СРАВНИТЕЛЬНЫЙ АНАЛИЗ СОВРЕМЕННЫХ МЕТОДОВ ОБНАРУЖЕНИЯ АНОМАЛИЙ В КОНТЕЙНЕРНЫХ СРЕДАХ НА ОСНОВЕ СИСТЕМНЫХ ВЫЗОВОВ

Вьюгов С. Г.[1]

DOI:10.21681/3034-4050-2026-2-58-69

Ключевые слова: *обнаружение вторжений, аномальное обнаружение, системные вызовы, HIDS, анализ поведения программ, кибербезопасность, машинное обучение, графовые модели.*

Цель исследования: *систематизация и сравнительный анализ современных методов обнаружения атак в контейнерных средах на основе системных вызовов с выявлением их сильных сторон, ограничений и областей применимости.*

Методы исследования: *исследование основано на комплексном анализе существующих подходов к обнаружению атак в контейнерных средах, их систематизации и классификации по ключевым характеристикам: полноте используемой информации о системных вызовах, необходимости предварительного обучения и спектру обнаруживаемых угроз. Сравнительная оценка методов проведена на основе количественных метрик с использованием публичного набора данных LID-DS и результатов экспериментов на реальных микросервисных приложениях.*

Результаты исследования: *выполнена систематизация современных подходов к обнаружению аномалий в контейнерных средах по критериям полноты анализируемых данных, спектра обнаруживаемых угроз и операционных требований. Количественное сравнение на наборе LID-DS показало, что нейросетевой метод на основе двухэтапного автоэнкодера с механизмом внимания превосходит графовый метод, так как оказался единственным, обнаружившим все рассмотренные сценарии, включая CVE-2014-0160 и CVE-2020-13942. Выявлено, что PROCATCH при F1-score 0,999 для исполнительных атак принципиально не обнаруживает неисполняемые атаки (SQL-инъекции, утечки данных через уязвимости протоколов, эксплуатацию уязвимостей веб-приложений), а CHIDS не способен выявить аномалии, проявляющиеся исключительно в параметрах системных вызовов. На основе полученных результатов сформулированы практические рекомендации по выбору метода обнаружения атак в зависимости от требований к безопасности системы.*

Научная новизна: *заключается в систематизации и сравнительном анализе методов обнаружения вторжений в контейнерных средах по критериям полноты используемой информации, спектра обнаруживаемых угроз и операционных требований. Выявлении компромисса между простотой методов без обучения и полнотой обнаружения нейросетевых методов. Предложена классификация методов по типу анализируемых данных и уровню абстракции.*

Введение

Технология контейнеризации стала неотъемлемой частью современного мира, обеспечивая масштабируемость, переносимость и лёгкую виртуализацию приложений, особенно в облачных средах. Согласно данным опроса *Cloud Native Computing Foundation* (*CNCF*) [1], более 92 % респондентов используют контейнеры в производственных средах, что свидетельствует о массовом переходе организаций на контейнерные архитектуры. Инструменты оркестрации, такие как *Kubernetes* [2], позволяют эффективно управлять развёртыванием и масштабированием контейнерных приложений, предоставляя предприятиям возможность оперативно адаптировать и масштабировать свою инфраструктуру к изменяющимся нагрузкам. Вместе с широким распространением контейнеров возросло и количество угроз информационной безопасности. По данным того же опроса *CNCF*, 32 % организаций называют безопасность главной проблемой при работе с контейнерами. Контейнерные приложения представляют собой мишень для кибератак, которые могут привести к отказу в обслуживании критически важных сервисов, утечке данных и значительному финансовому ущербу.

Традиционные подходы к обеспечению безопасности контейнеров: сканирование образов и управление уязвимостями – являются элементами защиты конвейера *CI*/*CD*, однако они неспособны противодействовать угрозам, возникающим непосредственно во время выполнения. К таким угрозам относятся эксплуатация уязвимостей нулевого дня (*Heartbleed CVE-2014-0160*), атаки с повышением привилегий (*DirtyCOW*, *Dirty Pipe*), а также более сложные комплексные атаки (*Scarleteel*, *LABRAT*). Для противодействия

1 Вьюгов Станислав Георгиевич, сотрудник Академии Федеральной службы охраны Российской Федерации. Орел, Орел. E-mail: stas.viugov@yandex.ru

подобным угрозам необходимы системы обнаружения вторжений (*IDS*), способные выявлять аномальное поведение в реальном времени.

Существующие подходы к обнаружению вторжений в контейнерных средах разделяются на три класса. Первый – сигнатурные (*rule-based*) системы, такие как *Falco* и *Tracee*, основанные на предопределённых правилах обнаружения. Они эффективны против известных угроз, однако требуют значительных экспертных усилий для создания правил и не обнаруживают ранее неизвестные атаки. Второй – системы без обучения (*training-free*), использующие стабильность атрибутов исполняемых файлов для обнаружения отклонений. Третий – нейросетевые системы на основе анализа системных вызовов, обучаемые на трассировках нормального поведения. В отечественной литературе данная проблематика активно развивается: Котенко и Мельник предложили подход к обнаружению аномалий в контейнерных системах на основе частотного анализа дизассемблированных инструкций и гибридной нейронной сети *AE-LSTM*. В последующей работе ими же выполнена классификация методов обнаружения атак и аномалий на основе анализа аномалий и профилирования, интегрированный подход на основе чёрных и белых списков представлен в [3]. Каждый из подходов обладает существенными ограничениями, что определяет необходимость их систематического сравнительного анализа.

Системные вызовы как источник данных для обнаружения атак

Системные вызовы представляют собой программный интерфейс взаимодействия пользовательских процессов с ядром операционной системы *Linux*. Каждая операция, выполняемая внутри контейнера: работа с файловой системой, сетевое взаимодействие, управление процессами, межпроцессное взаимодействие – реализуется через один или несколько системных вызовов. Это делает системные вызовы наиболее информативным источником данных о поведении программного обеспечения на уровне системы. Каждый системный вызов характеризуется набором атрибутов:

- имя вызова (*syscall name*), определяющий тип системной операции;
- имя процесса (*process name*), идентифицирующий инициатора вызова;
- идентификатор потока (*thread ID*), позволяющий разделять действия различных потоков исполнения;
- возвращаемое значение (*return value*), указывающий на успешность выполнения;
- параметры (*arguments*), содержащий конкретные аргументы операции.

