МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ

“СКИЛФЭКТОРИ”

SKILLFACTORY

Курс дополнительного профессионального образования

“Информационная безопасность”

Выпускная квалификационная работа

Метод поиска и устранения уязвимостей в открытом программном обеспечении для финансовой сферы

Выполнил:

Студент группы МИФИ-3

Ю.А. Воробьев

Москва

2024

Оглавление

Определения, обозначения и сокращения

В настоящем документе применяют следующие сокращения и обозначения, а также термины с соответствующими определениями:

ВПО	-	вредоносное программное обеспечение
ИТ	-	информационные технологии
Киберинцидент	-	действие, совершенное с использованием сетей компьютерных с целью действительного или потенциального нежелательного воздействия на систему информационную и/или содержащуюся в ней информацию
Контейнеризация	-	метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного
ОПО	-	открытое программное обеспечение
Парсинг	-	вычленение необходимой информации из файла по шаблону, что позволяет получить нужные данные в удобном формате
Пентестинг	-	метод проверки системы на уязвимости путем симуляции атак злоумышленников
ПО	-	программное обеспечение
Скоуп	-	весь объем работ, который нужно выполнить для достижения цели проекта

Фишинг	-	вид интернет-мошенничества, целью которого является получение доступа к конфиденциальным данным пользователей — логинам и паролям
API	-	Application Programming Interface, программный интерфейс - описание способов взаимодействия одной компьютерной программы с другой
CI/CD-пайплайн	-	(Continuous Integration and Continuous Delivery/Deployment) – это автоматизированная последовательность действий которая позволяет интегрировать, тестировать и доставлять обновления программного обеспечения с максимальной эффективностью
Django	-	свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC
Docker	-	программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации, контейнеризатор приложений
FastAPI	-	веб-фреймворк для создания API, написанный на Python
GET-запрос	-	запрос протокола HTTP к веб-серверу для получения нужной веб-клиенту информации

Flask	-	фреймворк для создания веб-приложений на языке программирования Python
Kubernetes	-	портативная расширяемая платформа с открытым исходным кодом для управления контейнеризованными рабочими нагрузками и сервисами
Payload	-	полезная нагрузка
POST-запрос	-	метод HTTP для отправки данных на сервер, один из самых распространенных методов HTTP
SIEM	-	Security Information and Event Management, система управления безопасностью
Open Source	-	программное обеспечение, которое поставляется для конечного пользователя с открытым исходным кодом
OWASP	-	Open Web Application Security Project, открытый проект обеспечения безопасности веб-приложений

Введение

В настоящей выпускной квалификационной работе рассматривается ОПО, а также методы поиска и устранения уязвимостей.

ОПО используется в различных сферах бизнеса, а также для государственных нужд. Данный вид программ предлагает множество преимуществ, таких как прозрачность, гибкость и снижение затрат. Однако использование ОПО также связано с определенными рисками, особенно в сфере безопасности. Финансовые приложения требуют высокой степени защиты, так как они работают с конфиденциальными данными и большими денежными средствами.

Актуальность выбранной темы обусловлена следующими факторами:

1) Рост мирового рынка услуг с открытым исходным кодом, который к 2027 году достигнет 54,1 млрд. долларов (среднегодовой темп роста 16,2%) [1] обеспечит смежный рост совокупности потенциальных угроз в сфере информационной и кибербезопасности ОПО, в первую очередь, в сфере финансов, поскольку финансовые институты активно ищут способы сократить издержки на обслуживание (использование ОПО обеспечивает экономию 10-15% на обслуживании) [2];

2) Необходимость изучения, разработки и актуализации методов поиска и устранения уязвимостей в открытом программном обеспечении исходит из анализа ключевых угроз информационной безопасности финансовой сферы: ВПО, фишинг, атаки на поставщиков ПО [3];

3) Геополитические риски, которые характерны для финансового рынка России, а также для рынков развивающихся стран. В 2023 году в России было зафиксировано 6,8 тыс. киберинцидентов в кредитно-финансовом секторе, а российские финорганизации в совокупности потратили более 200 млн. долларов. При этом 40% объявлений о нелегальных услугах в даркнете было связано с банками [4].

Цель данной работы заключается в разработке метода поиска и устранения уязвимостей в ОПО, ориентированного на финансовую сферу.

Результаты представлены в виде двух разделов. Первый раздел с теоретическими данными и анализом предметной области на основе открытых источников. Второй раздел - практическая реализация метода статического анализа в открытом программном обеспечении для финансовой сферы. Практическая часть направлена на повышение безопасности финансовых технологий путем выявления и устранения потенциальных угроз для финансовых данных и операций, обеспечивая надежность и защиту финансовых систем от кибератак и других угроз.

I. Аналитический обзор предметной области

1.1. Открытое программное обеспечение

Открытое программное обеспечение (англ. open source software) — это программное обеспечение с открытым исходным кодом. Исходный код создаваемых программ открыт, то есть доступен для просмотра и изменения. Это позволяет использовать уже созданный код для создания новых версий программ, для исправления ошибок и, возможно, помочь в доработке открытой программы [5]. Термин “открытый исходный код” возник в контексте разработки ПО для обозначения конкретного подхода к созданию компьютерных программ. Однако сегодня “открытый исходный код” обозначает более широкий набор ценностей и называется “путем с открытым исходным кодом”, который представляет собой набор принципов, вытекающих из моделей разработки ПО [6].

Идея сделать исходный код свободным возникла в 1983 году у идеологического движения, неофициально основанного Ричардом Столлманом, программистом из Массачусетского технологического института. Столлман считал, что программное обеспечение должно быть доступно программистам, чтобы они могли модифицировать его по своему желанию, с целью понять, изучить его и улучшить. Столлман начал выпускать свободный код под собственной лицензией, называемой GNU Public License. Этот новый подход и идеология создания программного обеспечения укрепились и в конечном итоге привели к формированию Инициативы открытого исходного кода в 1998 году. Инициатива открытого исходного кода (OSI) была создана для продвижения и защиты программного обеспечения с открытым исходным кодом и сообществ. OSI действует как центральный информационный и управляющий репозиторий программного обеспечения с открытым исходным кодом, предоставляет информацию о лицензировании кода, поддержку, определения и общее сотрудничество сообщества, чтобы

помочь сделать использование и обращение с открытым исходным кодом понятным и этичным [7].

Открытое программное обеспечение в соответствии со стандартами организации Open Source Initiative должно соответствовать ряду требований, представленных в Таблице 1.

Таблица 1

Перечень требований к определению соответствия программного обеспечения типу Open Source

№ п.п.	Наименование требования	Описание
1.	Свободное распространение	Лицензия на ПО не должна ограничивать продажу и распространение, а в случае неприкосновенности авторского исходного текста, производные программы и их исходные тексты должны свободно распространяться
2.	Доступность исходных текстов	ПО должно либо поставляться с исходными кодами, либо эти коды должны быть легко доступны
3.	Возможность модификации	Возможность читать исходные коды не должна позволять экспериментировать с ними и выпускать модификации
4.	Отсутствие дискриминации	ПО не может иметь дискриминирующий характер относительно людей или групп людей, а также по цели применения

		(свободная лицензия должна разрешать все виды деятельности)
5.	распространение лицензии	права, связанные с открытым ПО, должны быть применимы ко всем пользователям программы без заключения дополнительных соглашений, например, соглашения о неразглашении; лицензия не должна ограничивать другие программные продукты; лицензия не должна требовать что-либо от интерфейса или технологий, применяемых в производной программе; права на программный код не должны зависеть от того, является ли программа частью какого-то продукта. Человек, распространяющий программу в отрыве от сборника или перенесший часть кода в другой продукт, имеет такие же права, какие давал сборник

ПО с открытым исходным кодом, как правило, хранится в общедоступном репозитории. Любой может получить доступ к репозиторию, чтобы использовать код самостоятельно или внести улучшения в дизайн и функциональность всего проекта. Как и любое ПО распространяется по лицензии, которая включает условия определяющие, как разработчики могут использовать, изучать, изменять и, что наиболее важно, распространять

программное обеспечение. Наиболее популярными лицензиями на текущий момент являются:

1) Лицензия MIT;

2) Стандартная общественная лицензия GNU (GPL) 2.0 - она требует, чтобы копии измененного кода были доступны для публичного использования;

3) Стандартная общественная лицензия GNU (GPL) 3.0;

4) Лицензия BSD 2.0;

5) Apache License 2.0 [8].

Текущее состояние сферы ОПО характеризуется значительным ростом и развитием. В ИТ-индустрии использование и поддержка данного типа продуктов характерно для компаний, государственных организаций и отдельных разработчиков.

Ключевые аспекты текущего состояния сферы:

1) Широкое признание и принятие. Открытое ПО стало стандартом в ИТ-индустрии. Многие крупные компании, такие как Google, Microsoft, IBM и Facebook, активно используют и развивают открытые проекты. Microsoft, например, внесла значительные изменения в свою стратегию и стала одной из ведущих компаний по количеству вкладов в открытые проекты на GitHub [9];

2) Рост экосистемы и сообществ. Сообщества разработчиков вокруг открытого ПО продолжают развиваться. Платформы, такие как GitHub, GitLab и Bitbucket, способствуют сотрудничеству и совместной разработке проектов. Открытые форумы, конференции и встречи разработчиков помогают обмену знаниями и опытом, что ускоряет инновационное развитие и улучшает качество ПО;

3) Поддержка от правительств и организаций. Многие правительственные учреждения и международные организации поддерживают использование ОПО. Например, Европейская комиссия

активно продвигает использование ОПО в своих инициативах, таких как ISA [10];

4) Коммерциализация ОПО. Многие компании строят успешные бизнес-модели вокруг открытого ПО, предлагая коммерческую поддержку, консультации, интеграцию и дополнительные услуги. Примеры таких компаний включают Red Hat, которая была приобретена IBM, и Canonical, разработчик Ubuntu;

5) Инновации и технологии. Открытое ПО играет важную роль в развитии новых технологий, таких как облачные вычисления, контейнеризация, искусственный интеллект и машинное обучение. Проекты, такие как Kubernetes, TensorFlow и Apache Hadoop, являются лидерами в своих областях и активно развиваются сообществами;

6) Обеспечение безопасности. Безопасность остается критическим аспектом для ОПО. Разработчики и сообщества активно работают над улучшением безопасности, проводя регулярные аудиты, тесты и обновления. Инициативы, такие как Open Source Security Foundation (OpenSSF), направлены на улучшение безопасности открытого ПО.

