# Системный подход к безопасности пользовательских файлов: от первичной валидации до изоляции в Docker

О.В. Ехлаков

***Аннотация*—В статье представлено описание системного подхода к обеспечению безопасности пользовательских файлов в контейнеризованных средах. Исследован системно-интегрированный подход к защите пользовательских файлов в контейнеризованных средах. В работе предлагается и обосновывается многоуровневый алгоритм обеспечения безопасности пользовательских файлов, интегрирующий процедуры валидации, моделирования угроз и изоляции в единый конвейер. Алгоритм, описанный в рамках работы, включает этапы первичного статического и динамического анализа загружаемых сведений, превентивное threat-моделирование жизненного цикла Docker-контейнера с использованием деревьев атак и модели DREAD, а также применение комплекса мер по изоляции. Для статического сканирования образов применяются Clair, Trivy и Docker Scout, что позволяет выявлять уязвимости на уровне слоёв контейнера ещё до его запуска. Параллельно организуется динамический мониторинг при помощи Falco и Sysdig Secure, обеспечивающий непрерывное обнаружение аномалий и неподозреваемых эксплойтов. Механизмы изоляции контейнеров включают использование user namespaces, профилей seccomp и ограничений привилегий и ресурсов, что значительно снижает возможность эскалации прав. Представленные результаты имеют практическую ценность для исследователей и архитекторов информационной безопасности, работающих над формализацией гетерогенных моделей доверия и разработкой многоуровневых механизмов валидации пользовательских файлов, а также для DevSecOps-инженеров и администраторов Docker-платформ, стремящихся интегрировать методы динамической изоляции и продвинутые стратегии обнаружения аномалий в корпоративные CI/CD-пайплайны.**

***Ключевые слова*—контейнеризация, безопасность файлов, статический анализ, динамический мониторинг, Docker, threat-моделирование, DREAD**

## I. Введение

В современных условиях контейнеризация, прежде всего посредством Docker, зарекомендовала себя как ключевой инструмент сегрегации приложений и их зависимостей. Тем не менее, контейнеры остаются потенциальными векторами атак: злоумышленники могут осуществить escape из изолированной среды и посягнуть на хостовую систему или соседние контейнеры [1]. Параллельно с этим надёжная предварительная валидация форматов пользовательских файлов выступает непременным элементом защиты на уровне бизнес-логики приложений.

В научной литературе по вопросам безопасности контейнеризации и мониторинга пользовательских файлов можно выделить несколько направлений исследования.

Во-первых, исследования, посвящённые выявлению и анализу уязвимостей утечек данных из контейнера. В своей диссертации Aktolga İ. T. выполняет детальный разбор механизмов escape в Docker и предлагает комплекс методов статического и динамического анализа образов для обнаружения избыточных разрешений и рискованных системных вызовов контейнерного ядра [1]. Исследование Rajyashree R. et al. демонстрирует, каким образом небезопасно примонтированные Docker-сокеты могут предоставить злоумышленнику полномочия хоста, и формулирует рекомендации по усилению политик AppArmor и SELinux для защиты сокетов [7]. Shi H. et al. проводят масштабную эмпирическую оценку более миллиона публичных Docker-образов, систематизируя типичные ошибки в конфигурациях и зависимостях, что ляжет в основу практических рекомендаций по безопасному формированию и управлению жизненным циклом контейнеров [5].

Второе направление — эмпирическое тестирование и оценка безопасности контейнеров через пентестинг. Mubanda D. et al. применяют классические методики тестирования на проникновение к запущенным Docker-контейнерам, выявляя критические зоны как в сетевых настройках, так и в управлении томами и секретами. Авторы предлагают процедуру автоматизированного сканирования с помощью Metasploit, Docker Bench for Security и собственных тестовых сценариев, что способствует более оперативному обнаружению уязвимостей [6].

