой методологии разработки ОС Astra Linux как безопасного ПО [14].

В целом современные методы обнаружения активности вирусов-шифровальщиков в ОС семейства Linux условно можно разделить на четыре основные группы: статический и сигнатурный анализ; динамические детерминированные (эвристические) методы; методы классического машинного обучения (МО) на основе агрегированных признаков временных окон; методы глубокого обучения на базе больших языковых моделей.

К первой группе относятся подходы, сочетающие статический анализ запускаемых процессов с обработкой сопутствующей текстовой информации. Так, в работе [15] на первом этапе предлагается проводить сигнатурный анализ бинарных файлов с помощью технологии eBPF, а на втором – использовать текстовый классификатор (например, на базе наивного байесовского алгоритма) для выявления сообщений о выкупе. Главным ограничением таких подходов является их неэффективность против новых, видоизмененных или упакованных модификаций вредоносного ПО, при этом детектирование по текстовым сообщениям происходит на слишком поздних стадиях, когда компрометация пользовательских данных уже завершена.

Ко второй группе относятся экспертные динамические системы контроля (эвристические методы без применения МО). Наиболее известными представителями этой группы являются системы UNVEIL [16] и CryptoDrop [17]. UNVEIL осуществляет низкоуровневый мониторинг ввода-вывода файловой системы на базе драйвера минифильтра, отслеживая Шенноновскую энтропию записываемых данных, сходство версий файлов и шаблоны массового удаления/переименования. Система CryptoDrop выстраивает защиту как систему раннего предупреждения на основе взвешенной линейной формулы пяти поведенческих индикаторов (скорость удаления, энтропия, нечеткое хэширование, тип файлов, характер обхода каталогов). Основным недостатком детерминированных эвристик является высокий уровень ложноположительных срабатываний при выполнении легитимных, но схожих по сигнатурам операций (например, при массовом сжатии файлов архиватором 7-Zip или шифровании дисков средствами PGP). Кроме того, они уязвимы к методам искусственного замедления атак со стороны вредоносного ПО.

Третья группа включает решения, использующие классические алгоритмы МО (Random Forest, SVM, MLP, Logistic Regression), обучаемые на числовых признаках, агрегированных в скользящих временных окнах. В работах [18-21] параметры системных вызовов собираются из

пространства ядра (например, средствами eBPF или overlay-файловых систем, таких как GuardFS [21]) и преобразуются в усредненные за фиксированные интервалы времени векторы показателей: частота вызовов, количество уникальных дескрипторов файлов, изменение энтропии и соотношение операций чтения/записи. При этом в [19] предложен перенос инференса дерева решений напрямую в ядро ОС через eBPF-код, а в [20] агрегирование привязывается к идентификатору родительского процесса (PPID). Несмотря на высокую вычислительную скорость, данные методы обладают двумя ключевыми недостатками. Во-первых, агрегация событий во временные интервалы «уничтожает» последовательность выполнения системных вызовов и затрудняет обработку их точных семантических аргументов. Данное ограничение оконных подходов было на практике подтверждено авторами на этапе выбора модели в разд. 4.1, где алгоритм градиентного бустинга LightGBM продемонстрировал критический рост ложноположительных срабатываний из-за невозможности эффективной оценки сложных семантических признаков. Во-вторых, рассмотренные модели разработаны для стандартных дистрибутивов ОС семейства Linux и не учитывают специфику работы собственных механизмов защиты ОС Astra Linux. В их признаковое пространство не интегрированы уровни целостности субъектов и объектов, а также специальные битовые флаги и результаты доступа, генерируемые подсистемой безопасности PARSEC.

К четвертой группе относятся подходы на основе глубокого обучения больших языковых моделей, адаптированные для анализа трасс системных вызовов (например, собираемых через утилиту perf trace) [22]. В таких системах последовательность вызовов токенизируется и представляется в виде связного текста, где зависимости выявляются механизмами внимания. Однако для снижения размерности входных данных из таких трасс, как правило, полностью исключаются аргументы системных вызовов, что приводит к потере важного семантического контекста файловых операций. Кроме того, обладая высокой обобщающей способностью, данные решения требуют критически больших объемов вычислительных ресурсов (CPU, RAM, GPU), что делает невозможным их инференс в реальном времени в среде отдельного экземпляра ОС без значительной потери её общей производительности.

Предлагаемый авторами способ направлен на преодоление указанных ограничений. В отличие от существующих решений, предложенный подход: базируется на архитектуре двунаправленной рекуррентной нейронной сети BiGRU, которая (в отличие от статистических классификаторов на скользящих окнах) сохраняет и анализирует точный временной порядок гранулярных потоков системных вызовов, при этом оставаясь значительно менее ресурсоемкой, чем большие языковые модели; осуществляет детальный анализ сложных аргументов системных вызовов, включая пути к файлам (каталогам) и дискреционных флагов режимов доступа к ним; интегрирует в единое признаковое пространство специфичные для СЗИ ОС Astra Linux параметры системных вызовов (уровни целостности процессов, файлов, каталогов, специальные битовые флаги и результаты доступа). Это позволяет системе уверенно выявлять признаки несанкционированного шифрования на самых ранних стадиях компрометации данных при приемлемом уровне ложных срабатываний.

В связи с этим авторами проведено исследование по выработке способа выявления нарушений безопасности ОС Astra Linux с использованием МО. При этом в рамках данного исследования в качестве источника нарушений безопасности ОС рассматривались вирусы-шифровальщики, целью которых является преднамеренное шифрование пользовательских данных. Такой выбор был мотивирован тем, что, во-первых, вирусы-шифровальщики представляют существенную угрозу в первую очередь для целостности данных непривилегированных пользователей ОС, чему раньше при разработке МКЦ уделялось недостаточно внимания (хотя независимые исследования [23] показывают уже хорошую защищенность ОС Astra Linux от таких угроз). Во-вторых, при том, что существуют несколько развитых функционирующих в среде отечественных защищенных ОС антивирусных решений, таких как Kaspersky Endpoint Security или Dr.Web [24-27], или исследовательских проектов, например, [28], они в полной мере не учитывают особенности архитектуры и функционирования собственных СЗИ ОС Astra Linux,

включая МКЦ. В-третьих, сама природа вирусов-шифровальщиков, заключающаяся в необходимости им многократно осуществлять доступы к файлам и каталогам ОС, проявляя «активность», упрощает задачу сбора параметров связанных с этим событий системных вызовов достаточного объема для их результативной обработки средствами МО. В-четвертых, как уже отмечалось, механизм МКЦ в ОС Astra Linux не только направлен на защиту целостности системного ПО, он позволяет изолировать недоверенное подверженное атакам вирусов-шифровальщиков пользовательское ПО (например, браузеры, которые обрабатывают самые разные потенциально «опасные» данные из сети Интернет) в специализированных контейнерах-«песочницах», что также упрощает задачу их обнаружения средствами МО.

Кроме того, для сбора связанных с вирусами-шифровальщиками параметров системных вызовов можно воспользоваться технологией eBPF [29], позволяющей запускать специализированную программу мониторинга в привилегированном контексте, то есть на уровне ядра ОС. На основе данной технологии специалистами «Группы Астра» была реализована система сбора специфичных для СЗИ ОС Astra Linux параметров системных вызовов [30], включая перечисленные ранее уровни целостности и специальные флаги, коды возврата при отказах в доступах, нарушающих правила МКЦ, в том числе при несанкционированном взаимодействии непривилегированных низкоцелостных процессов с системной шиной D-Bus и другими системными компонентами. Из этих данных предлагается формировать признаковые пространства для моделей МО.

Непосредственно в части МО авторами были рассмотрены хорошо зарекомендовавшие себя в предметной области методы, основанные на использовании ансамблевых моделей градиентного бустинга и нейронных сетей, адаптированных к временным последовательностям событий аудита [31-32]. Для первоначальной разметки данных для обучения ансамблевой модели было предложено использовать поэлементную разметку каждого обогащенного вектора события, обладающего информацией о контексте окружающих системных вызовов, однако позже при применении нейронных сетей, разметка производилась с помощью присваивания метки аномальности целой группе оригинальных событий системных вызовов.

Для практической апробации предлагаемого способа авторами была реализована использующая описанную систему сбора параметров системных вызовов система мониторинга, которая, учитывая вероятностную природу методов МО, оперативно уведомляла об обнаружении «активности» вирусов-шифровальщиков через графический интерфейс ОС Astra Linux.

Таким образом, статья организована следующим образом. В следующем разделе рассматриваются свойства механизма МКЦ в ОС Astra Linux, существенные для реализации предлагаемого способа выявления нарушений ее безопасности. В разд. 3 описывается процесс формирования набора данных и подходящего признакового пространства для обучения моделей МО. В разд. 4 приводятся результаты исследований по адаптации к применению в способе методов и моделей МО, направленных на выявление вирусов-шифровальщиков. В разд. 5 приводятся результаты практической апробации разработанной системы мониторинга в ОС Astra Linux. Заключение завершает статью, в нем подводятся итоги выполненных исследований и разработок, а также рассматриваются их дальнейшие направления.