1.2. Тренды развития открытого программного обеспечения

Разработка продуктов и решений с открытым исходным кодом – мировой тренд, который начался достаточно давно и до сих пор не теряет актуальности. Обороты и капитализация компаний, занимающихся созданием продуктов на ОПО, показывают опережающую динамику роста. Практически каждая организация, в том числе ИТ-корпорации, используют открытый код в своих проектах и сами становятся контрибьюторами ОПО [11].

Для развития ОПО характерны следующие тренды:

1) Контейнеризация и оркестрация: Проекты, такие как Docker и Kubernetes, стали стандартом для развертывания и управления приложениями в контейнерах;

2) Облачные технологии: Открытые проекты, такие как OpenStack, активно используются для создания и управления облачными инфраструктурами;

3) Искусственный интеллект: TensorFlow, PyTorch и другие открытые фреймворки для машинного обучения и искусственного интеллекта получили широкое распространение;

4) Разработка приложений: Фреймворки, такие как React, Angular и Vue.js, доминируют в сфере веб-разработки.

Стоит отметить, что российские программисты вносят большой вклад в развитие глобальных проектов Open Source – например, Arenadata – активный участник международных проектов: Greenplum, Apache Hadoop, Apache PXF, Apache Kafka, Apache NiFi, OpenSearch и PostgreSQL, а также высоко востребованных российских проектов: ClickHouse и Tarantool. Отечественные разработчики создают на базе ядра Linux вполне самодостаточные и конкурентоспособные операционные системы. Российские ИТ-компании, в том числе IT_ONE, активно включаются в развитие различных направлений ОПО, создавая курсы для разработчиков и применяя продукты на открытом коде в реализации проектов [11].

1.3. Открытое программное обеспечение в финансовой сфере

ПО для финансовой сферы имеет свои особенности и требования, связанные с высокой степенью ответственности, строгими регуляторными стандартами и критичностью данных. Ключевые особенности ПО для финансовой сферы представлены в Таблице 2.

Таблица 2

Ключевые особенности ПО для финансовой сферы

Группа особенностей	Детализация
Высокая степень безопасности	Шифрование данных: защита конфиденциальной информации клиентов и

	финансовых транзакций с использованием передовых методов шифрования; Аутентификация и авторизация: многофакторная аутентификация и строгие механизмы контроля доступа для предотвращения несанкционированного доступа; Мониторинг и аудит: постоянный мониторинг транзакций и систем, ведение логов и проведение регулярных аудитов для выявления и предотвращения мошенничества и других угроз;
Соответствие регуляторным требованиям	Регуляции и стандарты: соответствие локальным и международным стандартам и регуляциям, таким как PCI DSS (Payment Card Industry Data Security Standard) [3], GDPR (General Data Protection Regulation) и в России - Федеральный закон № 152-ФЗ “О персональных данных” [12]; Отчетность: возможность генерации подробных отчетов для регуляторов и внутренних проверок, включая финансовую отчетность и отчеты о безопасности;
Надежность и доступность	Высокая доступность: использование отказоустойчивых архитектур, резервирования и кластеризации для обеспечения непрерывной работы сервисов;

	Масштабируемость: способность системы обрабатывать большое количество транзакций и поддерживать рост количества пользователей и данных;
Производительность	Обработка в реальном времени: быстрая обработка транзакций и обеспечение минимальных задержек для клиентов; Оптимизация ресурсов: эффективное использование вычислительных ресурсов для поддержания высокой производительности при минимальных затратах;
Интеграция	Интерфейсы API: поддержка различных API для интеграции с внешними системами, такими как банковские шлюзы, платежные системы и другие финансовые институты, например, открытые API/Open Banking [13]; Совместимость: поддержка стандартов и протоколов для беспрепятственной интеграции с существующими системами и сервисами;
Управление рисками	Аналитика и прогнозирование: использование методов машинного обучения и аналитики для оценки и управления финансовыми рисками [14]; Контроль за соответствием: автоматизированные системы контроля за соблюдением политик и процедур, связанных с управлением рисками;

Пользовательский опыт	Удобство использования: разработка интуитивно понятных интерфейсов для клиентов и сотрудников финансовых учреждений; мобильные приложения: поддержка мобильных платформ для обеспечения удобного доступа к финансовым сервисам.

Примерами такого вида ПО могут послужить банковские системы и интернет-банкинг, например, клиент-банк ПАО “Сбербанк России” или платежные шлюзы по типу PayPal и Stripe.

Перечисленные выше особенности показывают, что к ПО в данной сфере необходимо адаптироваться к высоким требованиям безопасности, надежности и производительности, чтобы удовлетворять потребности клиентов и соответствовать регуляторным стандартам.

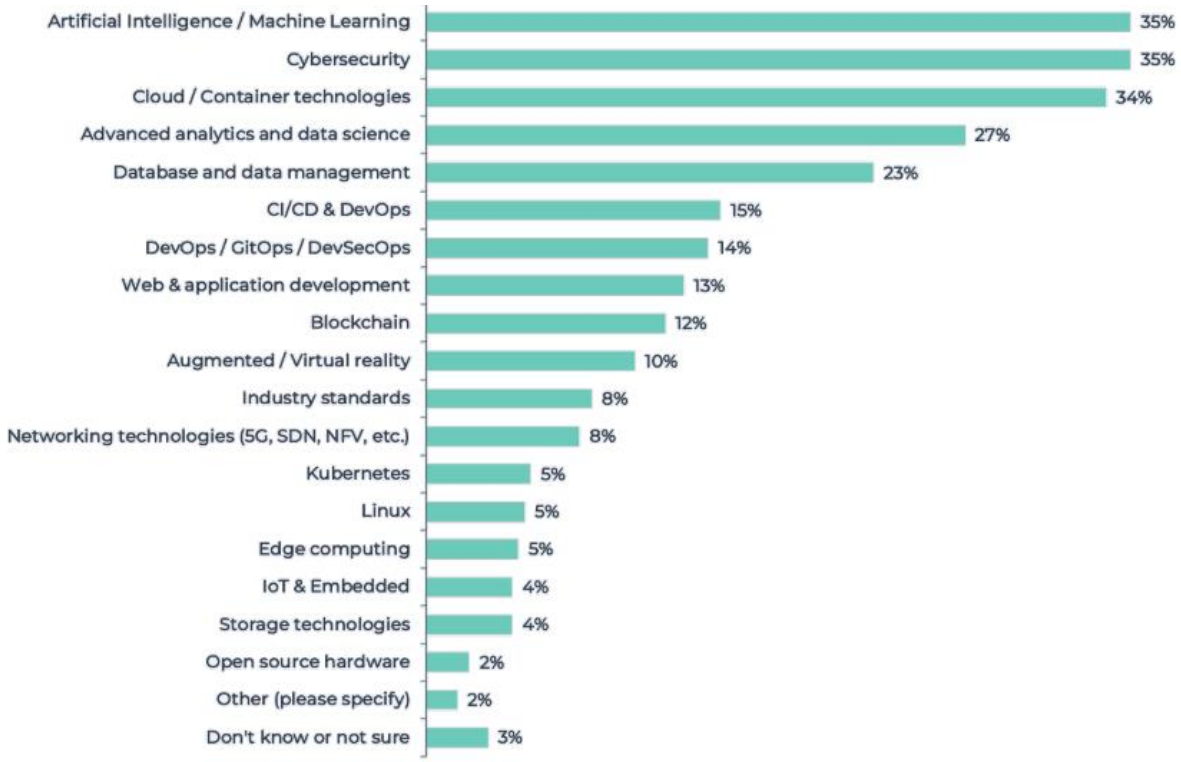
В сегменте финансовых услуг внедрение ПО с открытым исходным кодом становится эффективным способом быстрого запуска новых услуг на рынке [15]. Вместе с тем, на расширение использование ПО с открытым кодом оказывает тренд стремительного роста рынка приложений финтех-услуг - по оценкам Boston Consulting Group к 2030 году доля указанного рынка вырастет с 2% мирового дохода от финансовых услуг до 25%, что соответствует 1,5 трлн. Долларов [16]. Для создания таких приложений любой поставщик должен соответствовать строгим требованиям безопасности, конфиденциальности и доступности. При этом уровень доверия клиентов поставщикам финансовых услуг остается низким, например, более половины клиентов США и Великобритании не доверяют своему банку в вопросе защиты клиентских данных [17,18]. ПО с открытым исходным кодом часто может удовлетворить потребности в отношении доступности и надежности. Открытый исходный код доказал свою эффективность: такие сайты, как

Facebook, используют MySQL для управления данными, ежедневно обслуживая миллионы пользователей. Банки уже переводят некоторые из своих критически важных систем на использование вариантов с открытым исходным кодом, чтобы сократить расходы и улучшить опыт разработчиков для своих команд [15]. По данным проведенного опроса финансовых организаций компанией Fintech Open Source Foundation в 2023 году 94% организаций допускают определенный уровень использования открытого исходного кода, а 78% сообщают о возросшей ценности от использования открытого исходного кода по сравнению с 2022 годом (62%) [19]. На Рисунке 1 представлен рейтинг будущей ценности технологий сегмента финансовых услуг, созданных на основе открытого исходного кода.

На рынке действует немало игроков, которые в ряде направлений успешно конкурируют с поставщиками коммерческого ПО. Помимо Red Hat, это такие ориентированные на открытый софт организации, как Google, Apache Software, Novell, SUSE, Acquia и др. На рынке появляется все больше OpenSource аналогов существующих проприетартных решений. Сообщества разработчиков открытого ПО становятся сильнее и повышают качество и функциональность продуктов. Как правило, сама разработка предлагается бесплатно, а коммерциализируется она за счет поддержки и кастомизации. Это создает дополнительную конкуренцию и, как следствие, способствует развитию открытых программных продуктов.

Почти во всех отраслях и направлениях деятельности существует возможность использования ПО на основе открытого исходного кода. Почти у каждого проприетарного ПО имеется действующий аналог с открытым исходным кодом (Cisco VPN – OpenVPN, Oracle Database – PostgreSQL, Microsoft Hyper-V – OpenVZ, MS Active Directory – Samba, Avaya IP Office – Asterisk и т.д.) [5].