Авторский коллектив Forrest и др., предложивший модель *N-Gram* для описания поведения программного обеспечения через последовательности системных вызовов, исследования обнаружения аномалий на хосте последовательно расширяли объём используемой информации от простых имён вызовов через параметры к полной информации [4]. Сбор системных вызовов в контейнерных средах осуществлялся инструментами, которые взаимодействуют с ядром операционной системы Linux. Современные решения, такие как Falco, используют технологию *eBPF* (*Extended Berkeley Packet Filter*) для перехвата системных вызовов непосредственно в ядре с минимальными накладными расходами, что обеспечивает точность мониторинга без значительного влияния на производительность контролируемых контейнеров [5].

Анализ текущего состояния исследований в области обнаружения вторжений в контейнерных средах позволяет выделить ключевое противоречие: методы без обучения, избавляющие от операционных затрат на обучение и переобучение модели, принципиально ограничены в спектре обнаруживаемых угроз, в то время как нейросетевые методы, обеспечивающие более полный охват атак, требуют периодического переобучения при модификации контейнерного приложения.

Цель исследования – систематизация и сравнительный анализ современных методов обнаружения атак в контейнерных средах на основе системных вызовов с выявлением их сильных сторон, ограничений, слепых зон и областей применимости. Для достижения поставленной цели решаются следующие задачи:

1. Систематизировать существующие подходы к обнаружению аномалий в контейнерных средах на основе системных вызовов и предложить классификацию по типу анализируемых данных, спектру обнаруживаемых атак и операционным характеристикам.
2. Провести критический анализ четырёх современных методов (*CHIDS*, нейросетевой, *PROCATCH*, *AE-LSTM*) с количественным определением слепых зон каждого.
3. Формализовать модель обнаружения аномалий на основе полной информации о системных вызовах для определения базовой архитектуры нейросетевого подхода.
4. Выполнить сравнительный анализ рассмотренных подходов по критериям полноты обнаружения, точности, устойчивости к обходу,

операционных затрат и применимости к различным типам контейнерных сред.

Графовый метод контекстуализации системных вызовов

Работа El Khairi и соавторов [6], представленная на семинаре *CCSW'22* (*ACM*), предлагает систему обнаружения вторжений на уровне хоста (*HIDS*), основанную на мониторинге разнородных свойств системных вызовов с учётом их контекста. Авторы формулируют ключевую гипотезу: аномалии могут быть точно обнаружены, когда свойства системных вызовов рассматриваются совместно в рамках их относительного контекста, а не изолированно.

Предложенный метод использует граф системных вызовов процесса (*Process Syscall Sequence Graph*, $PSSG$) – взвешенный направленный граф $G = (N,E,W)$, в котором множество узлов N содержит комбинации имён системных вызовов и их значений возврата, множество ребер E отражает направление выполнения от исходных узлов к узлам назначения, а функция весов W определяет значимость узлов на основе обратной частоты обращений. Для каждого узла i вес вычисляется по формуле:

$$W_i = 1 - \sum_{j \in \text{neighbour}(i)} f_{ji}, \quad (1)$$

где $\text{neighbour}(i)$ – множество входящих соседей узла i, а f_{ji} – частота переходов от узла j к узлу i.

Метод анализирует три категории признаков: характеристики потока выполнения, характеристики файловых данных и характеристики частоты [7].

Подход был оценён на двух наборах данных из 20 сценариев атак (11 700 нормальных и 1 980 атакующих трассировок системных вызовов). Результаты показали эффективность обнаружения различных аномалий при разумных затратах времени. Однако выявлены существенные ограничения: в сценариях *CVE-2014-0160* (*Heartbleed*) и *CVE-2020-13942* (*Apache Struts*) метод не обнаружил аномалии (*recall* = 0, *F1-score* = 0) [8]. Это связано с двумя причинами: во-первых, модель использует только имена системных вызовов и значения возврата для построения *PSSG*, игнорируя имена процессов, идентификаторы потоков и параметры; во-вторых, для обнаружения аномалий используется относительно простая модель, не способная улавливать тонкие аномалии, проявляющиеся исключительно в параметрах системных вызовов (например, аномальный объём данных, передаваемых через сетевые вызовы при эксплуатации *Heartbleed*) [9].

Анализ результатов позволяет выделить три класса ограничений данного метода:

1. Система на основе *CHIDS* строит граф $PSSG$ исключительно по именам вызовов и знакам возвращаемых значений. Атаки, при которых последовательность и состав вызовов идентичны нормальному поведению, а аномалия проявляется только в параметрах (объём данных, адреса, аргументы), остаются полностью необнаруженными.
2. Пороговый классификатор *CHIDS* оперирует суммарными характеристиками графа и неспособен моделировать нелинейные зависимости между компонентами вектора признаков. Это приводит к пропуску атак со сложными, но статистически неяркими паттернами.
3. В отличие от четырёхкомпонентного вектора нейросетевого метода, *CHIDS* не содержит аналога компонента вектора характеристик системных ресурсов, что исключает обнаружение аномалий в объёмах передаваемых данных.

Нейросетевой метод на основе полной информации о системных вызовах

Работа Guo [10], представленная на конференции *DSAI 2024* (*ACM*), развивает графовый подход путём целенаправленного расширения набора анализируемых признаков. Автор впервые объединяет полный набор информации о системных вызовах – имена процессов, идентификаторы потоков, имена вызовов, возвращаемые значения и параметры в единую модель обнаружения вторжений. Архитектура системы включает три основных компонента, каждый из которых формализован математически. Модуль кодирования исходных данных преобразует последовательности системных вызовов в вектор аномалий A, состоящий из четырёх подвекторов:

$$A = [EFAV,UFV,FF,SRF], \quad (2)$$

где $EFAV$ – вектор аномалий потока выполнения (*Execution Flow Anomaly Vector*), UFV – вектор непросмотренных файлов (*Unseen File Vector*), FF – вектор характеристик частоты (*Frequency Features*), SRF – вектор характеристик системных ресурсов (*System Resource Features*).

Вектор аномалий потока выполнения $EFAV$ вычисляется на основе модифицированного графа $PSSG$, расширенного информацией об именах процессов, идентификаторах потоков и возвращаемых значениях. При обнаружении аномального ребра (ребра, отсутствующего в эталонном графе) $EFAV$ рассчитывается как:

$$EFAV = \sum_{i=1}^{M}(W_{sn_i} + w_{dn_i}), \quad (3)$$

где M – количество аномальных рёбер, W_{sn} – вес узла-источника, W_{dn} – вес узла назначения.