2. Механизм МКЦ и возможности его применения

Рассмотрим свойства механизма МКЦ, реализованного в подсистеме безопасности PARSEC, начиная с основного (называемого «Усиленный» или «Воронеж») режима защищенности ОС Astra Linux [9], существенные для разработки предлагаемого способа выявления нарушений ее безопасности с применением МО.

Научной основой механизма МКЦ в ОС Astra Linux является мандатная сущностно-ролевая ДП-модель управления доступом и информационными потоками в ОС семейства Linux (МРОСЛ ДП-модель) [33], в рамках которой осуществляется доказательство условий

безопасности этого механизма. Согласно МРОСЛ ДП-модели для задания уровней целостности учетных записей пользователей, процессов (субъектов), файлов и каталогов (сущностей) в ОС применяется решетка уровней целостности, являющаяся прямым (декартовым) произведением решетки подмножеств множества неиерархических категорий целостности (задаваемых маской из 4 байт или 32 бит) и линейных уровней целостности (задаваемых 1 байтом знаковых чисел от -128 до 127). Также для файлов и каталогов задаются специальные битовые флаги:

- IRELAX – присваивается каталогам. При наличии флага у каталога в него может осуществлять запись процесс с любым уровнем целостности;
- SSI – присваивается файлам и каталогам. При наличии данного флага у файла или каталога доступ к нему на чтение или выполнение разрешается только процессам с уровнем целостности не ниже, чем у данного файла или каталога;
- IINH – присваивается каталогам. Данный флаг позволяет создаваемым в каталоге новым файлам или подкаталогам наследовать уровень целостности от родительского каталога;
- SILEV – присваивается исполняемым файлам. Позволяет процессам в сессии учетной записи пользователя с низким уровнем целостности запускать некоторые процессы, которым для выполнения их функций необходим высокий текущий уровень целостности (например, с помощью флага SILEV помечается исполняемый файл консольной утилиты passwd, запускаемой для изменения пароля низкоцелостной учетной записи пользователя, образ которого хранится в обладающем высоким уровнем целостности файле «/ets/shadow»).

Представление таких уровней целостности и специальных флагов имеет следующий вид: <ilev>:<flag1,flag2,...>:<ilinear> (например, 0x00000002:SSI,IINH:-128), где ilev – неиерархические категории, ilinear – линейный уровень, flag1, flag2 – специальные флаги.

Эти уровни целостности и специальные флаги используются при принятии решений о предоставлении доступов в первую очередь на запись с целью предотвратить получение процессом с меньшим уровнем целостности управления другим процессом с большим уровнем целостности. Например, при предоставлении процессу доступа на запись к файлу проверяется, что биты неиерархических категорий целостности процесса включают биты неиерархических категорий целостности файла, а линейный уровень целостности процесса не ниже линейного уровня целостности файла.

При этом в уровнях целостности ОС Astra Linux более важны неиерархические категории целостности, чем линейный уровень целостности. С их помощью задается принадлежность к важным системным компонентам ОС, например, самый младший бит указывает, что это сетевые службы, следующий за ним – средства виртуализации. Также для каждого экземпляра ОС устанавливается максимальный неиерархический уровень (max_ilev), по умолчанию – 0x0000003F (то есть соответствующее ему десятичное значение – 63). Ни один процесс, функционирующий в среде ОС, файл или каталог, размещенный в ее файловой систем, не могут иметь неиерархический уровень целостности выше ее max_ilev. Линейный уровень целостности задействуется пока редко, например, для изоляции на отрицательных линейных уровнях с помощью средств контейнерной виртуализации потенциально «опасного» ПО [10].

Рассмотренные свойства механизма МКЦ в ОС Astra Linux целесообразно использовать в способе выявления нарушений ее безопасности (выявления вирусов-шифровальщиков) с применением МО.

Во-первых, как уже отмечалось, изоляция потенциально «опасного» ПО, которое в первую очередь может быть подвержено атакам вирусов-шифровальщиков, либо непосредственно, либо в контейнерах-«песочницах» на отрицательных линейных (промежуточных неиерархических) уровнях целостности позволяет добавить этот линейный уровень

целостности в признаковые пространства для моделей МО, что должно повысить их эффективность при выявлении вирусов-шифровальщиков.

Во-вторых, на текущий момент авторам не известно вирусов-шифровальщиков, адаптированных к МКЦ в ОС Astra Linux, чем целесообразно воспользоваться. Отказы МКЦ зараженному вирусом-шифровальщиком процессу, функционирующему на отрицательном линейном уровне целостности, при попытке доступа на запись к содержимому домашнего каталога его учетной записи пользователя, обладающему нулевым линейным уровнем целостности (чего вирус-шифровальщик «не ожидает»), также могут быть использованы МО. Кроме того, наличие на некоторых файлах, например, в каталогах «/etc» или «/dev», ограничивающего доступ на чтение к ним специального флага SSI (и, как следствие, отказ в доступе к этим файлам), тоже может оказаться «неожиданным» для вируса-шифровальщика.

В-третьих, на выделенном промежуточном неиерархическом уровне целостности можно обеспечить запуск реализующих методы МО программных компонент. Это с одной стороны защитит их от несанкционированного изменения функциональности со стороны нарушителя, даже получившего в ОС полномочия суперпользователя root, когда его процессы обладают меньшим или несравнимым уровнем целостности. С другой стороны, возможные уязвимости самих реализующих методы МО программных компонент в случае их эксплуатации нарушителем не позволят ему захватить управление над всей ОС, для чего нужен максимальный для нее уровень целостности max_ilev. Кроме того, с этим промежуточным неиерархическим уровнем целостности можно задать специальную учетную запись пользователя администратора антивирусной защиты (или в целом аудита). Это позволит для такого администратора выделить функции и необходимое для их выполнения ПО, отделив их от функций и ПО администратора безопасности ОС (учетная запись которого должна обладать максимальным уровнем целостности max_ilev), снизив на него «нагрузку».

3. Формирование обучающего набора данных

В рамках данного исследования рассматривались несколько видов вирусов-шифровальщиков с открытым исходным кодом: GonnaCry, Ransomware-PoC, Itssoeasy, Python-Ransomware [34-37]. С помощью основанной на технологии eBPF системы сбора параметров системных вызовов в ОС Astra Linux [30], во-первых, было проанализировано поведение каждого из них в сценариях успешного получения вирусом доступа к атакуемым файлам, а также в сценариях отказов в доступах, нарушающих правила МКЦ. Во-вторых, для последующего обучения моделей МО были собраны данные о работе вышеуказанных вирусов-шифровальщиков при запуске непосредственно в ОС или в контейнерной среде (на примере Docker [38]) вместе с фоновой активностью штатного ПО. При этом аномалиями, поиск которых является целью разработки способа, предлагается считать генерируемые шифровальщиками интенсивные попытки получения доступов на запись (модификацию) файлов (каталогов) или на их чтение, когда они защищены флагом SSI, что не типично для штатного ПО ОС.

3.1 Сбор данных

Сбор данных для обучения моделей МО осуществлялся авторами самостоятельно, что обусловлено отсутствием в открытом доступе релевантных наборов данных, учитывающих специфику механизма МКЦ в ОС Astra Linux. Данный процесс выполнялся с использованием вышеуказанной системы сбора параметров системных вызовов и был разделен на этапы записи наборов данных для контейнерной среды и для запуска вирусов-шифровальщиков непосредственно в среде ОС Astra Linux. В рамках первого этапа внутри изолированных контейнеров, функционирующих на промежуточных неиерархических или на отрицательных линейных уровнях целостности, последовательно инициировались штатные файловые операции (архивирование, разархивирование, копирование, перемещение), а затем

осуществлялся запуск вирусов, настроенных на шифрование каталогов с файлами различных неиерархических уровней целостности, опционально содержащих флаг SSI, заранее смонтированных из файловой системы основной ОС. Такая изоляция ПО в контейнере позволила отсечь фоновый шум, генерируемый системным ПО, и зафиксировать специфические шаблоны поведения каждого отдельного образца вируса-шифровальщика.