Рисунок 1. Рейтинг будущей ценности технологий сегмента финансовых услуг, созданных на основе открытого исходного кода



В России созданием среды Открытого банкинга занимаются Банк России и Ассоциация ФинТех. Совместно с ключевыми участниками финансовой отрасли разрабатываются стандарты взаимодействия, проводятся пилотные проекты для их тестирования, создается инфраструктурная основа доверенной среды Открытого банкинга:

сертификационный стенд, где участники проходят проверку на соответствие стандартам Открытых API,

информационный портал, объединяющий актуальную информацию и международный опыт в сфере Открытого банкинга,

удостоверяющий центр, гарантирующий безопасность обмена данными на уровне криптографической защиты [20].

Согласно Положению Банка России №683-П и №684-Па, все банки и некредитные финансовые организации должны обеспечить тестирование на

проникновение и анализ уязвимостей применительно к ПО, которое используется для оказания финансовых услуг: автоматизированная банковская система, интернет-банк, мобильный банк [21]. Помимо этого, в зоне ответственности IT-службы любой кредитно-финансовой организации - повышение надежности и доступности банковских систем при снижении стоимости владения средствами мониторинга, анализа, связи и инфраструктурных решений. Добиться этого можно тоже с помощью Open Source-продуктов (мониторинг IT-инфраструктуры - решения на основе Zabbix, Prometheus; мониторинг прикладных систем - ElasticSearch, Logstach, Kibana - связки продуктов, ставшей уже классическим решением; шины взаимодействия между Enterprise-системами - RabbitMQ, Apache Kafka; NoSQL базы данных и кэш - Redis, Apache Cassandra; контейнерные платформы - Red Hat OpenShift; СУБД - PostgreSQL, MySQL; операционные системы – Linux [22].

1.4. Риски использования ПО с открытым исходным кодом в финансовой сфере

5) Расширение практики использования открытого исходного кода для решений для финансовой сферы создает ряд рисков, связанных с целенаправленными атаками на цепочки поставок. Подобные риски реализуются путем сложных атак через внедрение вредоносного кода в компоненты ПО с открытым исходным кодом, что создает высокий уровень угрозы для финансовых организаций из-за взаимосвязи их систем и зависимости от цифровых платформ [23].

6) Ключевой риск - несанкционированный доступ. Злоумышленники, использующие уязвимости ОПО, могут взломать финансовые системы, что приведет к утечке данных или сбоям в обслуживании. Эти нарушения не только имеют финансовые последствия, но и подрывают доверие клиентов – критический риск в секторе, где доверие так важно [23]. Ключевыми проблемами в части поддержания доверия к

использованию программного обеспечения с открытым исходным кодом в цифровых активах финансового сектора является обеспечение безопасности и целостности ПО с открытым исходным кодом, которое все чаще используется в критически важных финансовых приложениях. Решающее значение при решении данной проблемы включает внедрение анализа состава программного обеспечения (SCA), а также составление спецификаций программного обеспечения (SBOM).

ОПО, играя важную роль в финансовой сфере, привносит свои преимущества и характеристики. Повсеместно используются такие продукты, как Apache Kafka, Kubernetes, Docker, Elasticsearch и т.д. Но это ПО, созданное крупными компаниями, зарекомендовавшими себя, нередки случаи использования ОПО в виде пакетов NPM или скачанных напрямую с сайтов разработчиков или хранилищ кода. Это несет дополнительные риски безопасности:

1) Открытый доступ к коду: хотя прозрачность и доступность исходного кода могут способствовать быстрому обнаружению и устранению уязвимостей, это также может предоставить злоумышленникам информацию о потенциальных слабых местах;

2) Недостаточное управление зависимостями: многие проекты ОПО используют сторонние библиотеки и зависимости. Незащищенные или устаревшие зависимости могут содержать известные уязвимости, что создает риск для финансовых систем. Например, уязвимость в библиотеке OpenSSL, известная как Heartbleed (CVE-2014-0160), привела к серьезным угрозам безопасности для множества финансовых организаций и сервисов [24].

3) Недостаточный мониторинг безопасности: некоторые организации могут не обеспечить должного уровня мониторинга и обновлений безопасности для используемого открытого ПО, что может привести к эксплуатации известных уязвимостей. Уязвимость в Apache Struts, известная

как Equifax breach (CVE-2017-5638), была использована для кражи личной информации более 147 миллионов клиентов Equifax [25].

Эти риски подчеркивают важность тщательного анализа и управления использованием открытого ПО в финансовой сфере, включая оценку безопасности, совместимости, лицензирования и поддержки.

1.5. Ключевые выводы по разделу

Открытое программное обеспечение (ОПО) стало неотъемлемой частью ИТ-индустрии, включая финансовую сферу, благодаря своей гибкости, экономичности и поддержке сообщества разработчиков. Текущее состояние сферы ОПО характеризуется широким признанием, ростом экосистемы, поддержкой со стороны правительств и коммерциализацией. Примеры таких успешных проектов включают Kubernetes, TensorFlow и Apache Hadoop.

Финансовое программное обеспечение, в свою очередь, имеет строгие требования по безопасности, соответствию регуляторным стандартам, надежности, производительности и интеграции. Использование ОПО в этой сфере предоставляет значительные преимущества, но также несет определенные риски. Прозрачность кода, недостаточное управление зависимостями и мониторинг безопасности могут привести к уязвимостям, как это было с уязвимостью Heartbleed в OpenSSL и Equifax breach в Apache Struts.

Следовательно, для успешного и безопасного применения ОПО в финансовой сфере необходимо тщательное управление рисками, включая регулярный анализ безопасности, управление зависимостями и поддержание актуальности используемых программных компонентов.

II. Поиск и устранение уязвимостей в открытом программном обеспечении для финансовой сферы

2.1. Средства и методы поиска и устранения уязвимостей

Для поиска и устранения уязвимостей требуется проводить ряд мероприятий, с целью полного покрытия продукта. Каждое мероприятие относится к одному из методов поиска и устранения уязвимостей (Таблица 3).

Таблица 3

Мероприятия по поиску и устранению уязвимостей в ОПО для финансовой сферы

№	Методы	Детализация
1	Статический анализ	Метод анализа исходного кода программного обеспечения без его выполнения. Он используется для выявления потенциальных ошибок, уязвимостей и проблем с качеством кода на ранних стадиях разработки. Статический анализ может проводиться вручную (рецензирование кода) или с помощью автоматизированных инструментов
2	Динамический анализ	Метод включает запуск приложения в контролируемой среде и наблюдение за его поведением в реальном времени. Динамический анализ позволяет выявлять уязвимости, которые могут проявиться только при выполнении программы, например, в случае некорректной обработки пользовательского ввода или недостаточной проверки данных

3	Пентестинг	Метод активного тестирования системы на уязвимости путем попыток взлома или эксплуатации ее слабых мест. Это может включать в себя различные виды атак, такие как SQL-инъекции, переполнение буфера, атаки на аутентификацию и другие. Пентестинг позволяет выявлять уязвимости, которые могут остаться незамеченными при других методах анализа
4	Аудит кода и ревизии	Метод включает тщательный анализ исходного кода программного обеспечения, проводимый специалистами по безопасности или разработчиками. В ходе аудита кода проверяется соответствие приложения стандартам безопасности, наличие потенциальных уязвимостей и соблюдение принципов безопасного программирования
5	Анализ зависимостей	Многие проекты программного обеспечения используют сторонние библиотеки и фреймворки. Анализ зависимостей позволяет выявлять уязвимости в этих зависимостях и принимать меры по их устранению или обновлению до более безопасных версий
6	Мониторинг безопасности	Постоянный мониторинг за безопасностью приложения и его окружения позволяет обнаруживать новые уязвимости, атаки и аномальное поведение в реальном времени.

		Это включает в себя мониторинг сетевой активности, журналов аудита, систем управления журналами событий (SIEM) и другие методы

Стоит отметить, что общие требования к методам и мероприятиям по поиску и устранению уязвимостей в ОПО для финансовой сферы закреплены в государственных стандартах (Таблица 4).

Таблица 4

Ключевые государственные стандарты Российской Федерации, связанные безопасностью ОПО в финансовой сфере

Номер	Наименование	Описание
ГОСТ Р 57580.1-2017[26]	Безопасность финансовых (банковских) операций. Защита информации финансовых организаций. Базовый состав организационных и технических мер	Стандарт определяет базовые требования к защите информации в финансовых организациях
ГОСТ Р 56939-2016[27]	Защита информации. Разработка безопасного программного обеспечения. Общие требования.	Стандарт описывает методы и средства обеспечения безопасности информационных систем, что актуально для ПО в финансовой сфере

ГОСТ Р ИСО/МЭК 27001-2006[28]	Информационная технология. Методы и средства обеспечения безопасности. Системы менеджмента информационной безопасности	Стандарт описывает требования к системам управления информационной безопасностью

2.2. Методология и инструментарий поиска и устранения уязвимостей в открытом программном обеспечении для финансовой сферы

В рамках рассматриваемой темы, в целях реализации практической части был выбран метод статического анализа.

Ключевые причины выбора метода статического анализа:

1) Раннее обнаружение уязвимостей - анализ проводится на этапе разработки, что позволяет выявлять уязвимости на ранних этапах жизненного цикла разработки ПО. Это значительно снижает стоимость и сложность устранения обнаруженных проблем, поскольку они исправляются до того, как попадут в рабочую среду;

2) Покрытие большего объема кода - анализ может охватывать весь объем исходного кода проекта, что позволяет выявить потенциальные уязвимости в каждой строке кода. Это обеспечивает более полное понимание безопасности приложения и возможность обнаружить скрытые уязвимости;

3) Автоматизация - статический анализ может быть полностью автоматизирован и интегрирован в процесс непрерывной интеграции и доставки. Это позволяет проводить анализ кода на регулярной основе и автоматически оповещать разработчиков о найденных проблемах;

4) Обнаружение потенциальных проблем без выполнения кода - анализ позволяет выявить широкий спектр потенциальных проблем без необходимости выполнения кода или проведения специальных тестов. Это

включает в себя обнаружение уязвимостей безопасности, неправильных практик программирования, потенциальных ошибок и других проблем;

5) Скорость и эффективность - статический анализ может быть проведен относительно быстро и эффективно, особенно с использованием специализированных инструментов, таких как SonarQube, Checkmarx, Fortify.