Третье направление объединяет работы, фокусирующиеся на стратегиях смягчения рисков и управлении безопасностью на уровне контейнеров и оркестрации. VS D. P., Sethuraman S. C. и Khan M. K. разрабатывают классификацию уровней предосторожности для контейнеров, исследуют изоляционные механизмы на базе namespaces и cgroups, а также описывают перспективные архитектуры безопасных контейнерных платформ, определяя вектор развития формальных моделей безопасности для контейнеров [2]. Mahavaishnavi V. et al. предлагают фреймворк для обнаружения и устранения уязвимостей на уровне оркестратора Kubernetes, включая аудит API-эндпоинтов, контроль цепочек доверия и автоматизацию пересборки компонентов при нарушении политик безопасности [10].

Четвёртое направление связано с управлением производительностью и уязвимостями контейнеров в привязке к нагрузке приложений. Alyas T. et al.

исследуют корреляцию метрик CPU, памяти и I/O с вероятностью возникновения уязвимостей в зависимости от состава библиотек и параметров запуска, и описывают процесс непрерывного мониторинга, где статистические методы обнаружения аномалий сочетаются с классификацией по сигнатурам известных уязвимостей [4]. Borglund N. рассматривает практики безопасного деплоя Docker-приложений, акцентируя внимание на управлении секретами, сегментации сети и интеграции проверок образов непосредственно в CI/CD-пайплайн [3].

Пятое направление касательно микросервисной архитектуры и контейнеризации представлено работами Sah K. P. et al. [8] и Mubanda D. et al. [9]. В рамках данного направления авторы в разные годы обсуждали эволюцию микросервисов под Docker, затрагивая вопросы сетевой безопасности сервисов, маршрутизации через сервисную сетку (service mesh) и применение шифрования внутри кластера, что отражает растущую потребность в многоуровневой защите при распределённом исполнении микросервисов.

Таким образом, несмотря на многообразие методик, среди работ встречаются противоречия: одни авторы делают упор на превентивную проверку образов до запуска, тогда как другие фокусируются на реактивных механизмах в работающей среде посредством оркестрации. Кроме того, исследования, направленные на оптимизацию производительности, часто используют упрощённые модели угроз и не охватывают сложные сценарии атак на сокеты и API-эндпоинты, выявленные в пентест-исследованиях. Эти пробелы открывают перспективы для дальнейших исследований и разработки интегрированных системных решений в рамках комплексного подхода.

Целью исследования является разработка и теоретическое обоснование алгоритма комплексной защиты пользовательских файлов в контейнеризованных средах, основанного на интеграции статических, динамических и инфраструктурных методов контроля.

Научная новизна работы заключается в предложении нового многоуровневого алгоритма защиты, который, в отличие от существующих узкоспециализированных подходов, обеспечивает сквозной контроль безопасности файла на всех этапах его жизненного цикла — от загрузки до обработки в изолированной контейнерной среде.

Авторская гипотеза базируется на том, что синергия надёжной предварительной валидации пользовательских файлов и построения многоступенчатой изоляции в контейнерной инфраструктуре позволит снизить шанс успешной компрометации данных и минимизировать потенциальный ущерб.

Теоретической и методологической основой исследования послужили: системный анализ, методы моделирования угроз (attack trees, DREAD) и сравнительный анализ существующих подходов к безопасности контейнеров. На основе синтеза данных подходов разработан алгоритм защиты.

## II. Первичная валидация пользовательских файлов

В современных веб- и облачных платформах этап приёма и обработки файлов от пользователей остаётся одним из наиболее критичных с точки зрения безопасности: именно здесь злоумышленник может вмешаться в бизнес-логику приложения и получить несанкционированный доступ к его ресурсам. Формальный анализ угроз на этапе загрузки позволяет выделить следующие основные векторы атак:

1. Инъекции и обход валидации. Злоумышленник может намеренно изменить расширение файла (например, превратив image.jpg в shell.php.jpg) или манипулировать заголовком Content-Type HTTP-запроса, чтобы пробить фильтрацию, основанную лишь на расширении или MIME-типе. Такие приёмы позволяют проникнуть мимо поверхностных проверок, не затрагивающих содержимое и внутреннюю структуру файла.
2. Эксплуатация уязвимостей парсеров. Сложные форматы (PDF, Office-документы) могут содержать макросы или скрытые объекты, в которых реализованы буферные переполнения или RCE-эксплоиты. При отсутствии проверки и изоляции загрузка таких файлов способна привести к компрометации памяти сервера либо отказу в обслуживании (DoS) на уровне процесса парсера.
3. DoS-атаки на сервер. Если загрузить чрезвычайно объёмный или специально «подготовленный» бинарный поток, то операции его разбора, валидации и конвертации могут полностью исчерпать ресурсы CPU и оперативной памяти. Это особенно опасно для контейнеризованных микросервисов, где без ограничений на потребление ресурсов подобные файлы способны вывести сервис из строя [1, 2].