Вектор непросмотренных файлов UFV формируется при обнаружении обращений к файловым путям, не наблюдавшимся на этапе обучения. В отличие от предшествующей работы [4], использовавшей четыре системных вызова для мониторинга файловых операций, данный подход задействует тринадцать вызовов, обеспечивая более полный охват:

$$UFV = N \times \sum_{i=1}^{N} W_i, \quad (4)$$

где N – количество аномальных узлов, а W_i – вес аномального узла, определённый из $PSSG$.

Модель обнаружения вторжений на основе двухэтапного автоэнкодера реализует двухфазную архитектуру с механизмом внимания. На первом этапе входной вектор признаков x подвергается реконструкции, на втором этапе ошибка реконструкции первого этапа объединяется с входом и обрабатывается модулем многоголового внимания (рис. 1). Функция потерь определяется как:

$$L = \frac{1}{n} \times MSE(x, O_1) + (1 - \frac{1}{n}) \times MSE(x, O_2), \quad (5)$$

где n – номер эпохи обучения, MSE – функция средней квадратичной ошибки, O_1 – выход первого этапа, O_2 – выход второго этапа.

На ранних эпохах обучения (малые n) доминирует первый этап, обеспечивающий базовую

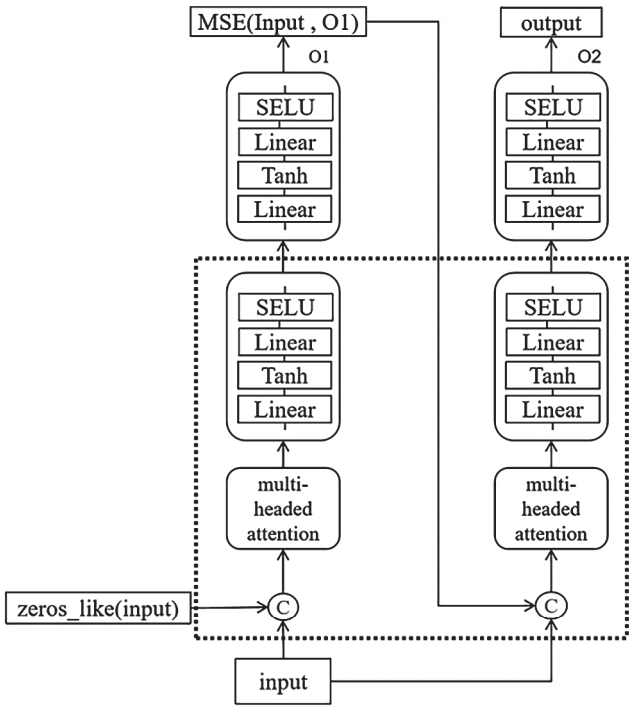

Рис. 1. Архитектура двухэтапного автоэнкодера

реконструкцию, по мере обучения (большие n) увеличивается вклад второго этапа с механизмом внимания, позволяющего выделять наиболее информативные паттерны аномалий.

Модуль динамического порога адаптивно изменяет порог обнаружения в зависимости от предшествующих результатов оценки:

$$\text{threshold} = \frac{1}{1 + e^{-\alpha(result - outlast)}} \times$$
$$\times \{-\beta \cdot (1 - \text{result} + \text{outlast}) \times \frac{1}{e} + \frac{1}{1 - e^{-0,2}}\} \times$$
$$\times (1 - e^{-w}), \quad (6)$$

где α управляет скоростью снижения порога, β – скоростью повышения, ω – скоростью начального повышения.

Экспериментально определены оптимальные значения: $\alpha = 1$, $\beta = 0,3$, $\omega = 50$. Когда в предыдущий момент обнаружена аномалия, порог быстро снижается, повышая чувствительность к последующим аномальным событиям в рамках продолжающейся атаки [11]. При нормальном поведении порог медленно возрастает, снижая число ложных срабатываний. Эксперименты по сравнению вышеизложенных методов проведены на публичном наборе данных LID-DS по 13 сценариям. Результаты представлены в таблице 1.

Результаты демонстрируют значительное улучшение: средний *F1-score* увеличился с 0,813 до 0,977 (+20,2 %), средний *recall* – с 0,795 до 0,968 (+21,8 %), среднее число ложных срабатываний снизилось с 3,23 до 2,84 (–12,1 %).

Принципиально важно, что метод успешно обнаружил все 13 сценариев атак, включая *CVE-2014-0160* (*Heartbleed*) и *CVE-2020-13942*, которые оставались полностью невидимыми для графового метода *CHIDS*. Это стало возможным благодаря учёту характеристик системных ресурсов – аномальных объёмов данных в параметрах сетевых вызовов при эксплуатации *Heartbleed* и более точному извлечению признаков потока выполнения [12].

Система обнаружения атак без обучения

Система *PROCATCH*, представленная El Khairi и соавторами [6,7], предлагает принципиально иную парадигму обнаружения вторжений, основанную на мониторинге стабильности атрибутов выполнения контейнерных микросервисов без какого-либо предварительного обучения. Ключевое наблюдение авторов: контейнерные микросервисы, следуя модели единой ответственности (*single-concern model*), выполняют единственную рабочую нагрузку на протяжении всего жизненного цикла, что обеспечивает стабильность

Таблица 1

Сравнительные результаты CHIDS и нейросетевого метода на наборе LID-DS

Сценарий	Precision (CHIDS)	Recall (CHIDS)	FP (CHIDS)	F1 (CHIDS)	Precision (NEURO)	Recall (NEURO)	FP (NEURO)	F1 (NEURO)
EPS_CWE-434	1,000	1,000	0	1,000	0,980	1,000	6	0,992
CVE-2020-9484	0,991	1,000	2	0,996	1,000	1,000	0	1,000
PHP_CWE-434	1,000	1,000	0	1,000	0,992	1,000	1	0,996
CVE-2020-23839	0,960	1,000	17	0,979	0,968	1,000	16	0,984
CVE-2019-5418	1,000	0,867	0	0,928	1,000	1,000	0	1,000
CVE-2014-0160	**0,000**	**0,000**	**1**	**0,000**	**0,967**	**1,000**	**11**	**0,984**
Bruteforce_CWE-307	0,956	0,949	19	0,953	1,000	0,978	0	0,989
CVE-2017-7529	1,000	0,916	0	0,956	1,000	1,000	0	1,000
CWE-89-SQL-injection	1,000	1,000	0	1,000	1,000	1,000	0	1,000
CVE-2018-3760	1,000	1,000	0	1,000	1,000	1,000	0	1,000
CVE-2012-2122	1,000	1,000	0	1,000	1,000	1,000	0	1,000
CVE-2020-13942	**0,000**	**0,000**	**0**	**0,000**	**1,000**	**0,992**	**0**	**0,996**
Juice Shop	0,986	0,611	3	0,755	0,988	0,620	3	0,761
Среднее	**0,838**	**0,795**	**3,23**	**0,813**	**0,992**	**0,968**	**2,84**	**0,977**

их атрибутов выполнения. Предварительное исследование на четырёх микросервисных приложениях (*BOOKINFO*, *SOCK-SHOP*, *MU-SHOP*, *MARTIAN BANK*) подтвердило, что атрибуты исполняемых файлов – имя (*exe*), путь (*exe_path*), индексный дескриптор (*exe_inode*) и текущий рабочий каталог (*cwd*) – сохраняют полную стабильность при нормальной работе. Единственным неустойчивым атрибутом является имя процесса (*proc*), подверженное изменениям из-за создания и переименования потоков в средах управляемого выполнения (*JVM*, *V8*, *HikariPool*).