Для практической реализации анализа был использован инструмент SonarQube [29], который является одним из самых зрелых анализаторов и покрывающим большинство популярных языков программирования. Использование анализатора SonarQube удовлетворяет требованиям государственных стандартов Российской Федерации (Таблица 5).

Таблица 5

Соотнесение требований государственных стандартов Российской Федерации и возможностей инструментов SonarQube

Требование	Номер ГОСТ	Детализация
Безопасность кода	ГОСТ Р 57580.1-2017, ГОСТ Р 56939-2016	Инструмент помогает выявлять уязвимости и недостатки в безопасности кода и способствует обеспечению безопасности информационных систем
Качество кода	ГОСТ Р ИСО/МЭК 27001-2006	Инструмент способствует соблюдению требований к управлению качеством кода и информационной безопасностью
Управление зависимостями	ГОСТ Р 56939-2016	Анализ зависимостей в SonarQube помогает выявлять

		и устранять уязвимости в сторонних библиотеках
Соответствие стандартам кодирования	ГОСТ Р 57580.1-2017	Инструмент проверяет соответствие кода установленным стандартам и рекомендациям по безопасности
Мониторинг и отчетность	ГОСТ Р 57580.1-2017, ГОСТ Р 56939-2016	Инструмент обеспечивает автоматическое логирование и генерацию отчетов по безопасности

Статический анализ представляет собой предпочтительный метод благодаря своей способности обнаруживать уязвимости на ранних этапах разработки, покрывать большой объем кода, автоматизироваться, обнаруживать потенциальные проблемы без выполнения кода, а также благодаря скорости и эффективности.

Использование SonarQube, в частности, для обеспечения безопасности программного обеспечения в финансовой сфере Российской Федерации способствует выполнению требований ГОСТов.

Эти инструменты и методы помогают выявлять и устранять уязвимости, соответствовать стандартам защиты информации и поддерживать высокий уровень информационной безопасности. Разработка в рамках рассматриваемой темы посвящена статическому анализу и базируется на решении SonarQube.

2.3. Проектирование системы статического анализа

Проектирование системы статического анализа проведено с целью обозначения спецификации вариантов использования системы на основе предварительного анализа предмета исследования.

Для проектируемой системы типичен один тип субъекта (актора) - аналитик. Описание субъекта представлено в Таблице 6.

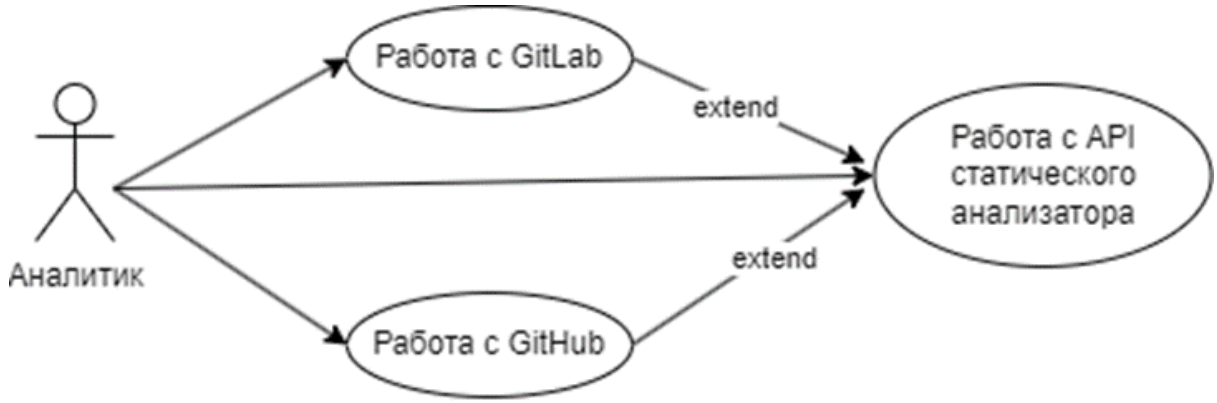
Таблица 6

Субъекты системы статического анализа

Критерий	Спецификация
Субъект (актор)	Аналитик
Действия субъекта	Запуск системы статического анализа с использованием API
Тип субъекта	Пользователь
Ответственности субъекта	Отсутствуют
Критерии успеха	Бесперебойное функционирование системы

В целях анализа вариантов использования системы, в том числе с целью их расширения были рассмотрены следующие варианты: GitLab-интеграция, GitHub-интеграция, использование статического анализатора посредством API (Рисунок 2).

Рисунок 2. Результирующая диаграмма вариантов использования системы



GitLab-интеграция позволяет субъекту выполнить CI/CD pipeline в системе. Прецедент начинается, когда пользователь запускает pipeline в GitLab. Анализ происходит в два этапа, на первом код переносится из репозитория на сервер с статическом анализатором, далее по API происходит создание проекта, генерация аналитического токена и статический анализ в SonarQube, после чего проект удаляется с сервера. На втором этапе происходит запрос данных по найденным проблемам безопасности, если такие есть, то система выдает пайплайну статус warning, а json-файл с ошибками оставляется в хранилище артефактов GitLab для скачивания пользователем, если ошибок не обнаружено, то pipeline завершается с кодом 0, что говорит об отсутствии ошибок. При этом необходимы: доступ к платформе GitLab, выделенный сервер с предустановленными SonarQube, SonarScanner, PostgreSQL и контроллер статического анализатора. По окончании потока событий с статусом warning будет доступен файл с списком замечаний безопасности, при любом из вариантов окончания в SonarQube будет создан проект с анализом.

GitHub-интеграция также позволяет субъекту выполнить CI/CD pipeline в системе. Прецедент начинается, когда пользователь запускает pipeline в рамках системы GitHub. Анализ происходит в два этапа, на первом код переносится из репозитория на сервер с статическом анализатором, далее по

API происходит создание проекта, генерация аналитического токена и статический анализ в SonarQube, после чего проект удаляется с сервера. На втором этапе происходит запрос данных по найденным проблемам безопасности, если такие есть, то система выдает аннотацию warning, а json-файл с ошибками оставляется в хранилище артефактов текущего пайплайна для скачивания пользователем, если ошибок не обнаружено, то pipeline завершается без аннотаций, а файл не создается. В этом варианте интеграции также необходимы: доступ к платформе GitHub, выделенный сервер с предустановленными SonarQube, SonarScanner, PostgreSQL и контроллер статического анализатора. При окончании потока событий с статусом warning будет доступен файл с списком замечаний безопасности, при любом из вариантов окончания в SonarQube будет создан проект с анализом.

Использование статического анализатора посредством API позволяет субъекту отправлять запросы к системе. Прецедент начинается, когда пользователь отправляет запрос к API контроллера и включает последовательность следующих событий: создание проекта, поиск проектов, создание токена анализа, запуск процесса анализа, поиск по замечаниям статического анализатора, получение списка всех замечаний безопасности по конкретному проекту. Для реализации инструмента по API необходимы: выделенный сервер с предустановленными SonarQube, SonarScanner и контроллер статического анализатора.

Каждый из вариантов использования системы статического анализатора предназначен для выполнения анализа посредством удаленных вызовов или CI/CD pipeline в системе, обеспечивая анализ кода на предмет уязвимостей и безопасности.

Все варианты использования включают несколько этапов выполнения анализа, включая перенос кода, запуск анализатора, обработку результатов и создание отчетов о найденных уязвимостях.

Специальные требования к вариантам использования указывают на необходимость доступа к соответствующей платформе (GitLab, GitHub), а также наличие выделенного сервера с установленными инструментами анализа кода (SonarQube, SonarScanner) и контроллером статического анализатора.

Таким образом, варианты использования предоставляют надежные инструменты для обеспечения безопасности и качества кода в процессе разработки и развертывания программного обеспечения.

2.4. Практическая реализация контроллера статистического анализа

В целях практической реализации контроллера статистического анализа был выполнен ряд подготовительных шагов.

Окружение

Для обеспечения работоспособности продукта требуется настроить окружение. Поскольку разработка имеет зависимости от статического анализатора и базы данных, для упрощения развертки был использован метод контейнеризации на базе Docker.

База данных

В целях создания хранилища данных с последующим наполнением его информацией статического анализатора была выбрана технология PostgreSQL.

Статический анализатор

В качестве статического анализатора был использован SonarQube.

Модуль HTTP Utils

Данный модуль отвечает за реализацию утилит для обработки HTTP-запросов и ответов на них. Основные функции включают отправку запросов к внешним серверам, обработку ошибок, чтение тела запроса и параметров запроса, а также отправку ответов.

Общий поток выполнения включает следующую последовательность: проверка авторизации и типа контента, формирование запроса с заголовками

и параметрами, отправка HTTP-запроса, обработка ответа и отправка результата.