Для противодействия перечисленным угрозам применяются как статические, так и динамические методы анализа. В таблице 1 представлены основные подходы и инструменты первичной валидации пользовательских файлов.

Таблица I.

Основные методы и инструменты первичной валидации пользовательских файлов [1, 2, 6].

| Метод | Тип анализа | Инструменты / Библиотеки | Преимущества | Ограничения |
|---|---|---|---|---|
| Белый список расширений | статический | собственная логика валидации | высокая скорость, простота реализации | легко обойти путём изменения расширения |
| Проверка MIME-типа по содержимому | статический | libmagic (file), Apache Tika | проверка «по факту» содержимого | может допускать false-positive/negative, накладные расходы |
| Сканирование | статический | ClamAV | обширная база | неэффективен против zero- |

| Метод | Тип анализа | Инструменты / Библиотеки | Преимущества | Ограничения |
|---|---|---|---|---|
| по сигнатурам | | | сигнатур, бесплатный | day и полиморфного вредоносного ПО |
| Статический анализ макросов | статический | oletools | детекция вредоносных макросов в MS Office | поддерживает только портированные форматы |
| Динамический анализ в песочнице | динамический | Cuckoo Sandbox | эмуляция среды, выявление поведения файла при исполнении | ресурсоёмкость, время выполнения теста |
| Гибридный (статический + динамический) | комбинированный | интеграция в CI/CD (Jenkins, GitLab CI + плагины анализа) | баланс скорости, точности и автоматизации | сложность настройки, дополнительные ресурсы |

Применение политики предопределённого перечня расширений файлов (например, .jpg, .png, .pdf) обеспечивает оперативную фильтрацию подавляющего большинства неподходящих объектов, однако неустойчиво к намеренному изменению расширения при модификации имени файла. Для преодоления данной уязвимости широко используется библиотека libmagic, анализирующая сигнатуры в первых байтах («magic bytes») и надёжно идентифицирующая внутренний формат файла вне зависимости от расширения. Аналогичным образом фреймворк Apache Tika обрабатывает структурированные документы (Microsoft Word, Excel, PDF), извлекая метаданные и содержимое для дальнейшего анализа. Антивирусные решения, такие как ClamAV, базируются на сигнатурных базах CVE для выявления известных вредоносных образцов, а интеграция с API VirusTotal позволяет единовременно запускать проверку через множество сканеров, хотя эксплойты нулевого дня могут отсутствовать в их репозиториях. В отношении офисных документов особое значение имеют средства oletools, способные детализированно исследовать VBA-макросы и выявлять потенциально опасные конструкции (например, автозапуск и Shell-вызовы). Методы динамического анализа на примере Cuckoo Sandbox разворачивают файл в изолированной виртуальной среде (виртуальная машина или контейнер) и мониторят его поведение (сетевые запросы, процессы, изменения файловой системы), что позволяет эффективно детектировать ранее неизвестные вредоносные программы [3, 5].

Следовательно, этап первичной валидации пользовательского контента представляет собой критически важный компонент стратегии defence-in-depth. Синергия классических методов (белые списки расширений, проверка MIME-типа) с современными решениями анализа (ClamAV, oletools, Cuckoo Sandbox) существенно повышает надёжность системы в целом. Интеграция этих проверок непосредственно в CI/CD-пайплайн позволяет выявлять и устранять угрозы ещё на ранних этапах жизненного цикла приложений, снижая нагрузку и риски для последующих слоёв защиты.