Система функционирует в два этапа. На первом этапе после достижения подом состояния *PodReady* система идентифицирует контейнер, разрешает его контрольную группу (*cgroup*) через «*/proc/<PID>/cgroup*», перечисляет все *PID* в данной группе и для каждого процесса извлекает атрибуты выполнения (*exe*, *exe_path*, *exe_inode*, *cwd*) через файловую систему «*/proc*». На втором этапе система непрерывно отслеживает отклонения от базовой линии с использованием *Falco – eBPF*-инструмента [13].

PROCATCH демонстрирует выдающиеся показатели: средняя *precision* = 99,77 %, средний *recall* = 100,00 %. Время обнаружения атак составляет от 700 до 900 мс для большинства сценариев. Однако исследование устойчивости к обходу выявило существенные ограничения. При попытке злоумышленника использовать технику выполнения в контексте существующих интерпретаторов (*in-line execution*) результаты обнаружения на различных стадиях атаки распределяются следующим образом: разведка – 0 %, подготовка среды – 0 %, подготовка полезной нагрузки – 66,72 %, выполнение – 100,00 %. Таким образом, *PROCATCH* полностью не эффективен на стадиях разведки и подготовки среды. На стадии подготовки полезной нагрузки обнаружение достигается только в тех микросервисах, которые не содержат встроенных интерпретаторов (*Python*, *Node.js*), что составляет 66,72 % случаев. Для микросервисов на *Python* и *Node.js* подготовка полезной нагрузки через встроенные библиотеки (*urllib.urlretrieve, os.chmod*) остаётся полностью необнаруженной.

Кроме того, *PROCATCH* принципиально не обнаруживает неисполнительные атаки: *SQL*-инъекции, утечки данных через уязвимости типа Heartbleed, обход каталогов (*directory traversal*) и уязвимости веб-приложений (*CVE-2020-13942*). Эти типы атак не порождают новых исполняемых файлов и не изменяют атрибуты выполнения, поэтому остаются за пределами возможностей любого метода, основанного исключительно на мониторинге атрибутов *executables*.

Метод на основе гистограмм дизассемблированных инструкций (AE-LSTM)

Работа Котенко и Мельника [3, 4], предлагает принципиально отличный подход к обнаружению аномалий в контейнерных системах, основанный

на частотном анализе дизассемблированных машинных инструкций бинарных файлов выполняемых процессов.

В отличие от методов, анализирующих системные вызовы или атрибуты выполнения, данный подход оперирует на уровне содержимого исполняемых файлов. Сбор данных осуществляется с помощью *Falco Security* (*eBPF*) для получения путей к бинарным файлам, после чего утилитой *objdump* извлекаются дизассемблированные инструкции. На этапе нормализации формируются гистограммы процессов фиксированного размера путём подсчёта количества каждой инструкции и вычисления её относительной частоты:

$$h_i = \frac{count(instr_i)}{\sum_{j=1}^{K} count(instr_j)}, \quad (7)$$

где $count(instr_i)$ – количество вхождений i-й инструкции, K – общее число уникальных инструкций в словаре.

Для обнаружения аномалий применяется гибридная неконтролируемая модель нейронной сети *AE-LSTM* (*Autoencoder – Long Short-Term Memory*). *AE*-компонент сжимает входные данные в представление размерности *ROWS* × *LINES/2*, а *LSTM*-компонент обрабатывает последовательности гистограмм, учитывая временные зависимости между процессами. Обнаружение аномалий выполняется путём вычисления ошибки реконструкции: если *MSE* входного вектора после реконструкции превышает порог $threshold = = \mu_{MSE} + \sigma_{MSE}$, то последовательность классифицируется как аномальная.

Экспериментальная оценка проведена на испытательном стенде, включающем два *Docker*-контейнера (*ssh/rdp*-сервер и почтовый сервер *Postfix*), с тремя группами наборов данных: нормальная активность (группа A), аномальная (группа B) и вредоносная (группа C). Каждый набор содержит не менее 700 тысяч записей последовательностей гистограмм.

Авторы отмечают следующие особенности подхода:

- высокую точность обнаружения аномальных последовательностей процессов;
- низкий уровень ложных срабатываний;
- эффективное выявление атак, связанных с перехватом выполнения программного кода и манипулированием адресами функций в бинарных файлах.

Однако метод имеет принципиальное ограничение: неспособность анализировать программы на интерпретируемых языках и языках с байт-кодом, поскольку дизассемблирование применимо только к нативным бинарным файлам формата ELF.

В последующей работе [5] авторами выполнена систематическая классификация методов обнаружения атак и аномалий в контейнерных системах на основе анализа аномалий и профилирования, включая анализ подходов на основе глубокого обучения (*LSTM*, *AE*, *CNN*, *SOM* + *MLP*), с выявлением их особенностей, преимуществ и недостатков.

Систематизация рассмотренных подходов

На основании проведённого анализа сформирована классификация рассмотренных методов по ключевым характеристикам (табл. 2 и табл. 3).

Формализация нейросетевого подхода к обнаружению аномалий

На основании проведённого анализа опишем в формализованном виде модель обнаружения аномалий, предложенную в работе Guo [10] и основанную на нейросетевом анализе полной информации о системных вызовах.

Пусть $S = \{s_1, s_2, \ldots, s_n\}$ – последовательность системных вызовов, собранных в течение периода $\tau(\tau = 1$ с). Каждый системный вызов S_k характеризуется кортежем:

$$s_k = (name_k, proc_k, tid_k, ret_k, arg\ s_k), \quad (8)$$

где $name_k$ – имя системного вызова, $proc_k$ – имя процесса-инициатора, tid_k – идентификатор потока, ret_k – возвращаемое значение, $arg\ s_k$ – вектор параметров.