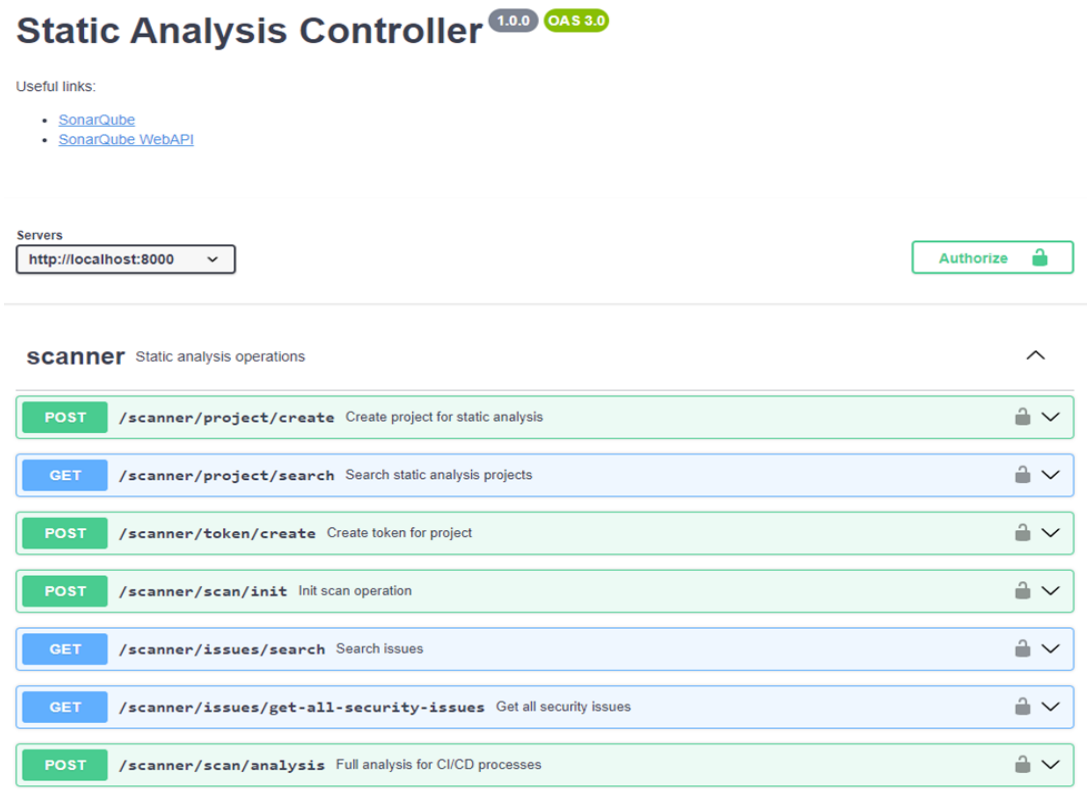
Модуль App

Внутренний модуль, предназначенный для управления запросами в приложении, связанными с документацией и статическими файлами. Выдача статики на примере документации представлена на Рисунке 3.

Общий поток выполнения:

1) Определение типа контента на основе расширения файла;
2) Инициализация файла по указанному пути;
3) Попытка загрузки и выполнить парсинг файла;
4) Отправка файла в ответ с необходимым типом контента;
5) Обработка ошибки, если файл не найден или возникла другая ошибка;
6) Обработка ответа и отправка результата.

Рисунок 4. Документация проекта в формате OpenAPI



Модуль Scanner

Основной модуль приложения. Он содержит несколько классов, каждый из которых отвечает за взаимодействие с различными аспектами SonarQube через HTTP-запросы.

Issues Controller отвечает за управление запросами, связанными с проблемами в SonarQube:

1) get_issues: получает список проблем на основе переданных параметров запроса (проверяет корректность параметров name, status, type, и page, формирует payload и отправляет GET-запрос к SonarQube API для поиска проблем);

2) get_all_security_issues: получает все проблемы безопасности для указанного компонента (проверяет наличие и корректность параметра name, формирует payload для запроса и отправляет его к SonarQube API, повторяет запрос до тех пор, пока не будут получены все проблемы безопасности, объединяя результаты в один список).

Project Controller управляет запросами, связанными с проектами в SonarQube:

1) create_project: создает новый проект (проверяет наличие и корректность параметров key, name, и mainBranch, формирует payload и отправляет POST-запрос к SonarQube API для создания проекта);

2) search_project: ищет проекты на основе переданных параметров запроса (проверяет корректность параметров query и page, формирует payload и отправляет GET-запрос к SonarQube API для поиска проектов).

Scan controller отвечает за инициацию и выполнение сканирования в SonarQube:

1) init_scan: инициализирует сканирование проекта (проверяет наличие и корректность параметров key, source, и token, формирует команду для выполнения сканера SonarQube и выполняет её);

2) full_analysis: полный анализ, включающий создание проекта, генерацию токена и запуск сканирования (проверяет наличие и корректность параметров key, name, mainBranch, и source, создает проект в SonarQube, генерирует токен для анализа проекта, выполняет команду для запуска сканера SonarQube);

3) scan_report: генерация отчета по проекту, в котором перечислены все проблемы, найденные статическим анализатором, указана критичность, путь до файла, строка, сообщение и теги. На рисунке 5 приведен пример отчета.

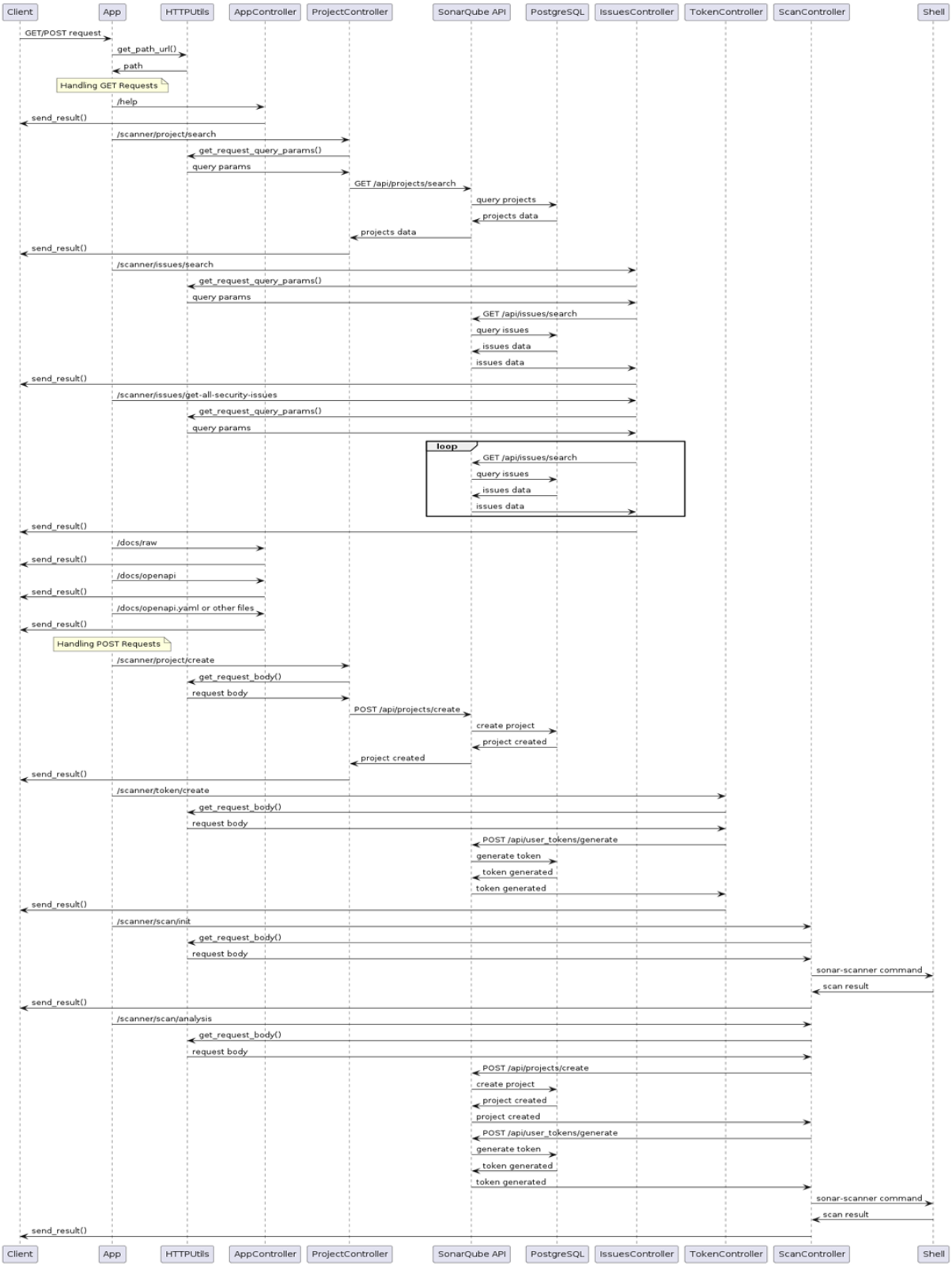
Рисунок 5. Пример сгенерированного отчета.

Security Issues Report for Project: test-1

Severity	Message	Line	Path	Tags
MINOR	Replace all tab characters in this file by sequences of white-spaces.	N/A	Backend/src/api/system/database/drivers/odbc/odbc_utility.php	convention,psr2
MINOR	Rename function "_backup" to match the regular expression ^[a-z][a-zA-Z0-9]*$.	57	Backend/src/api/system/database/drivers/odbc/odbc_utility.php	convention
MINOR	Rename class "CI _DB_odbc_utility" to match the regularexpression ^[A-Z][a-zA-Z0-9]* $.	49	Backend/src/api/system/database/drivers/odbc/odbc_utility.php	convention
MINOR	Write this "OR" keyword in lower case.	38	Backend/src/api/system/database/drivers/odbc/odbc_utility.php	convention,psr2
MAJOR	Remove the unused function parameter "$params".	57	Backend/src/api/system/database/drivers/odbc/odbc_utility.php	unused
MINOR	Replace "OR" with "||".	38	Backend/src/api/system/database/drivers/odbc/odbc_utility.php	suspicious

Token Controller управляет запросами, связанными с токенами в SonarQube create_analysis_token: создает токен для анализа проекта (проверяет наличие и корректность параметров key и name, формирует payload и отправляет POST-запрос к SonarQube API для генерации токена.

Рисунок 6. Общая схема сетевого взаимодействия.



Представленная разработка может улучшаться и развиваться.

При развитии проекта следует использовать фреймворки для Python по типу Flask, Django или FastAPI [30], это приведет к удешевлению разработки и добавления нового функционала, а также заберет ответственность с самоорганизации работы модуля HTTP Utils, что повысит надежность системы.

В данный момент вся инфраструктура имеет возможность запускаться из Docker, за исключением SonarScanner, который и проводит сканирование кода, требуется перевести его в контейнер и изменить часть кода, вызывающую данную утилиту. Это снимет ответственность с контроля совпадения версий SonarScanner и SonarQube и JRE, который необходим для работы SonarScanner, а также упростит контроль версий и обновления.

Репозиторий копируется на сервер целиком. Хоть сервер и содержит в себе только API-контроллер и инфраструктуру, описанную выше, репозиторий может включать вредоносные файлы. Перед отправкой на сервер следует проводить проверку антивирусными утилитами, например, ClamAV.