Для повышения обобщающей способности моделей МО был проведен сбор данных в глобальном режиме мониторинга всей ОС Astra Linux, включающий параллельную фиксацию штатной фоновой активности («шума») ее ПО и деструктивных действий вирусов-шифровальщиков. В качестве такого «шума» выступали работа прикладного ПО (RuPost, Mattermost, Chrome, Firefox, VSCode, Git, QBitTorrent), часть которого функционировала на промежуточных уровнях целостности, а также выполнение типовых файловых операций. Вирусы-шифровальщики здесь запускались на уровне целостности «низкий» с попытками модификации файлов, имеющих разные неиерархические уровни целостности, и как следствие были зафиксированы два типа результатов: успешное шифрование при совпадении уровней целостности вируса-шифровальщика и файлов (каталогов), или отказы в доступе при попытках записи в файлы (каталоги), обладающие промежуточными уровнями целостности, что является важным признаком для обнаружения попыток обхода механизма МКЦ.

Собранные наборы данных позволили детально проанализировать и сравнить алгоритмы работы каждого из рассматриваемых образцов вируса-шифровальщика с открытым исходным кодом. Несмотря на общую конечную цель (уничтожение или недоступность пользовательских данных), механизмы взаимодействия вирусов-шифровальщиков с файловой системой ОС и МКЦ различаются, что напрямую отражается в генерируемых системных вызовах и их аргументах. Пример зафиксированных шаблонов представлен в табл. 1.

Табл. 1. Шаблоны системных вызовов исследуемых образцов вирусов-шифровальщиков.
Table 1. System call patterns of the studied ransomware samples.

Название вируса	Последовательность системных вызовов	Этап отказа в доступе механизмом МКЦ
GonnaCry	OPEN (RDONLY), UNLINK, OPEN (WRONLY) (создание файла с расширением .GNNCRY)	Отказ при попытке удаления защищенных МКЦ файлов (каталогов)
Python-Ransomware	OPEN (RDONLY), OPEN (WRONLY)	Отказ при попытке записи защищенных МКЦ файлов (каталогов)
Ransomware-PoC	OPENAT (RDWR), RENAME (переименование файла в .wasted)	Отказ при попытке переименования защищенных МКЦ файлов (каталогов)
Itssoeasy	OPENAT (RDWR), UNLINKAT, OPEN (WRONLY) (создание файла с расширением .itssoeasy)	Отказ при попытке удаления защищенных МКЦ файлов (каталогов)

3.2 Формирование признакового пространства

Для эффективного обнаружения вредоносной активности вирусов-шифровальщиков с помощью моделей МО для разработки рассматриваемого в статье способа было необходимо сформировать репрезентативное признаковое пространство. Учитывая, что действие вирусов-шифровальщиков в ОС заключается в интенсивной модификации файлов (каталогов), было принято решение ограничить область мониторинга системными вызовами (табл. 2), связанными с рассмотренными выше операциями в файловой системе. Такой подход позволяет значительно снизить уровень «шума» от штатного ПО.

Табл. 2. Анализируемые системные вызовы.
Table 2. Analyzed system calls.

Группа операций	Системные вызовы	Функциональное назначение
Управление	OPEN, OPENAT, MKDIR, MKDIRAT, SYMLINK, SYMLINKAT, LINK	Создание и инициализация доступа к файлам (каталогам)
Модификация	UNLINK, UNLINKAT, RMDIR, RENAME, CHMOD, CHOWN	Удаление, переименование, изменение прав и атрибутов файлов (каталогов)
Получение параметров и исполнение	NEWFSTATAT, ACCESS, EXECVE	Получение метаданных файлов (каталогов) и запуск новых процессов

Исходя из природы деструктивного воздействия вирусов-шифровальщиков, в итоговой выборке были оставлены только операции изменения (запись, удаление, переименование) файлов (каталогов), а также операции чтения файлов (каталогов), защищенных флагом SSI, так как именно эти действия представляют основную угрозу безопасности.

При взаимодействии процессов ОС с механизмом МКЦ ключевое значение имеет контекст выполнения системных вызовов: кто именно инициировал запрос, к какому файлу (каталогу) он был направлен, и каков был результат запроса. Поэтому, помимо самих типов системных вызовов, в признаковое пространство необходимо включить их аргументы и параметры, специфичные для СЗИ ОС Astra Linux.

Представленный выше набор системных вызовов с их параметрами позволяет сформировать признаковое пространство, приведенное в табл. 3. Выбранные признаки обеспечивают всесторонний охват взаимодействия процесса с файловой системой, предоставляя модели МО необходимый объем данных для выявления рассматриваемых аномалий.

Табл. 3. Признаковое пространство.
Table 3. Feature space.

Признак	Семантическая роль признака	Пример значения
syscall_name	Тип системного вызова	open, unlink, rename
task_pid	Идентификатор процесса	14547
task_name	Название процесса (не используется при обучении моделей МО)	gonnacry, chromium
task_ilev	Неиерархический уровень целостности процесса	0 (низкий), 32 (промежуточный)
task_ilinear	Линейный уровень целостности процесса	-128
file_ilev	Неиерархические категории целостности файла (каталога)	63 (высокий)
file_ilinear	Линейный уровень целостности файла (каталога)	0
ret	Результат выполнения операции (ключевой индикатор блокировок механизмом МКЦ)	0 (успех), -13 (отказ)
flags	Флаги режима доступа процесса к файлу (каталогу)	O_WRONLY | O_CREAT
file_mode	Дискреционные права доступа к файлу (каталогу)	644
pathname	Путь к файлу (каталогу) в файловой системе	/home/user/docs/file.pdf
file_type	Специальные битовые флаги МКЦ файла (каталога)	SSI, IINH

4. Выбор, настройка, обучение модели

В рамках рассматриваемого в статье способа решение задачи обнаружения деструктивного поведения вирусов-шифровальщиков в ОС Astra Linux было сведено к бинарной классификации временных последовательностей системных вызовов. Учитывая специфику вирусов-шифровальщиков, проявляющуюся в интенсивных и контекстно-зависимых цепочках обращений к файловой системе, в качестве подходов к классификации аномалий были протестированы модели градиентного бустинга над решающими деревьями (LightGBM) [39] и двунаправленные рекуррентные нейронные сети (BiGRU) [40]. Выбор данных средств объясняется их высокой эффективностью в задачах анализа табличных данных и

моделирования сложных временных зависимостей, что является критически важным для выявления деструктивных шаблонов на фоне активности штатного ПО.

Учитывая специфику данной задачи обнаружения аномального поведения, где класс «атака» существенно менее представлен, чем класс «норма», качество ответов модели оценивалось с использованием следующих метрик:

- точность (precision, насколько можно доверять классификатору);
- полнота (recall, как много объектов класса «атака» определяет классификатор);
- F1-мера (F1-measure, гармоническое среднее между точностью и полнотой);
- F2-мера (F2-measure, модифицированная метрика, отдающая больший приоритет полноте по сравнению с точностью, что критически важно в задачах обнаружения аномалий);
- площадь под кривой «точность-полнота» (PR-AUC, оценка качества ранжирования объектов по вероятности принадлежности к целевому классу).

Выбор данных метрик обусловлен необходимостью поиска баланса между полнотой обнаружения и точностью классификации (минимизация ложноположительных срабатываний критически важна для обеспечения доверия к системам мониторинга). Использование метрики accuracy, склонной к искажению результатов в задачах с дисбалансом классов, было признано нецелесообразным. Для комплексной оценки классификатора выбор PR-AUC предпочтителен перед популярной метрикой ROC-AUC, так как последняя является оптимистичной при сильном дисбалансе и не учитывает точность предсказаний на редком классе. Анализ итоговой эффективности моделей, который включает выбор оптимального порога классификации относительно фиксации доли ложноположительных срабатываний и значений вышеперечисленных метрик, представлен в разд. 5 при оценке эффективности способа.

4.1 Выбор модели

Первоначально для решения данной задачи был протестирован один из алгоритмов градиентного бустинга – LightGBM [39]. Реализация данного подхода потребовала «ручного» конструирования признакового пространства, включающего как характеристики отдельных событий (уровни целостности процессов, файлов, каталогов, результаты доступа), так и агрегированные статистические показатели (частота операций, плотность и количество файловых кластеров процесса в скользящем окне). Несмотря на высокие показатели метрик на тестовой выборке (F1-score = 0.98 и Recall = 0.96), интегрирование данной модели в систему мониторинга событий информационной безопасности в ОС Astra Linux, позволило обнаружить критическое количество ложноположительных срабатываний. Проведенный анализ показал, что модель ограничена в использовании сложных семантических признаков (например, таких как пути к файлам), требует трудоемкой предварительной генерации признаков и не обладает достаточной гранулярностью для выявления сложных временных закономерностей в цепочках системных вызовов.