## III. Контейнеризация как уровень инфраструктурной защищённости

Технология контейнеризации обеспечивает минимально затратную изоляцию на уровне ядра операционной системы, однако встроенные механизмы namespace и cgroups сами по себе не способны обеспечить стопроцентную защиту: злоумышленник может преодолеть ограничения контейнера и осуществить атаку как на хост-машину, так и на соседние контейнеры [10]. Для управления безопасностью контейнерной инфраструктуры требуется построить всестороннее threat-моделирование, охватывающее все фазы жизненного цикла контейнера, — это позволит определить критические точки входа атак и выработать иерархию мер по снижению рисков.

В таблице 2 приведён перечень потенциальных угроз на каждом из этапов жизненного цикла контейнера и соответствующие стратегии их нейтрализации.

Таблица II.
Основные угрозы на каждой фазе жизненного цикла контейнера и соответствующие меры смягчения [1, 7, 10].

| Фаза жизненного цикла | Ключевые угрозы | Векторы атак | Основные меры смягчения |
|---|---|---|---|
| Build | • Инъекция команд • Заражённые базовые образы | • Небезопасные инструкции в Dockerfile • Отсутствие подписи образов | • Минимализация слоёв и использование «slim»/«alpine» образов • Подпись и проверка образов (Notary, Cosign) |
| Distribution | • MITM-атаки • Компрометация registry | • Незашифрованный трафик • Слабые учётные данные | • HTTPS/TLS для registry • Content Trust и проверка подписи образов |
| Run | • Контейнерный «escape» • Эскалация прав | • CAP_SYS_ADMIN, «–privileged» • Монтирование хоста | • User namespaces, drop all capabilities, seccomp-профили • Read-only файловые системы, ограничение ресурсов (cgroups) |

В настоящее время разработана систематическая модель угроз, в которой для каждой фазы жизненного цикла строятся «деревья атаки» (attack trees).

- Корень дерева: «компромисс контейнера».
- Уровни разбиения: стадии жизненного цикла (Build → Distribution → Run).

- Ветки: каналы эксплуатации (файловая система, сеть, привилегии).

Такое разбиение позволяет последовательно выявить промежуточные цели злоумышленника (поиск открытых портов, эскалация, эксфильтрация данных) и наглядно оценить комплекс необходимых мер защиты.

Для гибкого распределения усилий на смягчение угроз применяется DREAD — модель, учитывающая:

- Damage (урон),
- Reproducibility (воспроизводимость),
- Exploitability (эксплуатируемость),
- Affected users (число пострадавших),
- Discoverability (обнаруживаемость уязвимости).

Каждый фактор оценивается по шкале 1–10, а итоговый DREAD Risk равен отношению суммы его составляющих к их количеству: (D+R+E+A+D)/5. Угрозы с наивысшими баллами получают первоочерёдную защиту [1, 2].

Таким образом следует отметить, что контейнеризация усиливает инфраструктуру за счёт ОС-уровневой изоляции и гибкого resource-лимитирования, однако открывает новые точки входа для атак. Системный подход включает:

1. Фазовое моделирование угроз жизненного цикла (Build/Distribution/Run),
2. Attack trees для визуализации путей атаки,
3. Приоритизацию DREAD,
4. Интеграцию сканеров образов и анализ реальных эксплойтов.

Только сочетание этих элементов обеспечивает надёжную защиту контейнерной платформы от современных угроз.

## IV. Интеграция статического и динамического анализа и изоляция в Docker

Для обеспечения всесторонней защиты контейнерной инфраструктуры необходимо объединить углублённый статический анализ образов — основанный на формальных методах верификации — с продвинутым динамическим мониторингом в ходе выполнения, а также задействовать весь комплекс пространственных, ресурсных и сетевых механизмов изоляции, предоставляемых Docker. Такая многослойная модель обороны значительно уменьшает вероятность попадания уязвимостей в продуктивную среду и обеспечивает своевременное обнаружение и нейтрализацию попыток злоумышленников преодолеть границы контейнера.

В таблице 3 представлены ключевые характеристики ведущих инструментов для анализа угроз и усиления изоляции в Docker-средах.