2.4. Практическая реализация проекта (индивидуальная часть)

В рамках ВКР и итоговой практической работы были реализованы следующие части:

1) Развертка удаленного сервера и настройка окружения;
2) Программный код для HTTPUtils, App, AppController, IssuesController, ProjectController, TokenController, частично ScanController;
3) Написание CI-конфигураций;
4) Составление Postman-коллекций;
5) Составление схем и UML-диаграмм.

С полным кодом проекта и вспомогательными материалами можно ознакомиться в репозитории [31].

Видео демонстрации доступны на YouTube:

1) CI-пайплайн в GitLab [32];
2) CI-пайплайн в GitHub [33];

3) Взаимодействие с API с помощью Postman [34];

Сервер и окружение

Для удаленного сервера был заказан облачный сервер на базе рег.ру со следующими характеристиками:

1) Операционная система: Ubuntu 24.04 LTS;
2) Количество ядер vCPU: 4;
3) Объем памяти RAM: 4 ГБ.

В рамках работы с сервером была проведена настройка SSH для CI-пайпланов, установлен и настроен Docker. На сервере были развернуты SonarQube, PostgreSQL, SonarScanner и разработанный API контроллер.

API контроллер

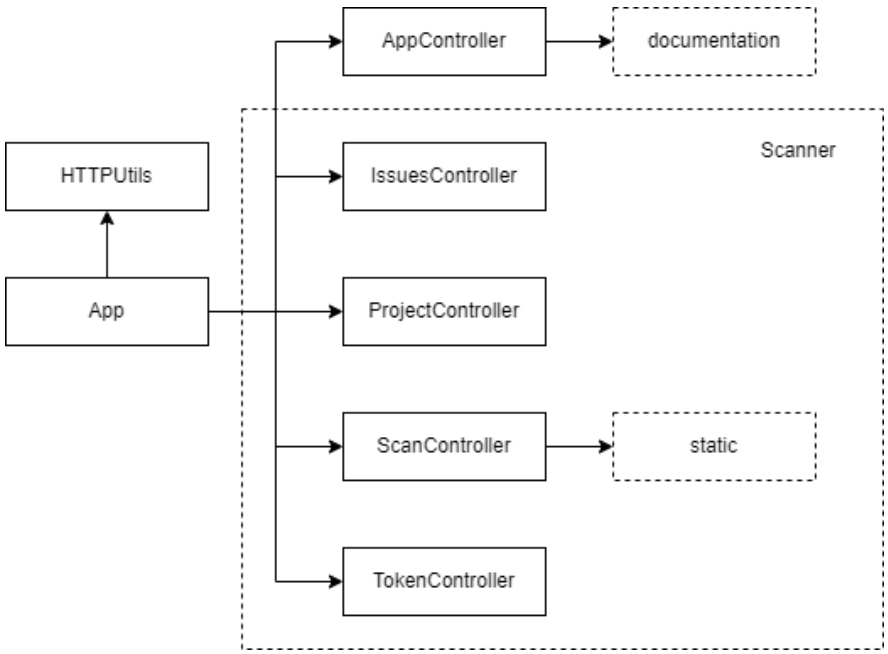
Для разработки было принято решение построить код на базе следующих библиотек:

1) requests – отправка запросов для связи с SonarQube;
2) python-dotenv – возможность работы с переменными окружения;
3) ruamel.yaml – библиотека для работы с yaml форматом, аналогично json;
4) sqlalchemy – работа с БД;
5) pg8000 – драйвер для работы с PostgreSQL;
6) pandas – для обработки результатов данных из БД;
7) reportlab – библиотека для формирования PDF-файлов.

Сервер осуществлен с помощью встроенного модуля http.server.

В основе архитектуры лежит паттерн MVC. Верхнеуровневая архитектура проекта представлена на рисунке 7. Основой является HTTP-сервер, принимающий входящие запросы и переводящий их на соответствующие внутренние контроллеры. Логически приложение разделено на три модуля, это App, HTTP Utils и Scanner со своими подмодулями, про каждый из которых описано выше.

Рисунок 7. Верхнеуровневая архитектура проекта.



CI-конфигурации

Для анализа кода, который потенциально будет обновляться было решено реализовать CI/CD-пайплайны. Для платформ хранения кода с встроенными механизмами пайплайнов были выбраны GitLab и GitHub, как одни из наиболее популярных решений. Диграмма пайплайна предствалена на рисунке 8.

Файлы конфигурации представляют из себя yaml файлы с заданными командами.

Набор команд:

1) Генерация случайного имени анализируемого проекта с помощью openssl;

2) Связь по SSH с удаленным сервером;

3) Создание и удаление директории с проектом для анализа и копирование кода из репозитория;

4) Запуск статического анализатора через API;

5) Получение данных по найденным инцидентам и извещение пользователя об этом.

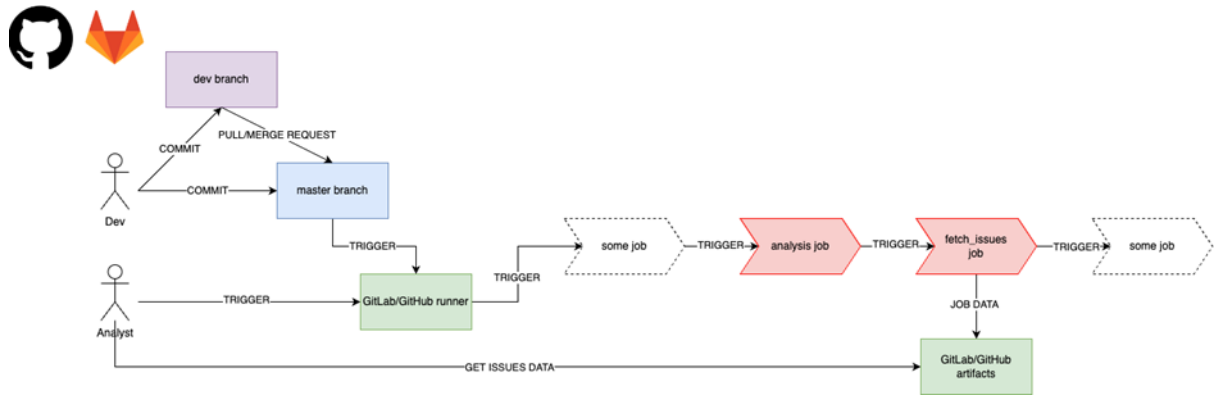
Для процесса пайплайна было принято решение разделить его на два этапа:

1) analysis – анализ проекта, команды 1 – 4;

2) fetch_issues – получение данных по инцидентам и сохранение информации в артефакты пайплайна с информированием пользователя (5 команда).

Дополнительные детали указаны в Приложении 1 данного документа.

Рисунок 8. Диаграмма пайплайна.

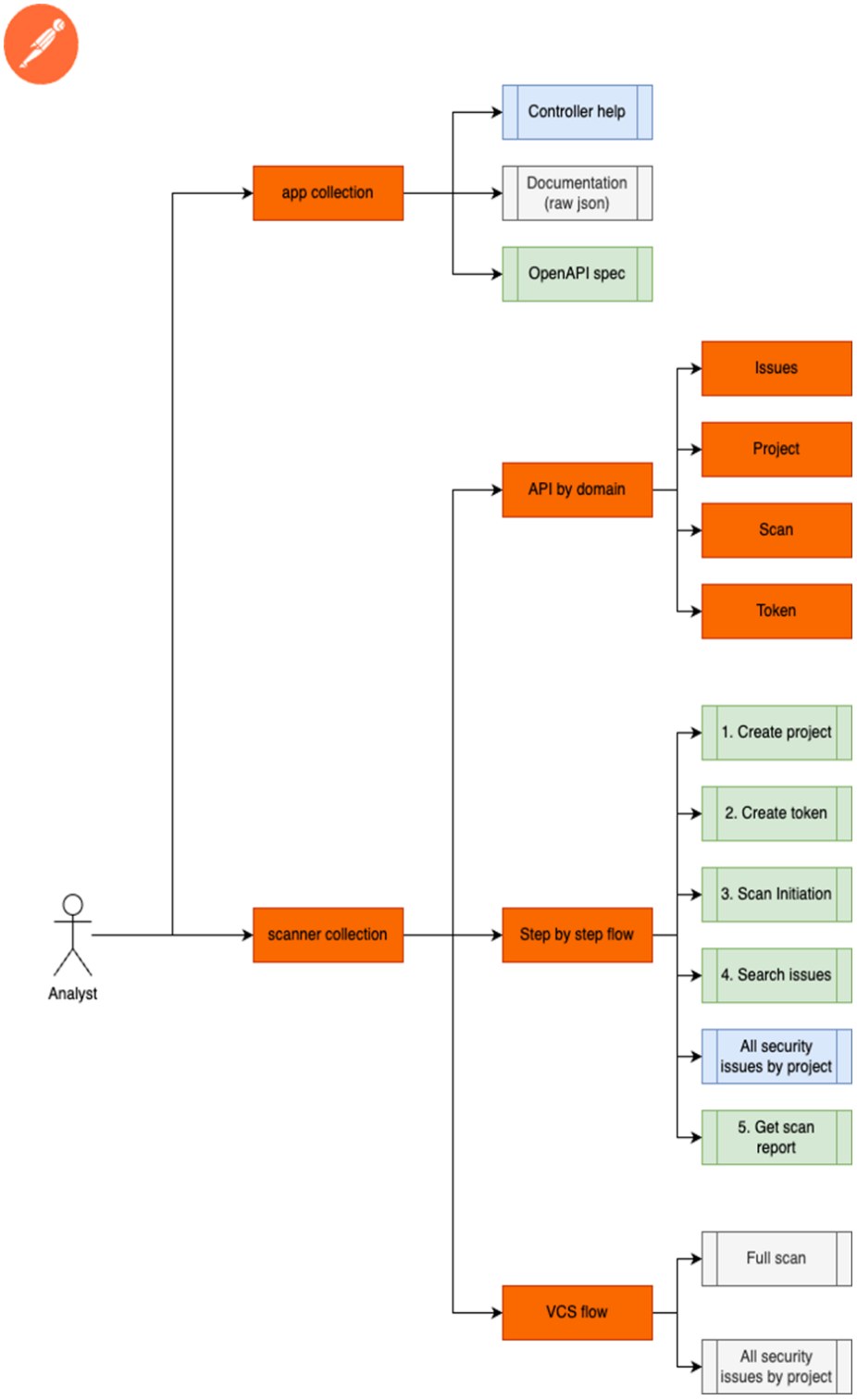


Postman-коллекции

Для работы с API не обязательно использовать CI-пайплайн. Запросы можно отправлять напрямую, с целью унификации и хранения запросов в одном месте было решено создать Postman-коллекцию. Плюсом данного решения является то, что при смене адреса или API-ключа, параметры можно изменить только в переменных окружения, что сразу обновит все запросы. Запросы в коллекции разделены на домены и функции, что отражено на рисунке 9.