Вследствие выявленных ограничений было принято решение об отказе от использования LightGBM в пользу архитектуры двунаправленной рекуррентной нейронной сети. В отличие от градиентного бустинга архитектура BiGRU позволила исключить этап ручного проектирования признаков благодаря разработке специальных обучаемых блоков для кодирования разнородных типов признаков, включая категориальные и текстовые. Двунаправленный характер нейронной сети обеспечивает учет контекста выполнения системного вызова во всей цепочке событий, что повышает устойчивость модели к зашумленным данным, характерным для работы вирусов-шифровальщиков непосредственно (вне контейнеров) в среде ОС Astra Linux, и существенно снижает вероятность ложноположительных срабатываний при классификации их деструктивных действий.

При этом альтернативные подходы на основе n-грамм в сочетании с простыми линейными моделями (например, логистической регрессией) были отвергнуты, поскольку, как и в случае с градиентным бустингом, они не способны эффективно кодировать сложную семантическую информацию параметров системных вызовов без значительного роста размерности признакового пространства. Применение сверточных нейронных сетей (CNN) с позиционным кодированием рассматривается авторами как перспективное направление для будущих исследований с целью оптимизации быстродействия инференса. В то же время, при решении задач на последовательностях, где критически важен строгий хронологический порядок и непрерывный контекст, сверточные архитектуры в общем случае уступают рекуррентным по качеству классификации из-за ограничений локального рецептивного поля сверточных слоев. Рекуррентные модели изначально ориентированы на последовательную обработку данных и эффективное отслеживание долгосрочных временных зависимостей, что и предопределило их выбор для текущего исследования.

4.2 Синтез классификатора аномального поведения

4.2.1 Подготовка данных и сэмплирование. Для формализации задачи классификации деструктивного поведения вирусов-шифровальщиков было принято решение перейти от анализа отдельных системных событий к классификации окон, содержащих последовательности событий, что позволяет учесть контекст взаимодействия процесса с файловой системой. В проведенных экспериментах длина контекстного окна была установлена равной 30 событиям, а критерием для определения окна как аномального (метка «1») являлось наличие в нем не менее 30% событий, относящихся к активности вируса-шифровальщика; в противном случае окно помечалось как «норма» (метка «0»). Выбор данных значений был обусловлен необходимостью первичной проверки предложенной стратегии классификации (окно такой длины позволяет захватить достаточный фрагмент работы вируса-шифровальщика, поскольку его деструктивная активность проявляется именно в устойчивых группах событий), при этом сохраняя высокую вычислительную эффективность модели на этапе проверки данной гипотезы. Исходный объем собранного набора данных составил 3 069 800 уникальных записей системных вызовов с их параметрами. Подготовка событий для классификации осуществлялась с помощью алгоритма скользящего окна (с размером окна в 30 событий и шагом сдвига, равным 5), а для обеспечения статистической достоверности результатов данные, собранные в режиме сценариев, были распределены по обучающей, валидационной и тестовой выборкам в соотношении 60% / 20% / 20% методом стратификации с учетом принадлежности окон к конкретным сценариям.

Поскольку изначальное соотношение количества экземпляров классов «норма» и «аномалия» в обучающей выборке составляло 70% / 30%, а характер поведения исследуемых образцов вирусов-шифровальщиков отличались, для предотвращения доминирования более частотных примеров в обучающей выборке была применена процедура балансировки классов. На этапе подготовки данных редким типам атак присваивались повышенные веса, что позволяло модели уделять большее внимание менее представленным сценариям. Однако, во избежание переобучения, связанного с многократным повторением одних и тех же редких примеров, для наиболее малочисленных групп событий была реализована процедура аугментации данных (метод искусственного расширения обучающей выборки путем генерации синтетических примеров). Она включала создание модифицированных копий существующих событий путем подмены системных вызовов и их аргументов в часто встречающихся последовательностях на логически эквивалентные шаблоны редких вирусов-шифровальщиков, а также варьирование уровней целостности процессов, файлов (каталогов) с соответствующим изменением результатов доступа, что позволило повысить устойчивость и обобщающую способность модели.

4.2.2 Конструирование архитектуры модели. Для эффективной классификации деструктивного поведения вирусов-шифровальщиков была спроектирована нейронная сеть,

учитывающая разнородную природу входных признаков, которые содержатся в записи системного вызова. Поскольку каждый тип признаков несет уникальную семантическую информацию, было принято решение реализовать для каждого из них специализированный блок кодирования, предоставляя возможность нейронной сети максимизировать извлечение полезной информации для решения задачи классификации. Схема кодирования признаков представлена на рис. 1, а архитектура нейросетевой модели – на рис. 2. При проектировании нейронной сети использовались следующие слои: Embedding (преобразование категориальных данных в плотные векторы), Identity (единичная матрица), GRU/BiGRU (однонаправленный/двунаправленный рекуррентный слой), LayerNorm (нормализация по оси размерности признаков), ReLU (нелинейная функция активации), Dropout (операция дропаута) [41], GlobalMaxPool/GlobalAveragePool (операции глобальной агрегации признаков путем выбора максимального и среднего значений по временной оси), Linear (полносвязный слой) и Sigmoid (сигмоидальная функция активации).

Рассмотрим каждый из представленных на рис. 1 признаков:

- **Тип системного вызова** – для его кодирования используется обучаемый слой Embedding, который переводит каждый тип системного вызова в плотный вектор фиксированной размерности. Это позволяет сети самостоятельно выучить векторные представления для каждого анализируемого типа;
- **Путь к файловой системе (pathname)** – является наиболее информативным, так как он сообщает об области доступа в файловой системе, так и о типе целевых файлов (каталогов). Предварительно был обучен токенизатор на основе Byte-level BPE [42], что позволило эффективно обрабатывать пути на уровне байтовых последовательностей, исключая риск появления неизвестных токенов при наличии в путях спецсимволов или нестандартных кодировок. Учитывая, что 99% путей в обучающей выборке покрываются 64 токенами, входная последовательность была ограничена 62 токенами начала пути (уровни вложенности) и 2 токенами конца пути (имя файла и расширение). Для токенов обучаются индивидуальные эмбеддинги (с помощью своего Embedding слоя), которые затем обрабатываются однонаправленным слоем GRU, фиксирующим иерархический порядок их вложенности. Полученные скрытые состояния проходят через слой AdaptiveMaxPool1d, который агрегирует информацию в 4 вектора (покомпонентный максимум для четырех сегментов пути). Эти векторы проецируются полносвязным слоем в пространство меньшей размерности для балансировки веса признака в итоговой конкатенации и нормализируются;
- **Флаги режима доступа (flags)** – представляют собой комбинацию конечного набора категориальных значений. Поэтому было решено представить их в виде бинарного вектора, где каждая компонента указывает на наличие конкретного флага в аргументах вызова. Для балансировки влияния данного признака относительно других размерность бинарного вектора уменьшена посредством полносвязного слоя, за которым следует слой нормализации;
- **Численные признаки** – к ним относятся идентификатор процесса (task_pid), уровни целостности (task_ilev, task_ilinear, file_ilev, file_ilinear) процесса и файла (каталога), специальные битовые флаги (file_type) и результат доступа (ret). Поскольку эти признаки имеют различные диапазоны значений, их численный кодировщик содержит только процедуру нормализации, что позволяет нивелировать разрыв в порядках значений и выровнять вклад каждого признака;
- **Права доступа (file_mode)** – представлены в виде бинарного вектора, описывающего дискреционные права доступа. Данный признак подается в сеть без дополнительных трансформаций, так как его структура уже оптимизирована для интерпретации моделью.

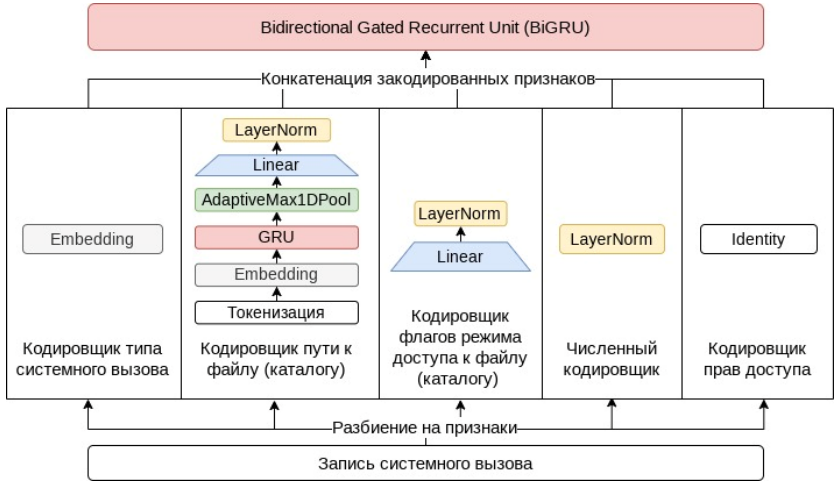

Рис. 1. Схема кодирования разнородных признаков записи системного вызова.
Fig. 1. Encoding scheme for heterogeneous system call record features.