Таблица III.
Сравнение статических и динамических фреймворков анализа контейнеров в Docker [4, 8, 9]

| Метод анализа | Пример инструмента | Фаза ЖЦ | Типы обнаруживаемых уязвимостей | Интеграция в CI/CD | Нагрузка на систему |
|---|---|---|---|---|---|
| Статический | Clair | build, dist | CVE в базовых слоях, небезопасные пакеты | GitLab CI, Jenkins | низкая |
| Статический | Trivy | build, dist | CVE, конфигурационные дефекты (CIS) | GitHub Actions | низкая |
| Статический | Docker Scout | build, dist | CVE, неиспользуемые зависимости | Docker Desktop, CLI | минимальная |
| Динамический | Falco (CNCF) | run | Аномальные syscalls, сетевые атаки | Kubernetes, Helm | средняя (eBPF-мониторинг) |
| Динамический | Sysdig Secure | run | Runtime CVE, подозрительные процессы | CI/CD + "canary-deploy" | средняя |

Для повышения стойкости контейнерной среды к методам статического и динамического анализа необходимо расширить штатный функционал Docker следующими мерами. Во-первых, реализовать маппинг пользователя root внутри контейнера на непривилегированный UID хоста, что полностью исключит возможность эскалации прав [1]. Во-вторых, убрать все стандартные Linux-capabilities и внедрить политику seccomp, допускающую лишь заранее определённый перечень системных вызовов (к примеру, запрет ptrace и mount). Далее корневую файловую систему контейнера следует монтировать в режиме «только для чтения», а подключение томов (volumes) — осуществлять исключительно по необходимости. Для предотвращения DoS-атак на уровне контейнера важно ограничивать использование CPU, оперативной памяти и операций ввода-вывода [1]. Кроме того, рекомендуется сегментировать сетевой трафик через Docker networks или CNI (в Kubernetes) с применением белых списков портов и IP-адресов и устанавливать взаимную аутентификацию по TLS между микросервисами.

Лишь комплексный подход, объединяющий статический анализ во время сборки образов (Clair, Trivy, Scout), динамический мониторинг в рантайме (Falco, Sysdig Secure) и жёсткую изоляцию с помощью user namespaces, seccomp, cgroups и сетевых политик, позволяет выстроить архитектуру, устойчивую к современным угрозам. Интеграция статических сканеров в CI/CD-пайплайн гарантирует «чистоту»

образов при каждом деплое, а механизмы рантайм-мониторинга и изоляции минимизируют риски эксплуатации ранее не выявленных уязвимостей.

## V. Алгоритм комплексной защиты пользовательских файлов в контейнеризованной среде

На основе проведенного анализа существующих подходов к валидации файлов и изоляции контейнеров, выявившего фрагментарность существующих решений, предлагается комплексный алгоритм защиты. Данный алгоритм систематизирует защитные меры в последовательную структуру, обеспечивая эшелонированную оборону (defence-in-depth) на протяжении всего жизненного цикла обработки пользовательского файла. В отличие от отдельных методик, сфокусированных либо на проверке данных таблицы 1, либо на изоляции среды (таблица 3), предлагаемый алгоритм объединяет эти уровни, что позволяет минимизировать поверхность атаки. Структура алгоритма включает следующие этапы.

На уровне HTML-формы задаётся атрибут accept в элементе <input type="file">, перечисляющий допустимые MIME-типы (например, image/jpeg, image/png, video/mp4 и др.). Это лишь удобство для пользователя: браузер скроет неподходящие файлы, однако зловредный код может умышленно обойти такие ограничения.

Перед приёмом файлов сервер проверяет, имеет ли запросивший пользователь необходимые права. Одновременно в форму внедряется скрытый крипто-токен для защиты от CSRF-атак; сервер валидирует его при получении POST-запроса, гарантируя, что загрузка инициирована легитимно. Nginx конфигурируется с ограничением числа одновременных запросов (rate limiting) и выставляет таймауты client_body_timeout и client_header_timeout, не позволяя «медленным» или массовым загрузкам исчерпать сетевые и системные ресурсы. Рекомендуется вынести временное хранилище PHP-загрузок на отдельный том.