Этап 1. Построение графа системных вызовов процесса (PSSG)

Для каждого процесса $p \in P$, наблюдаемого в период τ, строится взвешенный направленный граф:

$$G_p = (N_p, E_p, W_p). \quad (9)$$

Множество узлов N_p формируется из пар $(name, ret\_sign)$, где $ret\_sign \in \{success, failure\}$ определяется знаком возвращаемого значения. Множество ребер E_p содержит направленные ребра между последовательно вызванными парами системных вызовов в рамках одного потока tid. Весовая функция W_p для узла i определяется формулой (1).

Этап 2. Извлечение вектора признаков

Вектор аномалий $A \in R^d$ для периода τ формируется как конкатенация четырех подвекторов:

$$A = EFAV \oplus UFV \oplus FF \oplus SRF. \quad (10)$$

Компонент $EFAV$ кодирует аномалии потока выполнения. При фиксированном эталоне графа G_p^{ref}, построенном на этапе обучения, для каждого ребра $(u,v) \in E_p$ текущего периода

Таблица 2

Классификация методов по типу анализируемых данных

Метод	Тип данных	Источник данных	Предобработка
CHIDS	Имена системных вызовов и возвращаемые значения	eBPF	Построение PSSG
Нейросетевой	Полная информация о системных вызовах	eBPF	Построение расширенно PSSG и извлечение 4-компонентного вектора
PROCATCH	Атрибуты выполнения	/proc, eBPF	Построение базовой линии атрибутов
AE-LSTM	Гистограммы дизассемблированных инструкций	objdump	Нормализация частот инструкций

Таблица 3

Матрица обнаруживаемых типов атак

Тип атаки	Пример	CHIDS	Нейро-сетевой	PROCATCH	AE-LSTM
Исполнительные (exec-based)	Kinsing, TeamTNT, Mirai	Да	Да	Да	Да
Утечки данных (data exfiltration)	Heartbleed (CVE-2014-0160)	Нет	Да	Нет	Нет
Инъекции	SQL-injection (CWE-89)	Да	Да	Нет	Нет
Удалённое выполнение кода	Apache Struts CVE-2020-13942	Нет	Да	Частично	Нет
Повышение привилегий	DirtyCOW, Dirty Pipe	Частично	Да	Да	Да
Криптоджекинг	Kinsing, скрытый майнинг	Да	Да	Да	Да
Разведка (reconnaissance)	Сканирование сети, перечисление	–	Частично	Нет	–
In-line execution (через интерпретаторы)	Python reverse shell	–	Да	66,72 %	Нет
Перехват выполнения бинарного кода	Подмена адресов функций в ELF	–	Частично	Частично	Да

проверяется его принадлежность множеству G_p^{ref}. Если ребро отсутствует в эталоне, оно классифицируется как аномальное и его вклад определяется суммой весов инцидентов узлов.

Компонент UFV аккумулирует информацию об обращениях к файловым путям, не наблюдавшимся при обучении, с использованием расширенного набора из 13 файловых системных вызовов: *open*, *openat*, *creat*, *rename*, *renameat*, *unlink*, *unlinkat*, *mkdir*, *mkdirat*, *rmdir*, *chmod*, *chown*, *link* [14].

Компонент FF отражает общую частоту системных вызовов за τ, без разделения по процессам, поскольку аномалии в частоте отдельных процессов отражаются в общей последовательности.

Компонент SRF кодирует аномалии в объемах данных, передаваемых через системные вызовы с параметрами размера (*send*, *sendto*, *recv*, *recfrom*, *read*, *write*). Данный компонент критически важен для обнаружения атак, при которых уязвимость проявляется исключительно в аномально большом объеме данных, возвращаемых через сетевые вызовы.

Этап 3. Двухэтапный автоэнкодер с механизмом внимания

Модель $f: R^d \rightarrow R^d$ обучается минимизировать ошибку реконструкции на данных нормального поведения. На первом этапе кодер E_1 и декодер D_1 формируют выход O_1:

$$O_1 = D_1(E_1(A)). \quad (11)$$

Ошибка реконструкции первого этапа $\varepsilon_1 = A - O_1$ конкатенируется с входом и обрабатывается модулем внимания MHA и вторым декодером D_2:

$$O_2 = D_2(MHA(A \oplus \varepsilon_1)). \quad (12)$$

Общая функция потерь определяется формулой (5), где балансировка между этапами контролируется номером эпохи n. При тестировании аномальность определяется сравнением

$MSE(A,O_2)$ с адаптивным порогом, вычисляемым по формуле (6).

Этап 4. Принятие решения

Для каждого периода вычисляется оценка аномальности:

$$score(\tau) = MSE(A_\tau, O_{2,\tau}). \quad (13)$$

Период классифицируется как аномальный, если $score > threshold(\tau)$, где $treshhold$ адаптивно вычисляется по формуле (6). Агрегация результатов по периодам внутри образа позволяет классифицировать образец как нормальный или атакующий.

Ключевые отличия от существующих решений

Модель, предложенная Guo, отличается от графового метода *CHIDS* по следующим параметрам:

1. **Полнота представления.** *CHIDS* использует для построения *PSSG* исключительно имена системных вызовов, игнорируя имена процессов, идентификаторы потоков и параметры. Предлагаемая модель расширяет узлы *PSSG* комбинациями (*name*, *ret_sign*) в контексте конкретного потока *tid* конкретного процесса *proc*, что обеспечивает более точное отображение потока выполнения.
2. **Мощность модели.** *CHIDS* применяет пороговый классификатор на основе совокупности извлечённых признаков. Предлагаемая двухэтапная архитектура автоэнкодера с механизмом многоголового внимания обладает существенно большей моделирующей способностью, позволяя улавливать тонкие аномалии в нелинейных зависимостях между компонентами вектора признаков.
3. **Охват файловых операций.** Расширение набора контролируемых файловых системных вызовов с 4 до 13 повышает полноту обнаружения атак, связанных с несанкционированным доступом к файловой системе.
4. **Учет системных ресурсов.** Введение компонента *SRF* позволяет обнаруживать атаки, проявляющиеся исключительно в аномальных объёмах передаваемых данных, что принципиально недоступно для *CHIDS*. Именно этот компонент обеспечил обнаружение *CVE-2014-0160*, при котором уязвимый сервер возвращает до 64 КБ содержимого оперативной памяти через обычный *TLS* запрос.