Дополнительные детали указаны в Приложении 1 данного документа.

Рисунок 9. Структура Postman-коллекции



Схемы и UML-диаграммы

Схемы были выполнены в diagrams.net и PlantUML.

Заключение

В результате изучения предметной области и практической реализации метода поиска и устранения уязвимостей в ОПО сделаны следующие выводы.

Текущее состояние сферы ОПО характеризуется значительным ростом и развитием. В ИТ-индустрии использование и поддержка данного типа продуктов характерно для компаний, государственных организаций и отдельных разработчиков. Открытое ПО стало стандартом в ИТ-индустрии. Многие крупные компании, такие как Google, Microsoft, IBM и Facebook, активно используют и развивают открытые проекты.

ПО для финансовой сферы имеет свои особенности и требования, связанные с высокой степенью ответственности, строгими регуляторными стандартами и критичностью данных. В сегменте финансовых услуг внедрение ПО с открытым исходным кодом становится эффективным способом быстрого запуска новых услуг на рынке. Согласно Положению Банка России №683-П и №684-Па, все банки и некредитные финансовые организации должны обеспечить тестирование на проникновение и анализ уязвимостей применительно к ПО, которое используется для оказания финансовых услуг: автоматизированная банковская система, интернет-банк, мобильный банк.

Расширение практики использования открытого исходного кода для решений для финансовой сферы создает ряд рисков, связанных с целенаправленными атаками на цепочки поставок. Подобные риски реализуются путем сложных атак через внедрение вредоносного кода в компоненты ПО с открытым исходным кодом, что создает высокий уровень угрозы для финансовых организаций из-за взаимосвязи их систем и зависимости от цифровых платформ. Ключевой риск - несанкционированный доступ. Для успешного и безопасного применения ОПО в финансовой сфере необходимо тщательное управление рисками, включая регулярный анализ

безопасности, управление зависимостями и поддержание актуальности используемых программных компонентов.

В целях реализации практической части выпускной квалификационной работы был выполнен статический анализ ОПО UnSafe Bank. Было обнаружено пять уязвимостей различного уровня критичности, даны рекомендации по их устранению.

Список источников и литературы

1) Open Source Services Marker by Service [Электронный ресурс]. - Электрон. текстовые дан. - Режим доступа: URL: https://www.marketsandmarkets.com/Market-Reports/open-source-services-market-27852275.html (дата обращения 01.06.2024).

2) HOW MUCH DOES OPEN SOURCE COST? [Электронный ресурс]. - Электрон. текстовые дан. - Режим доступа: URL: https://www.siriusopensource.com/en-us/blog/how-much-does-open-source-cost (дата обращения 01.06.2024).

3) Защита данных в финансовой сфере [Электронный ресурс]. - Электрон. текстовые дан. - Режим доступа: URL: https://rt-solar.ru/products/solar_dozor/blog/3587/ (дата обращения 01.06.2024).

4) Отчет об атаках на финансовый сектор в 2023 году [Электронный ресурс]. - Электрон. текстовые дан. - Режим доступа: URL: https://rt-solar.ru/upload/iblock/a63/5b5veyxliulzjrd2eq5zo1kizwk6750f/Otchet-finsektor-2023_final.pdf (дата обращения 01.06.2024).

5) Открытое программное обеспечение [Электронный ресурс]. - Электрон. текстовые дан. - Режим доступа: URL: https://www.tadviser.ru/index.php/%D0%A1%D1%82%D0%B0%D1%82%D1%8C%D1%8F:%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_(Open_Source) (дата обращения 01.06.2024).

6) What is open source? [Электронный ресурс]. - Электрон. текстовые дан. - Режим доступа: URL:https://opensource.com/resources/ (дата обращения 01.06.2024).

7) What is the history of open source software? [Электронный ресурс]. - Электрон. текстовые дан. - Режим доступа: URL:https://www.synopsys.com/glossary/what-is-open-source-software.html (дата обращения 01.06.2024).

8) Get unparalleled insight into open source components [Электронный ресурс]. - Электрон. текстовые дан. - Режим доступа: URL:https://www.synopsys.com/software-integrity/software-composition-analysis-tools/knowledgebase.html (дата обращения 01.06.2024).

9) Microsoft Open Source [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://opensource.microsoft.com (дата обращения 25.05.2024).

10) Interoperability solutions for public administrations, businesses and citizens [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://ec.europa.eu/isa2/home_en (дата обращения 25.05.2024).

11) От Open Source к ОПО? Факторы развития свободного программного обеспечения в России [Электронный ресурс]. - Электрон. текстовые дан. - Режим доступа: URL:https://www.comnews.ru/digital-economy/content/217248/2021-11-08/2021-w45/open-source-k-opo-faktory-razvitiya-svobodnogo-programmnogo-obespecheniya-rossii (дата обращения 01.06.2024).

12) Федеральный закон "О персональных данных" от 27.07.2006 N 152-ФЗ (последняя редакция) [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://www.consultant.ru/document/cons_doc_LAW_61801 (дата обращения 26.05.2024).

13) КОНЦЕПЦИЯ ВНЕДРЕНИЯ ОТКРЫТЫХ API НА ФИНАНСОВОМ РЫНКЕ [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL:

https://www.cbr.ru/Content/Document/File/142114/concept_09-11-2022.pdf (дата обращения 26.05.2024).

14) The future of bank risk management [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://www.mckinsey.com/~/media/mckinsey/dotcom/client_service/risk/pdfs/the_future_of_bank_risk_management.pdf (дата обращения 26.05.2024).

15) Finance And Fintech: The Role Of Open Source In The Future Of Banking [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://www.forbes.com/(дата обращения 26.05.2024).

16) Global Fintech 2023: Reimagining the Future of Finance [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://www.bcg.com/publications/2023/future-of-fintech-and-banking (дата обращения 26.05.2024).

17) Financial Services Firms Need To Learn How To Earn Customers' Trust [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://www.forrester.com/blogs/ (дата обращения 26.05.2024).

18) Consumer trust is crucial for banks to undergo successful digital transformation [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://www.finextra.com/blogposting/22354/ (дата обращения 26.05.2024).

19) The 2023 State of Open Source in Financial Services [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://www.linuxfoundation.org/hubfs/LF%20Research/FINOS%20State%20of%20Open%20Source%20in%20Financial%20Services%202023%20-%20Report.pdf?hsLang=en (дата обращения 26.05.2024).

20) Открытый банкинг в России [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://openbankingrussia.ru/ (дата обращения 26.05.2024).

21) Анализ уязвимостей ПО в финансовых организациях [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://www.infosec.ru/glavnye-temy/analiz-uyazvimostey-po-v-finansovykh-organizatsiyakh/ (дата обращения 26.05.2024).

22) Свободное ПО в банках — persona grata [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL:https://bosfera.ru/bo/svobodnoe-po-v-bankah-persona-grata (дата обращения 26.05.2024).

23) Open-Source Software and the Financial Sector: Risks and Mitigation Measures [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL:https://www.europeanfinancialreview.com/ (дата обращения 26.05.2024).

24) Уязвимость Heartbleed (CVE-2014-0160) [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160 (дата обращения 26.05.2024).

25) Уязвимость Equifax breach (CVE-2017-5638) [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5638 (дата обращения 26.05.2024).

26) ГОСТ Р 57580.1-2017 [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://docs.cntd.ru/document/1200146534 (дата обращения 26.05.2024).

27) ГОСТ Р 56939-2016 [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://docs.cntd.ru/document/1200135525 (дата обращения 26.05.2024).

28) ГОСТ Р ИСО/МЭК 27001-2006 [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://docs.cntd.ru/document/1200058325 (дата обращения 26.05.2024).

29) clean code for teams and enterprises with {SonarQube} [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://www.sonarsource.com/products/sonarqube/ (дата обращения 26.05.2024).

30) Comparison of Flask, Django, and FastAPI: Advantages, Disadvantages, and Use Cases [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://medium.com/@tubelwj/comparison-of-flask-django-and-fastapi-advantages-disadvantages-and-use-cases-63e7c692382a (дата обращения 29.05.2024).

31) GitHub: static-analysis-controller [Электронный ресурс]. – Электрон. текстовые дан. – Режим доступа: URL: https://github.com/YuriyVorobyov96/static-analysis-controller (дата обращения 02.06.2024).

32) Static analysis controller - GitLab integration [Электронный ресурс]. – Видео – Режим доступа: URL: https://www.youtube.com/watch?v=4wlm6VMN29I (дата обращения 02.06.2024).

33) Static analysis controller – GitHub integration [Электронный ресурс]. – Видео – Режим доступа: URL: https://www.youtube.com/watch?v=nwe8hHR1Pr0 (дата обращения 02.06.2024).

34) Static analysis controller - step by step api calls [Электронный ресурс]. – Видео – Режим доступа: URL: https://www.youtube.com/watch?v=5zm-hJwtCpI (дата обращения 02.06.2024).

Приложение №1

к выпускной квалификационной работе

Отчет о проведенном статическом анализе открытого программного обеспечения для финансовой сферы

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

1) Выполнить анализ кода открытого ПО для финансовой сферы UnSafe Bank;

2) Выявить проблемы и уязвимости в ОПО. Присвоить уязвимостям уровни критичности;

3) Разработать рекомендации для устранения проблем и исключения факторов, которые могут спровоцировать возникновение инцидентов информационной безопасности.

1. МЕТОДОЛОГИЯ И ЦЕЛИ

1.1. Описание

Методология:

1) Выбор инструментов: Определение и установка инструментов для статического анализа кода, таких как SonarQube, Checkmarx, Fortify, или другие подходящие для анализа языков, использованных в проекте.

2) Сбор требований: Определение критичных компонентов системы, особенностей бизнеса и потенциальных угроз.