Приём файла начинается с переименования: оригинальное имя заменяется на сгенерированную комбинацию метки времени и случайного числового идентификатора, что исключает коллизии и утечки информации об исходных именах. Файловый формат сверяется по расширению (приводится к нижнему регистру) и исследуется на наличие трюков с обратным порядком байтов и специально созданными последовательностями UTF-8, используемыми для обхода простых фильтров. Помимо MIME-типа, анализируются «магические числа» в начале файла, метаданные (гео-теги, цветовое пространство и т. д.), а также физические размеры изображения или видеокадра. Любое несоответствие заявленному формату приводит к отклонению загрузки.

В среде PHP-FPM отключаются функции exec, shell_exec, system, passthru, proc_open и др., чтобы исключить запуск произвольных команд. Все долгие или потенциально опасные операции передаются в PHP-CLI или обрабатываются через очереди (RabbitMQ, Redis и т. п.). Ресурсозатратные операции (перекодирование видео, сжатие и конвертация изображений) выносятся в бек-офис: HTTP-воркер отдаёт быстрый ответ, а CLI-скрипты, запущенные через cron или очередь, выполняют работу асинхронно и независимо от HTTP-контекста. Все постобработки происходят в контейнерах с минимальными привилегиями (--no-new-privileges), без режима --privileged. Применяются cgroups (лимиты CPU/памяти), remapping user-namespaces, ограничение сетевого доступа и фильтры системных вызовов через seccomp-профили. Дополнительно можно включить SELinux или AppArmor и хранить файлы в удалённом хранилище S3, выделяя отдельные зоны с повышенными рисками.

Предложенный алгоритм обладает рядом преимуществ по сравнению с разрозненными подходами, рассмотренными в литературном обзоре. В отличие от исследований, фокусирующихся исключительно на статическом сканировании образов, как у Shi H. et al. [5], или на runtime-анализе, как у Mahavaishnavi V., Saminathan R., Prithviraj R. [10], данный алгоритм интегрирует оба подхода (этапы 8 и 9) в единый пайплайн. Подходы, описанные Mubanda D. et al. [6], концентрируются на пентестинге уже запущенных систем, тогда как предлагаемая структура делает акцент на превентивных мерах на этапах валидации (этапы 3-5) и сборки контейнера. Таким образом, преимуществом является полнота (completeness) и непрерывность (continuity) контроля, что позволяет выявлять и блокировать угрозы на более ранних стадиях, снижая вероятность успешной эксплуатации уязвимостей, которые могли бы быть обнаружены только на этапе выполнения.

## VI. Заключение

В работе предложен и теоретически обоснован многоуровневый алгоритм защиты пользовательских файлов, интегрирующий бизнес-уровневую валидацию с инфраструктурной изоляцией в контейнерах Docker. Было показано, что синергия превентивного статического анализа на стадиях сборки и CI/CD (с использованием Clair, Trivy) и проактивного динамического мониторинга в runtime (Falco, Sysdig Secure) является основным элементом предложенного алгоритма для обеспечения непрерывного контроля безопасности. Результаты моделирования показывают, что именно комплексная природа предложенного алгоритма, охватывающая все фазы жизненного цикла файла и приложения, позволяет сократить риски успешных атак и гарантировать целостность, конфиденциальность и доступность пользовательских данных. В дальнейшем необходимо развивать автоматизированные механизмы генерации и анализа threat-моделей с использованием методов машинного обучения, а также исследовать особенности безопасности в serverless-средах и на периферии (edge computing).

## Библиография

[1] Aktolga İ. T. A study on analysis and detection of container escape vulnerabilities in Docker: дис. – Middle East Technical University (Turkey), 2024.