Отличие от системы без обучения *PROCATCH* носит фундаментальный характер: *PROCATCH* отслеживает дискретное множество атрибутов выполнения (*exe*, *exe_path*, *exe_inode*, *cwd*), тогда как предлагаемая модель анализирует непрерывный многомерный вектор признаков, извлечённый из полной информации о системных вызовах. Это даёт возможность обнаруживать атаки, не порождающие новые исполняемые файлы: *SQL*-инъекции, утечки данных, эксплуатацию уязвимостей приложений, аномальное использование сетевых ресурсов.

Анализ проблемы переобучения

Нейросетевые модели обнаружения вторжений на основе системных вызовов требуют обучения на данных нормального поведения и периодического переобучения при изменении профиля контейнерного приложения. Как отмечается в работе [16], базовые профили деградируют в динамических микросервисных средах, что требует частого переобучения – операционно затратного процесса. Причины необходимости переобучения можно классифицировать по трём категориям. Обновление версии микросервиса приводит к изменению набора выполняемых системных вызовов, структуры потоков исполнения и обращений к файловой системе. Изменение конфигурации (параметры запуска, переменные окружения, подключаемые библиотеки) может модифицировать паттерны взаимодействия с ядром. Изменение характера нагрузки (количество одновременных запросов, типы операций) влияет на частотные характеристики и объёмы передаваемых данных [15].

Интеграция в конвейер *CI/CD*. В современных микросервисных архитектурах развёртывание новых версий приложений осуществляется через автоматизированные конвейеры непрерывной интеграции и доставки. Переобучение модели обнаружения вторжений может быть встроено в этот конвейер как автоматический этап, выполняемый после успешного прохождения функциональных тестов. Формально конвейер расширяется следующим образом:

$$Pipeline_{extended} = Build \rightarrow Test \rightarrow Train_{IDS} \rightarrow \rightarrow Deploy \rightarrow Monitor. \quad (14)$$

На этапе $Train_{IDS}$ модель обучается на трассировках системных вызовов, собранных в ходе функционального тестирования. Поскольку тестовая среда воспроизводит типичные сценарии использования микросервиса, собранные данные достоверно представляют нормальное поведение новой версии. Модуль извлечения признаков на основе *PSSG* существенно снижает размерность данных. Из миллионов системных вызовов за один образец формируется вектор размерности d, порядок которой определяется числом уникальных процессов, системных вызовов и файловых путей. Экспериментальные

данные [16] показывают, что обучение на наборе *LID-DS* завершается за приемлемое для промышленного использования время.

Контейнерные микросервисы, спроектированные в соответствии с принципом единой ответственности, демонстрируют высокую стабильность поведения между версиями. Как подтверждено в исследовании [16], атрибуты выполнения сохраняют полную стабильность в пределах жизненного цикла микросервиса. Это означает, что переобучение требуется не непрерывно, а дискретно – при каждом обновлении версии. Перспективным направлением снижения операционных затрат является переход от полного переобучения к инкрементному обновлению модели. Формально:

$$\theta_{new} = \theta_{\{old\}} - \eta \nabla_{\theta} L(D_{new}; \theta_{old}), \quad (15)$$

где θ – параметры модели, η – скорость обучения (существенно меньшая, чем при полном обучении), D_{new} – набор данных нового нормального поведения. Такой подход сокращает время переобучения в десятки раз по сравнению с обучением с нуля.

Для корректной оценки перспективности нейросетевого подхода необходимо сопоставить операционные затраты на переобучение (C_{train}) с ожидаемыми потерями от пропущенных атак (C_{miss}):

$$C_{retrain} \ll C_{miss} = P_{attack} \times P_{miss} \times D_{impact}, \quad (16)$$

где P_{attack} – вероятность атаки за цикл обновления, P_{miss} – вероятность пропуска атаки методом без обучения, D_{impact} – ожидаемый ущерб от инцидента.

По данным исследований [17, 18], публичные контейнерные микросервисы подвергаются атакам с высокой частотой. При этом P_{miss} для *PROCATCH* составляет не менее 33,28 % для атак через встроенные интерпретаторы и 100 % для неисполнительных атак. В противоположность этому, стоимость $C_{retrain}$ ограничивается вычислительными ресурсами на обучение модели и интеграцией в существующий *CI/CD*-конвейер. Таким образом, даже при консервативных оценках выполняется неравенство $C_{retrain} \ll C_{miss}$, что экономически обосновывает выбор нейросетевого подхода с периодическим переобучением [19].

Сравнительный анализ подходов

Для систематизации результатов проведённого анализа представим сравнение рассмотренных подходов по ключевым критериям.

Анализ таблицы 4 показывает, что нейросетевой метод на основе полной информации о системных вызовах является единственным подходом, обеспечивающим обнаружение всех рассмотренных классов атак. *PROCATCH*, несмотря на более высокий *F1-score* в пределах своей области применения (исполнительные атаки), обладает фундаментальной слепой зоной, составляющей значительную часть реального ландшафта угроз [20]. Метод *AE-LSTM* дополняет систему обнаружения на ином уровне абстракции – анализ содержимого бинарных файлов позволяет выявлять атаки, при которых злоумышленники перехватывают выполнение программного кода и манипулируют адресами функций, однако

Таблица 4

Сравнительный анализ подходов к обнаружению атак

Критерий	CHIDS	Нейросетевой	PROCATCH	AE-LSTM
Метод обнаружения	Графовый + порог	Автоэнкодер с вниманием	Сравнение атрибутов	AE-LSTM + гистограммы
Тип анализируемых данных	Имена системных вызовов и возвращаемые значения	Полная информация о системных вызовах	Атрибуты выполнения	Дизассемблир. инструкции
Необходимость обучения	Да	Да	Нет	Да
Средний F1-score	0,813	0,977	0,999	Высокий
Обнаружение атак без исполнения	Частично	Да	Нет	Нет
Устойчивость к обходу	–	Частично	0 %	–
Анализ Python/Java	Да	Да	Частично	Нет
Число сценариев с recall = 0	2 из 13	0 из 13	0 из 10	–
Набор данных	LID-DS	LID-DS	Реальные атаки	Собственный стенд

он неприменим к интерпретируемым средам и не обнаруживает атаки, не порождающие аномалий на уровне машинных инструкций.