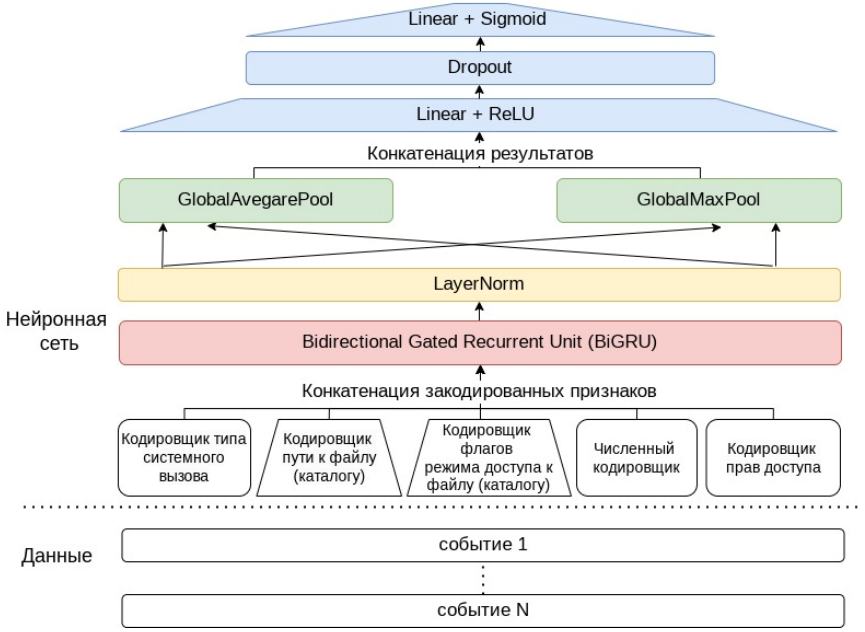

Рис. 2. Архитектура классификатора аномалий.
Fig. 2. Architecture of the anomaly classifier.

В соответствии со схемой на рис. 1 после прохождения всех кодировщиков полученные векторы объединяются операцией конкатенации в единый вектор признаков, формирующий семантическое представление конкретной записи системного вызова. Затем согласно представленной на рис. 2 архитектуре классификатора аномалий последовательность закодированных событий подается на вход двунаправленному рекуррентному слою BiGRU для

выявления временных зависимостей как от предыдущих, так и от последующих событий в окне.

Для извлечения наиболее значимой информации из скрытых состояний нейронной сети применяется стратегия глобальной агрегации: скрытые состояния рекуррентного слоя проходят через параллельные блоки взятия покомпонентного среднего и максимума. Такой подход позволяет одновременно учитывать усредненные характеристики последовательности и выделять наиболее выраженные аномалии, что формирует разносторонний набор признаков для классификации. Для повышения обобщающей способности и предотвращения переобучения в архитектуру интегрирован слой Dropout, который в процессе обучения случайным образом отключает заданную долю нейронов, способствуя более равномерному обучению последнего слоя. На завершающем этапе данные проходят через полносвязные слои с нелинейной функцией активации ReLU, после чего сигмоидная функция активации формирует итоговую оценку вероятности принадлежности анализируемого окна событий к аномальному классу.

4.2.3 Стратегия обучения модели. Процесс обучения модели классификации был направлен на минимизацию ошибок при работе с несбалансированными данными и предотвращение переобучения. В качестве функции потерь была выбрана Focal Loss [43], представляющая собой модификацию стандартной бинарной перекрёстной энтропии. Данная функция позволяет адаптировать веса функции потерь в зависимости от сложности классификации примера, что критически важно в условиях дисбаланса классов. Для настройки баланса между точностью и полнотой обнаружения были установлены гиперпараметры $\alpha = 0.5$ и $\gamma = 4.0$. Первый из них обеспечивает выравнивание весов классов, а второй позволяет снизить вклад низкоинформативных примеров «шума» штатного ПО, сосредоточив внимание модели на наиболее сложных для классификации случаях (деструктивной активности вирусов-шифровальщиков). Такое сочетание гиперпараметров обеспечивает более точную настройку весов классификатора, что позволяет эффективно выделять редкие, но критически важные признаки вредоносного поведения на фоне «шума».

Для обучения нейронной сети использовался оптимизатор AdamW [44], который является модификацией метода Adam [45], дополненной механизмом уменьшения весов модели, интегрированным непосредственно в процедуру их обновления. Значение параметра регуляризации было установлено на уровне 0.0005, что позволяет эффективно штрафовать модель за наличие весов с чрезмерно высокими значениями, снижая тем самым риск переобучения и обеспечивая лучшую обобщающую способность.

Для обеспечения эффективной сходимости процесса обучения применялась циклическая стратегия регулирования шага обучения OneCycleLR [46]. Данный метод реализует адаптивную динамику изменения шага обучения (learning rate): в течение нескольких первых эпох выполняется фаза линейного возрастания скорости обучения до заданного пикового значения (0.001), после чего осуществляется ее постепенное снижение по косинусному закону. Такой подход позволяет модели быстрее находить оптимальные области в пространстве весов и минимизировать вероятность застревания в локальных минимумах. Общее количество эпох обучения было ограничено десятью, что, при использовании циклической стратегии, обеспечивает достижение необходимого качества классификации при высокой вычислительной эффективности.

5. Экспериментальная оценка эффективности разработанного способа

5.1 Оценка модели на тестовых данных

Оценка качества работы классификатора, используемого в изложенном в статье способе и реализующей его системе мониторинга, была проведена на выделенной тестовой выборке. В

ходе эксперимента модель оценивалась при различных порогах принятия решения, что позволило проанализировать зависимость точности классификации от заданного допустимого уровня ложноположительных срабатываний (FPR). Вариативность порогов дает возможность гибко настраивать систему мониторинга, адаптируя ее к требованиям безопасности: от режима максимальной полноты обнаружения до режима с предельно низким уровнем ложных уведомлений. Выбор оптимального порога позволяет достичь необходимого компромисса между чувствительностью системы и её специфичностью, что является критически важным для минимизации нагрузки на администратора безопасности (аудита или антивирусной защиты) при сохранении высокой полноты выявления угроз.

Результаты оценки классификации, полученные для фиксированных уровней FPR, представлены в табл. 4.

Табл. 4. Результаты оценки качества классификатора.
Table 4. Classifier performance evaluation results.

Уровень FPR, %	Порог классификации	Precision	Recall	F1-score	F2-score	PR-AUC
1.00%	0.007724	0.983076	0.999999	0.991466	0.996569	0.999999
0.10%	0.309388	0.999199	0.999964	0.999581	0.999811	0.999999
0.01%	0.516108	**0.999987**	**0.999840**	**0.999985**	**0.999847**	0.999999

Из табл. 4 видно, что классификатор демонстрирует высокую стабильность метрик даже при ужесточении ограничений на уровень ложных срабатываний. Наилучшие показатели эффективности были зафиксированы при пороге срабатывания модели 0.516108. Данная конфигурация позволила достичь уровня FPR 0.01% при сохранении показателя полноты обнаружения на уровне 99.98%. Столь высокие значения Precision и Recall свидетельствуют о способности нейронной сети эффективно различать деструктивную активность вирусов-шифровальщиков и штатную работу прикладного ПО в условиях работы механизма МКЦ ОС Astra Linux. Высокий показатель PR-AUC (0.999999) подтверждает хорошее качество ранжирования вероятностей принадлежности событий к классу аномалий во всем диапазоне порогов классификации.

Помимо усредненных показателей, для оптимального порога классификации была проведена детализированная оценка качества обнаружения для каждого типа вирусов-шифровальщиков, представленных в обучающей выборке. Результаты тестирования приведены в табл. 5.

Табл. 5. Результаты обнаружения шифровальщиков, представленных в обучающей выборке.
Table 5. Detection results for known ransomware types from the training set.

Шифровальщик	Precision	Recall	F1-score	F2-score	PR-AUC	Количество примеров
Ransomware-PoC	0.999	0.9998	0.999	0.999	0.999	48707
Python-Ransomware	0.999	0.9998	0.999	0.9998	0.999	10419
Itssoeasy	0.9998	0.9998	0.999	0.9998	0.999	25906
GonnaCry	0.9998	0.9998	0.999	0.9998	0.999	21487

Для оценки обобщающей способности предложенного классификатора, а также для проведения анализа вклада специфичных для СЗИ ОС Astra Linux параметров системных вызовов (признаков МКЦ), авторами был проведен дополнительный эксперимент, в рамках которого осуществлялось тестирование обученной модели на обнаружение новых типов вирусов-шифровальщиков, данные об активности которых ранее не были представлены в обучающей выборке. Это позволило оценить способность адаптации модели классификации к новым типам угроз.