[2] VS D. P., Sethuraman S. C., Khan M. K. Container security: precaution levels, mitigation strategies, and research perspectives //Computers & Security. – 2023. – T. 135. – C. 103490.
[3] Borglund N. Security and Application Deployment using Docker. – 2024. – C. 15-35.
[4] Alyas T. et al. Container performance and vulnerability management for container security using docker engine //Security and Communication Networks. – 2022. – T. 2022. – №. 1. – C. 1-8.
[5] Shi H. et al. Dr. Docker: A Large-Scale Security Measurement of Docker Image Ecosystem //Proceedings of the ACM on Web Conference 2025. – 2025. – C. 2813-2823.
[6] Mubanda D. et al. Evaluating docker container security through penetration testing: a smart computer security //2023 International Conference on Communication, Security and Artificial Intelligence (ICCSAI). – IEEE, 2023. – C. 415-419.
[7] Rajyashree R. et al. An Empirical Investigation of Docker Sockets for Privilege Escalation and Defensive Strategies //Procedia Computer Science. – 2024. – T. 233. – C. 660-669.
[8] Sah K. P. et al. Advancing of Microservices Architecture with Dockers //2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT). – IEEE, 2024. – C. 1-6.
[9] Mubanda D. et al. Evaluating docker container security through penetration testing: a smart computer security //2023 International Conference on Communication, Security and Artificial Intelligence (ICCSAI). – IEEE, 2023. – C. 415-419.
[10] Mahavaishnavi V., Saminathan R., Prithviraj R. Secure container Orchestration: A framework for detecting and mitigating Orchestrator-level vulnerabilities //Multimedia Tools and Applications. – 2024. – C. 1-21.

# A Systematic Approach to User File Security: From Primary Validation to Isolation in Docker

O.V. Ekhlakov

***Abstract* - The article presents a description of a system approach to ensuring the security of user files in containerized environments. A system-integrated approach to protecting user files in containerized environments is studied. The paper proposes and substantiates a multi-level algorithm for ensuring the security of user files that integrates validation, threat modeling, and isolation procedures into a single pipeline. The algorithm described in the paper includes the stages of primary static and dynamic analysis of loaded information, preventive threat modeling of the Docker container life cycle using attack trees and the DREAD model, and the use of a set of isolation measures. Clair, Trivy, and Docker Scout are used for static scanning of images, which allows identifying vulnerabilities at the container layer level even before it is launched. In parallel, dynamic monitoring is organized using Falco and Sysdig Secure, which ensures continuous detection of anomalies and unsuspected exploits. Container isolation mechanisms include the use of user namespaces, seccomp profiles, and privilege and resource restrictions, which significantly reduces the possibility of privilege escalation. The presented results have practical value for researchers and information security architects working on formalizing heterogeneous trust models and developing multi-level mechanisms for validating user files, as well as for DevSecOps engineers and Docker platform administrators seeking to integrate dynamic isolation methods and advanced anomaly detection strategies into corporate CI/CD pipelines.***

***Keywords – containerization, file security, static analysis, dynamic monitoring, Docker, threat modeling, DREAD***

REFERENCES

[1] Aktolga İ. T. A study on analysis and detection of container escape vulnerabilities in Docker: dis. – Middle East Technical University (Turkey), 2024.

[2] V. S. D. P., Sethuraman S. C., Khan M. K. Container security: precaution levels, mitigation strategies, and research perspectives //Computers & Security, 2023, Vol. 135.

[3] Borglund N. Security and Application Deployment using Docker, 2024, pp. 15-35.

[4] Alyas T. et al. Container performance and vulnerability management for container security using docker engine //Security and Communication Networks, 2022, Vol.2022 (1), pp.1-8.

[5] Shi H. et al. Dr. Docker: A Large-Scale Security Measurement of Docker Image Ecosystem //Proceedings of the ACM on Web Conference 2025, 2025, pp. 2813-2823.

[6] Mubanda D. et al. Evaluating docker container security through penetration testing: a smart computer security //2023 International Conference on Communication, Security and Artificial Intelligence (ICCSAI). – IEEE, 2023, pp. 415-419.

[7] Rajyashree R. et al. An Empirical Investigation of Docker Sockets for Privilege Escalation and Defensive Strategies //Procedia Computer Science, 2024, Vol. 233, pp. 660-669.

[8] Sah K. P. et al. Advancing of Microservices Architecture with Dockers //2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT). – IEEE, 2024, pp. 1-6.

[9] Mubanda D. et al. Evaluating docker container security through penetration testing: a smart computer security //2023 International Conference on Communication, Security and Artificial Intelligence (ICCSAI). – IEEE, 2023, pp. 415-419.

[10] Mahavaishnavi V., Saminathan R., Prithviraj R. Secure container Orchestration: A framework for detecting and mitigating Orchestrator-level vulnerabilities // Multimedia Tools and Applications, 2024, pp1-21.