Выводы

В ходе исследования получены следующие результаты:

1. Проведён систематический сравнительный анализ четырёх современных подходов к обнаружению вторжений в контейнерных средах. Предложена классификация методов по типу анализируемых данных. Выявлены ключевые ограничения каждого метода.
2. Систематизирована и представлена в формализованном виде модель обнаружения аномалий, включающая: расширенный граф *PSSG*, четырёхкомпонентный вектор признаков, двухэтапный автоэнкодер с механизмом внимания и адаптивный динамический порог. Описаны формулы весовых функций, векторов аномалий, функции потерь и динамического порога.
3. На экспериментальных данных набора *LID-DS* продемонстрировано значительное превосходство нейросетевого подхода: средний *F1-score* увеличился с 0,813 (*CHIDS*) до 0,977 (рост на 20,2 %), средний *recall* – с 0,795 до 0,968 (рост на 21,8 %), число обнаруженных сценариев – с 11 до 13 из 13 (полный охват). Среднее число ложных срабатываний снизилось с 3,23 до 2,84 (снижение на 12,1 %).
4. Проведен анализ проблемы переобучения и обоснован выбор в пользу систем с переобучением.

Литература

1. CNCF Survey 2020. – 2020. – URL: https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf (дата обращения: 15.02.2026).
2. Aqasizade, H. Kubernetes in Action: Exploring the Performance of Kubernetes Distributions in the Cloud / H. Aqasizade, E. Ataie, M. Bastam // Software – Practice and Experience. – 2025. – Vol. 55, No. 10. – P. 1711–1725. – DOI 10.1002/spe.70000. – EDN DRFXVH.
3. Котенко, И. В. Обнаружение аномалий в контейнерных системах: применение частотного анализа и гибридной нейронной сети / И. В. Котенко, М. В. Мельник // Программные продукты и системы. – 2025. – № 3. – С. 426–437. – DOI 10.15827/0236-235X.151.426-437. – EDN GMQDSA.
4. Котенко, И. В. Обнаружение атак и аномалий в контейнерных системах: подходы на основе анализа аномалий и профилирования / И. В. Котенко, М. В. Мельник // Искусственный интеллект и принятие решений. – 2025. – № 2. – С. 3–18. – DOI 10.14357/20718594250201. – EDN DZEPTN.
5. Kotenko I., Melnik M., Abramenko G. Anomaly detection in container systems: using normal process histograms and an autoencoder // 2024 IEEE 25th International Conference of Young Professionals in Electron Devices and Materials (EDM 2024). IEEE. 2024. P.1930–1934. DOI: 10.1109/EDM61683.2024.10615118.
6. Khairi, Asbat & Peter, Andreas & Continella, Andrea. (2025). PROCATCH: Detecting Execution-Based Anomalies in Single-Instance Microservices. 1–9. 10.1109/CNS66487.2025.11194959.
7. El Khairi A., Caselli M., Knierim C., Peter A., Continella A. Contextualizing System Calls in Containers for Anomaly-Based Intrusion Detection // Proceedings of the 2022 Cloud Computing Security Workshop (CCSW '22). – ACM, 2022. – P. 9–21.
8. Jain V., Singh B., Khenwar M., Sharm M. Static vulnerability analysis of docker images // IOP Conference Series: Materials Science and Engineering. IOP Publishing. 2021. V. 1131. No 1. P. 012018. EDN: KUZGXN.
9. Nakata R., Otsuka A. Evaluation of vulnerability reproducibility in container-based Cyber Range // arXiv preprint arXiv:2010.16024. 2020.
10. Guo P. Intrusion Detection Based on Complete System Call Information // 2024 International Conference on Digital Society and Artificial Intelligence (DSAI 2024). – ACM, 2024. – 5 p.
11. Ahmed M. E., Kim H., Camtepe S., Nepal S. Peeler: Profiling kernel-level events to detect ransomware // Computer Security-ESORICS 22: 26th European Symposium on Research in Computer Security. Springer International Publishing. 2021. P. 240–260.
12. Tien C. W., Huang T. Y., Tien C. W., Huang T. C., Kuo S. Y. KubAnomaly: Anomaly detection for the Docker orchestration platform with neural network approaches // Engineering reports. 2019. V. 1. No 5. P. e12080.
13. Gantikow H., Zohner T., Reich C. Container anomaly detection using neural networks analyzing system calls // 2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). IEEE. 2020. P. 408–412.
14. Kosinska J., Tobiasz M. Detection of Cluster Anomalies With ML Techniques // IEEE Access. 2022. V. 10. P. 110742–110753. EDN: VQHQOQ.
15. Wang Y., Wang Q., Qin X., Chen X., Xin B., Yang R. DockerWatch: a two-phase hybrid detection of malware using various static features in container cloud // Soft Computing. 2023. V. 27. No 2. P. 1015–1031. EDN: ISIKGC.
16. Chan K. Y., Abu-Salih B., Qaddoura R., Ala'M A. Z., Palade V., Pham D. S., Javier D. S., Muhammad K. Deep neural networks in the cloud: Review, applications, challenges and research directions // Neurocomputing. 2023. P. 126327.

17. Grimmer M., Röhling M. M., Kreusel D., Ganz S. A Modern and Sophisticated Host Based Intrusion Detection Data Set // IT-Sicherheit als Voraussetzung für eine erfolgreiche Digitalisierung. – 2019. – P. 135–145.
18. Castanhel G. R., Heinrich T., Ceschin F., Maziero C. Taking a peek: An evaluation of anomaly detection using system calls for containers // 2021 IEEE Symposium on Computers and Communications (ISCC). IEEE. 2021. P. 1–6.
19. Karn R. R., Kudva P., Huang H., Suneja S., Elfadel I. M. Cryptomining detection in container clouds using system calls and explainable machine learning // IEEE transactions on parallel and distributed systems. 2020. V. 32. No 3. P. 674–691.
20. Abubakar A. I., Chiroma H., Muaz S. A., Ila L. B. A review of the advances in cyber security benchmark datasets for evaluating data-driven based intrusion detection systems // Procedia Computer Science. 2015. V. 62. P. 221–227.