Для выявления практической значимости признаков МКЦ тестирование проводилось в двух изолированных сценариях: в первом случае механизм МКЦ ограничивал шифровальщикам доступ к атакуемым файлам (атака сопровождалась каскадами отказов в доступе); во втором – доступ к объектам файловой системы не ограничивался, что позволяло шифровальщикам беспрепятственно осуществлять их модификацию.

Табл. 6. Оценка качества обнаружения новых шифровальщиков и анализ вклада признаков МКЦ.
Table 6. Detection quality of unseen ransomware and MIC features contribution analysis.

Сценарий активности	Шифровальщик	Precision	Recall	F1-score	F2-score	PR-AUC
Отказы в доступах (нарушение правил МКЦ)	Oracle-Ransomware	0.999	0.999	0.999	0.999	0.999
	Linux-malware-2	0.995	0.995	0.995	0.995	0.999
	Alexmr-Ransomware	0.995	0.995	0.995	0.995	0.999
Успешный доступ (отсутствие блокировок МКЦ)	Oracle-Ransomware	0.990	0.999	0.995	0.998	0.999
	Linux-malware-2	0.997	**0.787**	**0.880**	**0.822**	0.986
	Alexmr-Ransomware	0.999	**0.812**	**0.896**	**0.844**	0.999

Из анализа полученных результатов можно сделать вывод о способности модели к адаптации к новым типам угроз и о роли специфичных параметров системных вызовов (признаков МКЦ) при классификации. Во-первых, модель демонстрирует высокую обобщающую способность при переносе на новые типы угроз. Даже в сценариях успешного доступа, когда вирусы не нарушали правила МКЦ, точность классификации для всех новых образцов осталась на высоком уровне (от 0.990 до 0.999). Более того, шаблоны поведения шифровальщика Oracle-Ransomware оказались настолько выраженными, что модель выявила его с полнотой 99.9% исключительно на основе базовых файловых операций, без опоры на результаты доступа, контролируемые механизмом МКЦ.

Во-вторых, результаты проведенного абляционного анализа наглядно подтверждают критическую значимость включения параметров МКЦ в признаковое пространство классификатора. В сценариях успешного доступа полнота обнаружения вирусов-шифровальщиков Linux-malware-2 и Alexmr-Ransomware снизилась до 78.7% и 81.2% соответственно. Это обусловлено спецификой работы данных вредоносных программ, которая существенно отличалась от образцов из обучающей выборки: новые угрозы порождали большое количество кратковременных дочерних процессов (например, вызов утилиты openssl для обработки отдельных файлов) вместо выполнения всех операций в рамках пары дочерних процессов. Подобное, не рассмотренное ранее дробление активности затруднило выявление признаков вредоносной активности моделью. Однако в условиях работы механизма МКЦ (сценарий отказов в доступах) показатели эффективности возросли: полнота обнаружения для обоих скрытных вирусов увеличилась до 99.5%. Это доказывает, что каскады событий отказов в доступе, обогащенные контекстом параметров МКЦ, создают интенсивный сигнал аномальности, позволяя модели с высокой точностью классифицировать новые и сложные модификации вирусов-шифровальщиков, компенсируя нехватку знаний о конкретных способах работы ВПО.

5.2 Архитектура системы мониторинга

Для практической апробации предлагаемого способа выявления вирусов-шифровальщиков, была разработана система мониторинга событий информационной безопасности, архитектура которой представлена на рис. 3 (где УЦ – уровень целостности).

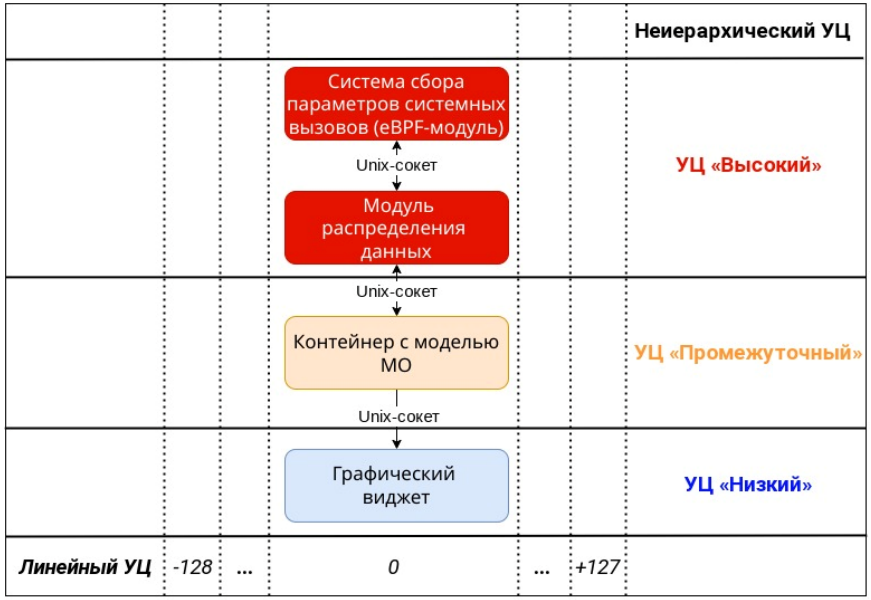

Рис. 3. Архитектура разработанной системы мониторинга.
Fig. 3. Architecture of the developed monitoring system.

Архитектура системы включает следующие функциональные модули:

- **Система сбора параметров системных вызовов** – является доверенным источником данных, функционирующим на максимальном неирархическом уровне целостности ОС (max_ilev);
- **Модуль распределения данных** (реализован на языке программирования Go) – отвечает за гибкую фильтрацию, маршрутизацию потоков событий и масштабируемость системы, позволяя динамически подключать дополнительные аналитические модули;
- **Контейнер с моделью МО** – изолированная среда, в которой выполняется выявление деструктивной активности;
- **Графический виджет** – пользовательский интерфейс для отображения оповещений об обнаруженных инцидентах (выявлении вирусов-шифровальщиков).

Обмен данными между компонентами системы осуществляется через Unix-сокеты, что обеспечивает высокую скорость передачи событий при сохранении изоляции между процессами.

Важной особенностью предложенной архитектуры является разделение функциональных модулей по неиерархическим уровням целостности, что повышает общую устойчивость системы к компрометации. Система сбора данных и модуль распределения (как критически важные доверенные компоненты) функционируют с привилегиями администратора безопасности ОС на уровне max_ilev. Программные компоненты, реализующие методы МО, изолированы на промежуточном уровне целостности. Данное решение продиктовано двумя факторами. Во-первых, это защищает компонент с моделью МО от несанкционированной модификации со стороны нарушителя, даже в случае получения им прав суперпользователя

root (при условии обладания меньшим или несравнимым уровнем целостности). Во-вторых, потенциальные уязвимости в коде библиотек, необходимых для работы методов МО, не позволят нарушителю захватить управление всей ОС. Такой подход позволяет выделить функции администратора антивирусной защиты (аудита), отделив их от функционала администратора безопасности ОС, что снижает нагрузку на последнего и обеспечивает изоляцию процессов системы мониторинга. Графический виджет, в свою очередь, функционирует на уровне сессии пользователя (по умолчанию уровень «низкий»), обеспечивая доступ к результатам мониторинга без предоставления прав на управление функциональными компонентами системы мониторинга.

5.3. Тестирование системы обнаружения в режиме реального времени

Для оценки эффективности рассмотренного способа и реализующей его системы мониторинга был проведен ряд экспериментов по запуску на низком неиерархическом уровне целостности вирусов-шифровальщиков на каталоге, содержащем 1000 текстовых файлов со средним размером файла 2.75 КБ, включавший сценарии успешного шифрования низкоцелостных файлов внутри данного каталога и сценарии с попытками доступа к защищенным высокоцелостным файлам (каталогам).

Все тесты проводились на виртуальной машине ОС Astra Linux в режиме защищенности «Воронеж» с выделенными 12 ГБ ОЗУ и 8 ядрами центрального процессора. Время реакции системы определялось как интервал между регистрацией первого деструктивного события и моментом отправки оповещения в графический виджет. Результаты тестирования показали высокую оперативность: в сценарии успешного шифрования система зафиксировала активность за 1,96 секунды после компрометации 16 файлов, тогда как при попытках записи в высокоцелостные файлы реакция системы составила всего 0,70 секунды. Более высокая скорость и точность обнаружения во втором сценарии объясняются использованием моделью МО специфических параметров СЗИ ОС Astra Linux, фиксирующих явные попытки процесса с низким уровнем целостности воздействовать на защищенные файлы (каталоги). В подобных случаях механизм МКЦ генерирует каскад событий отказов, которые, в сочетании с аргументами системных вызовов, создают для классификатора более сильный сигнал аномального поведения, чем при обычной записи в доступные процессу файлы.