ICOMPARATIVE ANALYSIS OF CURRENT METHODS FOR DETECTING ANOMALIES IN CONTAINER ENVIRONMENTS BASED ON SYSTEM CALLS

***Vyugov S. G.*[2]**

Keywords: *intrusion detection, anomaly detection, system calls, HIDS, program behavior analysis, cybersecurity, machine learning, graph-based models.*

Abstract

Purpose of the study: *to systematize and conduct a comparative analysis of modern attack detection methods in containerized environments based on system calls, identifying their strengths, limitations, and areas of applicability.*

Methods of research: *the study is based on a comprehensive analysis of existing approaches to attack detection in containerized environments, including their systematization and classification according to key characteristics: completeness of system call information utilized, requirement for prior training, and the spectrum of detectable threats. A comparative evaluation was performed using quantitative metrics on the public LID-DS dataset, as well as experimental results obtained from real microservice-based applications.*

Results: *a systematic classification of modern detection approaches in containerized environments was performed based on the completeness of analyzed data, the range of detectable threats, and operational requirements. Quantitative comparison on the LID-DS dataset demonstrated that the neural network-based two-stage autoencoder method with an attention mechanism outperforms the graph-based method, as it was the only approach that successfully detected all considered attack scenarios, including CVE-2014-0160 and CVE-2020-13942. It was found that PROCATCH, despite achieving an F1-score of 0.999 for executable attacks, fundamentally fails to detect non-executable attacks (such as SQL injections, data exfiltration via protocol vulnerabilities, and exploitation of web application vulnerabilities). CHIDS was shown to be incapable of detecting anomalies manifested exclusively in system call parameters. Based on the obtained results, practical recommendations were formulated for selecting an attack detection method depending on system security requirements.*

Scientific novelty: *the novelty of this research lies in the systematization and comparative analysis of intrusion detection methods in containerized environments according to the completeness of utilized information, the spectrum of detectable threats, and operational requirements. A trade-off between the simplicity of training-free methods and the detection completeness of neural network–based approaches is identified. A classification of methods based on the type of analyzed data and the level of abstraction is proposed.*

References

1. CNCF Survey 2020. – 2020. – URL: https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf (data obrashheniya: 15.02.2026).
2. Aqasizade, H. Kubernetes in Action: Exploring the Performance of Kubernetes Distributions in the Cloud / H. Aqasizade, E. Ataie, M. Bastam // Software - Practice and Experience. – 2025. – Vol. 55, No. 10. – P. 1711–1725. – DOI 10.1002/spe.70000. – EDN DRFXVH.
3. Kotenko, I. V. Obnaruzhenie anomalij v kontejnerny'x sistemax: primenenie chastotnogo analiza i gibridnoj nejronnoj seti / I. V. Kotenko, M. V. Mel'nik // Programmny'e produkty' i sistemy'. – 2025. – № 3. – S. 426–437. – DOI 10.15827/0236-235X.151.426-437. – EDN GMQDSA.
4. Kotenko, I. V. Obnaruzhenie atak i anomalij v kontejnerny'x sistemax: podxody' na osnove analiza anomalij i profilirovaniya / I. V. Kotenko, M. V. Mel'nik // Iskusstvenny'j intellekt i prinyatie reshenij. – 2025. – № 2. – S. 3–18. – DOI 10.14357/20718594250201. – EDN DZEPTN.
5. Kotenko I., Melnik M., Abramenko G. Anomaly detection in container systems: using normal process histograms and an autoencoder // 2024 IEEE 25th International Conference of Young Professionals in Electron Devices and Materials (EDM 2024). IEEE. 2024. P.1930-1934. DOI: 10.1109/EDM61683.2024. 10615118.

2 Stanislav G. Vyugov, Academy of the Federal Security Service of the Russian Federation. Orel, Russia. E-mail: stas.viugov@yandex.ru

6. Khairi, Asbat & Peter, Andreas & Continella, Andrea. (2025). PROCATCH: Detecting Execution-Based Anomalies in Single-Instance Microservices. 1–9. 10.1109/CNS66487.2025.11194959.
7. El Khairi A., Caselli M., Knierim C., Peter A., Continella A. Contextualizing System Calls in Containers for Anomaly-Based Intrusion Detection // Proceedings of the 2022 Cloud Computing Security Workshop (CCSW 22). – ACM, 2022. – P. 9–21.
8. Jain V., Singh B., Khenwar M., Sharm M. Static vulnerability analysis of docker images // IOP Conference Series: Materials Science and Engineering. IOP Publishing. 2021. V. 1131. No 1. P. 012018. EDN: KUZGXN.
9. Nakata R., Otsuka A. Evaluation of vulnerability reproducibility in container-based Cyber Range // arXiv preprint arXiv:2010.16024. 2020.
10. Guo P. Intrusion Detection Based on Complete System Call Information // 2024 International Conference on Digital Society and Artificial Intelligence (DSAI 2024). – ACM, 2024. – 5 p.
11. Ahmed M. E., Kim H., Camtepe S., Nepal S. Peeler: Profiling kernel-level events to detect ransomware // Computer Security-ESORICS 22: 26th European Symposium on Research in Computer Security. Springer International Publishing. 2021. P. 240–260.
12. Tien C. W., Huang T. Y., Tien C. W., Huang T. C., Kuo S. Y. KubAnomaly: Anomaly detection for the Docker orchestration platform with neural network approaches // Engineering reports. 2019. V. 1. No 5. P. e12080.
13. Gantikow H., Zohner T., Reich C. Container anomaly detection using neural networks analyzing system calls // 2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). IEEE. 2020. P. 408–412.
14. Kosinska J., Tobiasz M. Detection of Cluster Anomalies With ML Techniques // IEEE Access. 2022. V. 10. R. 110742–110753. EDN: VQHQOQ.
15. Wang Y., Wang Q., Qin X., Chen X., Xin B., Yang R. DockerWatch: a two-phase hybrid detection of malware using various static features in container cloud // Soft Computing. 2023. V. 27. No 2. P. 1015–1031. EDN: ISIKGC.
16. Chan K. Y., Abu-Salih B., Qaddoura R., AlaM A. Z., Palade V., Pham D. S., Javier D. S., Muhammad K. Deep neural networks in the cloud: Review, applications, challenges and research directions // Neurocomputing. 2023. P. 126327.
17. Grimmer M., Röhling M. M., Kreusel D., Ganz S. A Modern and Sophisticated Host Based Intrusion Detection Data Set // IT-Sicherheit als Voraussetzung für eine erfolgreiche Digitalisierung. – 2019. – P. 135–145.
18. Castanhel G. R., Heinrich T., Ceschin F., Maziero C. Taking a peek: An evaluation of anomaly detection using system calls for containers // 2021 IEEE Symposium on Computers and Communications (ISCC). IEEE. 2021. P. 1–6.
19. Karn R. R., Kudva P., Huang H., Suneja S., Elfadel I. M. Cryptomining detection in container clouds using system calls and explainable machine learning // IEEE transactions on parallel and distributed systems. 2020. V. 32. No 3. P. 674–691.
20. Abubakar A. I., Chiroma H., Muaz S. A., Ila L. B. A review of the advances in cyber security benchmark datasets for evaluating data-driven based intrusion detection systems // Procedia Computer Science. 2015. V. 62. P. 221–227.