В ходе тестирования были также выявлены единичные ложноположительные срабатывания на системное ПО (journalctl, systemctl) и браузер Firefox. Анализ показал, что данные инциденты вызваны ограниченным представлением указанного ПО в обучающей выборке, сформированной ручным методом. Дальнейшее развитие системы предполагает расширение обучающего набора за счет сбора новых сценариев и аугментации данных штатного ПО, что позволит минимизировать количество ложных срабатываний без потери полноты обнаружения аномалий.

6. Заключение

В рамках данной работы предложен способ выявления нарушений безопасности (на примере вирусов-шифровальщиков) в ОС Astra Linux, базирующийся на применении методов МО к анализу потока событий системных вызовов. Основным практическим результатом исследования стала разработанная система мониторинга, включающая синтезированный классификатор деструктивного поведения, архитектура которой спроектирована с учетом специфики функционирования механизма МКЦ. Проведенные эксперименты подтвердили, что интеграция специфических параметров МКЦ (таких как неиерархические и линейные уровни целостности процессов, файлов, каталогов, а также их специальных флагов) в признаковое пространство позволяет нейронной сети формировать контрастные и интерпретируемые сигналы аномального поведения, существенно повышая полноту обнаружения деструктивной активности.

Разработанный способ показал высокую эффективность, обеспечивая оперативное выявление попыток модификации файлов, включая сценарии с каскадными отказами в доступе, характерными для попыток нарушения МКЦ ОС Astra Linux. Синтезированная архитектура классификатора, основанная на двунаправленном рекуррентном слое (BiGRU) с механизмом специализированного кодирования признаков, показала способность к эффективному анализу временных зависимостей. Высокое быстродействие системы мониторинга обеспечивает своевременное уведомление пользователя об активности вируса-шифровальщика на ранних стадиях процесса компрометации данных. Однако, несмотря на достигнутые показатели точности, система требует дальнейшей доработки в части расширения обучающих выборок для снижения частоты ложноположительных срабатываний, вызванных редкими сценариями штатной активности системного и прикладного ПО.

Перспективным направлением дальнейших исследований является масштабирование системы путем тестирования на расширенном спектре современных вирусов-шифровальщиков и сценариев их эксплуатации. В части совершенствования архитектуры классификатора планируется изучение возможности перехода к трансформерным архитектурам, обладающим высоким потенциалом в анализе длинных последовательностей событий [47], с учетом ограничений вычислительных ресурсов при функционировании системы мониторинга в среде отдельного экземпляра ОС. Кроме того, дальнейшее развитие предложенного способа предполагает применение передовых техник обучения, включая расширение спектра аугментаций данных, а также возможного использования методов контрастного обучения [48] и генеративно-состязательных подходов [49] для генерации синтетических сценариев атак, что позволит существенно повысить обобщающую способность и устойчивость модели к новым типам угроз.

Также следует отметить, что при участии авторов ведутся исследования по включению в систему мониторинга технологий МО для выявления скрытых каналов по времени [33] в контексте мандатного управления доступом, используемого в режиме защищенности «Максимальный» («Смоленск») ОС Astra Linux.

Список литературы / References

[1]. He Z., Davila D., Bi S., Wang T., Hou T. Machine Learning for Cybersecurity: A Survey of Applications, Adversarial Challenges, and Future Research Directions. Electronics, 2025, vol. 14, issue 23: 4563, 28 p.

[2]. Ozkan-Okay M., Akin E., Aslan Ö., Kosunalp S., Iliev T., Stoyanov I., Beloev I. A Comprehensive Survey: Evaluating the Efficiency of Artificial Intelligence and Machine Learning Techniques on Cyber Security Solutions. IEEE Access, 2024, vol. 12, pp. 12229-12256.

[3]. Pinto A., Herrera L.-C., Donoso Y., Gutierrez J. A. Survey on Intrusion Detection Systems Based on Machine Learning Techniques for the Protection of Critical Infrastructure. Sensors, 2023, vol. 23, issue 5: 2415, 18 p.

[4]. ML IDS: сетевая система обнаружения вторжений с применением машинного обучения. Доступно по ссылке: https://www.ispras.ru/technologies/ml_ids/, 09.09.2026. / ML IDS: Network Intrusion Detection System Using Machine Learning. Available at: https://www.ispras.ru/technologies/ml_ids/, accessed 09.09.2026. (in Russian).

[5]. Ferdous J., Islam R., Mahboubi A., Islam M. Z. A Survey on ML Techniques for Multi-Platform Malware Detection: Securing PC, Mobile Devices, IoT, and Cloud Environments. Sensors, 2025, vol. 25, issue 4: 1153, 38p.

[6]. Модуль поведенческого анализа Behavioral Anomaly Detection для SIEM-системы MaxPatrol. Доступно по ссылке: https://ptresearch.media/articles/modul-bad-kak-ml-algoritmy-pomogayut-otslezhivat-hakerov, 09.09.2026. / Behavioral Anomaly Detection module for MaxPatrol SIEM. Available at: https://ptresearch.media/articles/modul-bad-kak-ml-algoritmy-pomogayut-otslezhivat-hakerov, accessed 09.09.2026. (in Russian).

[7]. Обзор Ankey ASAP 2.4, средства расширенного анализа событий по безопасности. Доступно по ссылке: https://www.anti-malware.ru/reviews/Ankey-ASAP, 09.09.2026. / Ankey ASAP 2.4 Review: Advanced Security Event Analysis Tool. Available at: https://www.anti-malware.ru/reviews/Ankey-ASAP, accessed 09.09.2026. (in Russian).

[8]. Операционная система специального назначения Astra Linux Special Edition. Доступно по ссылке: https://astra.ru/software-services/os/, 09.09.2026. / Astra Linux Special Edition operating system. Available at: https://astra.ru/software-services/os/, accessed 09.09.2026. (in Russian).

[9]. Девянин П.Н., Тележников В.Ю., Третьяков С.В. Основы безопасности операционной системы Astra Linux Special Edition. Управление доступом. Учебное пособие. М., Горячая линия – Телеком, 2022, 148 с. / Devyanin P.N., Telezhnikov V.Y., Tret'yakov S.V. Astra Linux Special Edition security basics. Access control. Hotline-Telecom, 2022, 148 p. (in Russian).

[10]. Девянин П.Н., Старостин А.А., Панов Д.С., Усачев С.В. Проектирование и развитие механизма мандатного контроля целостности в операционной системе Astra Linux // Труды ИСП РАН, том 37, вып. 2, 2025, С. 61-78 / Devyanin P.N., Starostin A.A., Panov D.S., Usachev S.V. Design and Development of a Mandatory Integrity Control Mechanism in the Astra Linux Operating System. Trudy ISP RAN/Proc. ISP RAS, 2025, vol. 37, issue 2, pp. 61-78 (in Russian).

[11]. Девянин П.Н., Горбатов В.В., Трубников А.А. Подход к применению мандатного контроля целостности в ОС Astra Linux для управления доступом к пользовательским данным // Труды ИСП РАН, 2026, том 38, вып. 3, часть 1, стр. 71–86 / Devyanin P.N. Gorbatov V.V., Trubnikov A.A. Implementing Mandatory Integrity Control in Astra Linux OS for Access Control of User Data. Trudy ISP RAN/Proc. ISP RAS, 2026, vol. 38, issue 3, part 1, pp. 71-86 (in Russian).

[12]. Выписка из Требований по безопасности информации, утвержденных приказом ФСТЭК России от 2 июня 2020 г. N 76. Доступно по ссылке: https://fstec.ru/dokumenty/vse-dokumenty/spetsialnye-normativnye-dokumenty/trebovaniya-po-bezopasnosti-informatsii-utverzhdeny-prikazom-fstek-rossii-ot-2-iyunya-2020-g-n-76, 09.09.2026 / Excerpts from Requirements for information security approved by FSTEK Russia order #76 of 2nd June 2020. Available at: https://fstec.ru/dokumenty/vse-dokumenty/spetsialnye-normativnye-dokumenty/trebovaniya-po-bezopasnosti-informatsii-utverzhdeny-prikazom-fstek-rossii-ot-2-iyunya-2020-g-n-76, accessed 09.09.2026. (in Russian).

[13]. Аветисян А.И. Доверенный искусственный интеллект. Вестник Российской академии наук, 2024, том 94, № 3, с. 200-209 / Avetisyan A.I. Trusted artificial intelligence. Bulletin of the Russian Academy of Sciences, 2024, vol. 94, № 3, pp. 200-209 (in Russian).

[14]. Девянин П.Н., Тележников В.Ю., Хорошилов А.В. Формирование методологии разработки безопасного системного программного обеспечения на примере операционных систем. Труды ИСП РАН, 2021, том 33, вып. 5, С. 25-40 / Devyanin P.N., Telezhnikov V.Y., Khoroshilov A.V. Building a methodology for secure system software development on the example of operating systems. Trudy ISP RAN/Proc. ISP RAS, 2021, vol. 33, issue 5, pp. 25-40 (in Russian).

[15]. Sekar A., Kulkarni S.G., Kuri J. LeARN: Leveraging eBPF and AI for Ransomware Nose Out. 2025 17th International Conference on COMmunication Systems and NETworks (COMSNETS), 2025, pp. 1323-1328.

[16]. Kharraz A., Arshad S., Mulliner C., Robertson W., Kirda E. UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware. Proceedings of the 25th USENIX Security Symposium, 2016, pp. 757-772.

[17]. Scaife N., Carter H., Traynor P., Butler K.R.B. CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), 2016, pp. 303-312.

[18]. Wu Y.-C., Chang Y.-L. Ransomware Detection on Linux Using Machine Learning with Random Forest Algorithm. TechRxiv, 2024.

[19]. Brodzik A., Malec-Kruszyński T., Niewolski W., Tkaczyk M., Bocianiak K., Loui S.-Y. Ransomware Detection Using Machine Learning in the Linux Kernel. arXiv preprint, 2024. DOI: 10.48550/arXiv.2409.06452.

[20]. Higuchi K., Kobayashi R. Real-Time Defense System using eBPF for Machine Learning-Based Ransomware Detection Method. 2023 Eleventh International Symposium on Computing and Networking Workshops (CANDARW), 2023, pp. 213-219.

[21]. von der Assen J., Feng C., Celdrán A.H., Oleš R., Bovet G., Stiller B. GuardFS: A file system for integrated detection and mitigation of Linux-based ransomware. Journal of Information Security and Applications, 2025, vol. 93, 104078, 14 p.

[22]. Sánchez P.M.S., Celdrán A.H., Bovet G., Pérez M.G., Pérez G.M. Transfer Learning in Pre-Trained Large Language Models for Malware Detection Based on System Calls. arXiv preprint, 2024. DOI: 10.48550/arXiv.2405.09318.

[23]. Огнев И.А., Никрошкин И.В., Медведев М.А., Красников А.Д. Исследование встроенных средств защиты информации Unix-систем на примере заражения вредоносным программным

обеспечением. Доклады ТУСУР, 2023, т. 26, № 4, с. 29–34 / Ognev I.A., Nikroshkin I.V., Medvedev M.A., Krasnikov A.D. Study of built-in information protection tools of Unix systems on the example of malware infection. Reports TSUCSR, 2023, vol. 26, №. 4, pp. 29-34 (in Russian).

[24]. Kaspersky Endpoint Security 10 Service Pack 1 Maintenance Release 1 for Linux: Hardware and software requirements. Available at: https://support.kaspersky.com/kes4linux/10.1.1/en-US/180377.htm, accessed 09.09.2026.

[25]. Kaspersky Endpoint Security for Linux 11 for Linux: Hardware and software requirements. Available at: https://support.kaspersky.com/kes11linux/245117, accessed 09.09.2026.

[26]. Dr.Web Workstation for Linux: System Requirements and Compatibility. Available at: https://cdn-download.drweb.com/pub/drweb/unix/workstation/11.1/business/documentation/html/en/dw_8_sysrequirements.htm, accessed 09.09.2026.

[27]. Dr.Web Enterprise Security Suite: Anti-virus Network Quick Installation Guide. Available at: https://cdn-download.drweb.com/pub/drweb/esuite/13.0.1/documentation/drweb-13.0.1-esuite-quick-install-en.pdf, accessed 09.09.2026.

[28]. Калинкин А.О., Голуб С.А., Коркин И.Ю., Пятовский Д.Н. Обнаружение программ-шифровальщиков на основе данных механизма трассировки событий и применения метода машинного обучения // Безопасность информационных технологий, 2022, т. 29, № 3, с. 82-93 / Kalinkin A.O. et al. Ransomware detection based on machine learning models and Event Tracing for Windows. IT Security, 2022, vol. 29, №. 3, pp. 82–93 (in Russian).

[29]. Gregg B. BPF Performance Tools. Addison-Wesley Professional, 2019, 880 p.

[30]. Девянин П.Н., Жиляков С.С., Смирнов А.И. Тестирование подсистемы безопасности ОС Astra Linux на основе формализованного описания модели управления доступом. Труды ИСП РАН, 2025, том 37, вып. 6, часть 2, с. 21-36 / Devyanin P.N., Zhiliakov S.S., Smirnov A.I. Testing the Astra Linux OS security subsystem based on a formalized description of the access control model. Trudy ISP RAN/Proc. ISP RAS, 2025, vol. 37, issue 6, part 2, pp. 21-36 (in Russian).

[31]. Liu M., Cen L., Ruta D. Gradient Boosting Models for Cybersecurity Threat Detection with Aggregated Time Series Features. 2023 18th Conference on Computer Science and Intelligence Systems (FedCSIS), 2023, pp. 1311-1315.

[32]. Landauer M., Onder S., Skopik F., Wurzenberger M. Deep learning for anomaly detection in log data: A survey. Machine Learning with Applications, 2023, vol. 12, 100470, 19p.

[33]. Девянин П.Н. Формальные модели управления доступом. Учебное пособие. М.: Горячая линия – Телеком, 2026. 424 с.: ил. / P.N. Devyanin. Formal access control models. Hotline-Telecom, 2026, 424 p. (in Russian).

[34]. GonnaCry Ransomware. Available at: https://github.com/tarcisio-marinho/GonnaCry, accessed 09.09.2026.

[35]. Ransomware-PoC. Available at: https://github.com/listinvest/Ransomware-PoC-1, accessed 09.09.2026.

[36]. Itssoeasy. Available at: https://github.com/bstnbuck/ItsSoEasy, accessed 09.09.2026.

[37]. Python-Ransomware. Available at: https://github.com/ncorbuk/Python-Ransomware, accessed 09.09.2026.

[38]. Kabbe J.-A. Security analysis of Docker containers in a production environment. Norwegian University of Science and Technology, 2017, 91 p.

[39]. Ke G., Meng Q., Finley T., Wang T., Chen W., Ma W., Ye Q., Liu T. Y. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Advances in Neural Information Processing Systems (NIPS), 2017, vol. 30, pp. 3146-3154.

[40]. Schuster M., Paliwal K. K. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 1997, vol. 45, no. 11, pp. 2673-2681.

[41]. Srivastava Nitish, Hinton Geoffrey, Krizhevsky Alex et al. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 2014, vol. 15, no. 1, pp. 1929-1958.

[42]. Radford A., Wu J., Child R., Luan D., Amodei D., Sutskever I. Language Models are Unsupervised Multitask Learners. OpenAI Blog, 2019.

[43]. Lin T.-Y., Goyal P., Girshick R., He K., Dollar P. Focal Loss for Dense Object Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2020, vol. 42, issue 2, pp. 318-327.

[44]. Loshchilov I., Hutter F. Decoupled Weight Decay Regularization. International Conference on Learning Representations (ICLR), 2019, 15 p.

[45]. Kingma D.P., Ba J. Adam: A Method for Stochastic Optimization. The International Conference on Learning Representations (ICLR), San Diego, 2015, 15 p.

[46]. Smith L.N., Topin N. Super-convergence: Very fast training of neural networks using large learning rates. In: Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications, 2019, vol. 11006, pp. 369-386.

[47]. Albert R. G. System Logs Anomaly Detection. Are we on the right path? Applied Artificial Intelligence, 2025, vol. 39, issue 1: 2440692.

[48]. Lin Z., et al. Contrastive learning for intrusion detection system. IEEE Access, 2023, vol. 11, pp. 12345-12356.

[49]. Dunmore A., Jang-Jaccard J., Sabrina F., Kwak J. A Comprehensive Survey of Generative Adversarial Networks (GANs) in Cybersecurity Intrusion Detection. IEEE Access, 2023, vol. 11, pp. 76071-76094.

Информация об авторах / Information about authors

Иван Олегович КЕМАЕВ – старший инженер ООО «РусБИТех-Астра» («Группа Астра»). Область интересов: искусственный интеллект, разработка безопасного программного обеспечения, операционные системы семейства Linux.

Ivan Olegovich KEMAEV – senior engineer in RusBITech-Astra (Astra Linux). Field of Interest: artificial intelligence, secure software development, operating systems of Linux family.

Петр Николаевич ДЕВЯНИН – академик Академии криптографии России, доктор технических наук, профессор, научный руководитель ООО «РусБИТех-Астра» («Группа Астра»). Область интересов: теория информационной безопасности, формальные модели безопасности компьютерных систем, разработка безопасного программного обеспечения, операционные системы семейства Linux.

Petr Nikolaevich DEVYANIN – Dr. Sci. (Tech.), academician 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, secure software development, operating systems of Linux family.