3) Настройка окружения: Подготовка среды анализа, включая репозитории кода и конфигурационные файлы.

4) Сканирование кода: Запуск статического анализа кода с использованием выбранных инструментов.

5) Анализ результатов: Разбор найденных уязвимостей, их классификация по уровням критичности (высокий, средний, низкий).

6) Отчетность: Формирование отчета, включающего найденные уязвимости, их описание, их расположение и рекомендации по устранению.

Цели:

1) Выявление уязвимостей: Обнаружение потенциальных слабых мест в коде, которые могут быть использованы злоумышленниками.

2) Повышение уровня безопасности: Устранение найденных уязвимостей для предотвращения возможных атак.

3) Соответствие стандартам: Обеспечение соответствия продукту стандартам безопасности и лучшим практикам индустрии.

4) Минимизация рисков: Снижение вероятности возникновения инцидентов безопасности, которые могут повлиять на пользователей.

1.2. ГОСТы

Список используемых для анализа государственных стандартов Российской Федерации представлен в Таблице 1.

Таблица 1

Номер	**Наименование**	**Описание**
ГОСТ Р 57580.1-2017	Безопасность финансовых (банковских) операций. Защита информации финансовых организаций. Базовые требования	Этот стандарт определяет базовые требования к защите информации в финансовых организациях.

ГОСТ Р 56939-2016	Информационные технологии. Методы и средства обеспечения безопасности информационных систем	Данный стандарт описывает методы и средства обеспечения безопасности информационных систем, что актуально для ПО в финансовой сфере.
ГОСТ Р ИСО/МЭК 27001-2006	Информационные технологии. Методы и средства обеспечения безопасности. Системы управления информационной безопасностью. Требования	Этот стандарт описывает требования к системам управления информационной безопасностью.

1.3. Статический анализ

Статический анализ — это метод анализа исходного кода программного обеспечения без его выполнения. Он используется для выявления потенциальных ошибок, уязвимостей и проблем с качеством кода на ранних стадиях разработки. Статический анализ может проводиться вручную (рецензирование кода) или с помощью автоматизированных инструментов.

В рамках анализа продукта решено уделить внимание статическому анализу, как предпочтительному способу поиска уязвимостей в ПО.

Ряд причин, почему выбран данный метод:

1) Раннее обнаружение уязвимостей - анализ проводится на этапе разработки, что позволяет выявлять уязвимости на ранних этапах жизненного цикла разработки ПО. Это значительно снижает стоимость и сложность

устранения обнаруженных проблем, поскольку они исправляются до того, как попадут в рабочую среду;

2) Покрытие большего объема кода - анализ может охватывать весь объем исходного кода проекта, что позволяет выявить потенциальные уязвимости в каждой строке кода. Это обеспечивает более полное понимание безопасности приложения и возможность обнаружить скрытые уязвимости;

3) Автоматизация - статический анализ может быть полностью автоматизирован и интегрирован в процесс непрерывной интеграции и доставки. Это позволяет проводить анализ кода на регулярной основе и автоматически оповещать разработчиков о найденных проблемах;

4) Обнаружение потенциальных проблем без выполнения кода - анализ позволяет выявить широкий спектр потенциальных проблем без необходимости выполнения кода или проведения специальных тестов. Это включает в себя обнаружение уязвимостей безопасности, неправильных практик программирования, потенциальных ошибок и других проблем;

5) Скорость и эффективность - статический анализ может быть проведен относительно быстро и эффективно, особенно с использованием специализированных инструментов, таких как SonarQube, Checkmarx, Fortify и другие.

1.4. Выбор инструментов

В качестве основы, поверх которой будет строится работы, был выбран SonarQube. На данный момент это один из самых зрелых анализаторов, который обладает покрытием большинства языков программирования. Также данный инструмент удовлетворяет требованиям ГОСТов:

1) Безопасность кода:
    a. ГОСТ Р 57580.1-2017: SonarQube помогает выявлять уязвимости и недостатки в безопасности кода, соответствуя требованиям ГОСТ по защите информации в финансовых организациях;

b. ГОСТ Р 56939-2016: Инструмент способствует обеспечению безопасности информационных систем;

2) Качество кода:
   a. ГОСТ Р ИСО/МЭК 27001-2006: SonarQube способствует соблюдению требований к управлению качеством кода и информационной безопасностью;
3) Управление зависимостями:
   a. ГОСТ Р 56939-2016: Анализ зависимостей в SonarQube помогает выявлять и устранять уязвимости в сторонних библиотеках, что соответствует требованиям ГОСТ;
4) Соответствие стандартам кодирования:
   a. ГОСТ Р 57580.1-2017: SonarQube проверяет соответствие кода установленным стандартам и рекомендациям по безопасности;
5) Мониторинг и отчетность:
   a. ГОСТ Р 57580.1-2017 и ГОСТ Р 56939-2016: Инструмент обеспечивает автоматическое логирование и генерацию отчетов по безопасности, что соответствует требованиям ГОСТ по мониторингу и отчетности.

2. СЦЕНАРИЙ И СКОУП

2.1. Сценарий

2.1.1. Подготовительный этап:

- Определение цели анализа и конкретных задач.
- Сбор информации о проекте, его архитектуре и используемых технологиях.
- Определение критических частей системы для приоритизации анализа.

2.1.2. Планирование и определение скоупа:

- Выбор конкретных модулей и компонентов для анализа.
- Учет требований безопасности и регуляторных норм.

- Определение критериев успеха и приемлемого уровня риска.

2.1.3. Проведение анализа:

- Выполнение статического анализа с использованием выбранных инструментов.
- Проведение анализа кода на уровне отдельных модулей и всего проекта в целом.

2.1.4. Пост-анализ:

- Обсуждение результатов анализа с командой разработки.
- Приоритизация найденных уязвимостей и планирование работ по их устранению.
- Повторное сканирование после исправления для проверки эффективности внесенных изменений.

2.2. Скоуп

2.2.1. Исходный код:

- Все компоненты приложения, включая backend и frontend.
- Сторонние библиотеки и зависимости.

2.2.2. Конфигурационные файлы:

- Настройки безопасности, конфигурации серверов, базы данных и других используемых сервисов.

2.2.3. Интеграции:

- Анализ взаимодействия с внешними сервисами и API.
- Проверка безопасности данных при передаче и хранении.

2.2.4. Тесты:

- Анализ тестов на наличие потенциала для утечек данных и других уязвимостей.

3. ИНФОРМАЦИЯ О ПРОЕКТЕ

3.1. Общая информация

Объектом анализа является open-source продукт UnSAFE_Bank [1], предоставляющий собой эмулятор банковского ОПО, в состав которого входят: Backend, Frontend, мобильные клиенты.

Код пакета UnSAFE_Bank доступен в репозитории GitHub.[2]

В ходе первичного анализа, был выделен следующий функционал ОПО UnSAFE_Bank:

- Перевод средств;
- Просмотр выписки по счету;
- Управление бенефициарами;
- Кредиты;
- Настройки счета;
- Форма обратной связи.

Необходимо:

- Провести анализ продукта в финансовой сфере и найти уязвимости перед его внедрением;
- Настроить CI/CD-пайплайны для GitLab CI и GitHub Actions;
- Предоставить возможность вызова API для роли аналитиков;
- Сделать генерацию отчетов с информацией об инцидентах для команды разработки.

3.2. Техническая информация

Используемые технологии:

1) База данных: MySQL;
2) Backend: PHP;
3) Frontend: React, Redux;

---

[1] UnSAFE Bank — это базовый виртуальный банковский пакет, разработанный с учетом рисков кибербезопасности и различных тестовых ситуаций.

[2] Ссылка на репозиторий: https://github.com/YuriyVorobyov96/UnSAFE_Bank

4) iOS-приложение: Swift.

Архитектура: приложение представляет собой классическую MVC-архитектуру.

4. АНАЛИЗАТОР

Для работы со статическим анализатором используется разработка static-analysis-controller. Это приложение, которое позволяет проводить статический анализ в SonarQube, расположенном на удаленном сервере. Связь с приложением осуществляется посредством API. В функционал приложения входит:

- Статический анализ кода через CI/CD-пайплайны в GitLab CI и GitHub Actions;
- Статический анализ через API;
- Получение информации по всем найденным проблемам;
- Получение информации только по проблемам безопасности;
- Генерация отчетов, включающая приоритет проблемы, расположение, комментарии и теги.

Код анализатора доступен в репозитории GitHub.[3]

4.1. Подготовка окружения

Для обеспечения работоспособности продукта требуется настроенное окружение. В данный момент разработка имеет зависимости от статического анализатора и базы данных. Для упрощения развертки рекомендуется использовать контейнеризацию на базе Docker.

4.1.1. База данных

Необходимо создать хранилище данных для наполнения его информацией статического анализатора.

В качестве базы данных была выбрана технология PostgreSQL. Для развертки надо выполнить docker-команду.

[3] Ссылка на репозиторий: https://github.com/YuriyVorobyov96/static-analysis-controller

4.1.2. Статический анализатор

В качестве статического анализатора используется SonarQube, для его использования необходимо выполнить docker-команду. Также необходимо установить SonarScanner.

4.1.3. Запуск окружения

```
docker pull sonarqube:latest

docker pull postgres:latest

docker pull sonarsource/sonar-scanner-cli:latest

 docker network create host-net

 mkdir sonarqube_data sonarqube_extensions sonarqube_logs postgresql postgresql_data

docker run --hostname=postgres --env=POSTGRES_USER=sonar --env=POSTGRES_PASSWORD=<POSTGRES_PASSWORD> --env=PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/lib/postgresql/16/bin --env=PGDATA=/var/lib/postgresql/data --volume=<POSTGRESQL_DIR_PATH>:/var/lib/postgresql --volume=<POSTGRESQL_DATA_DIR_PATH>:/var/lib/postgresql/data --volume=/var/lib/postgresql/data -p 5432:5432 --name postgres --network host-net -d postgres:latest

docker run --hostname=sonar --user=sonarqube --env=SONAR_JDBC_URL=jdbc:postgresql://postgres:5432/sonar --env=SONAR_JDBC_USERNAME=sonar --env=SONAR_JDBC_PASSWORD=<POSTGRES_PASSWORD> --env=PATH=/opt/java/openjdk/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin --env=JAVA_HOME=/opt/java/openjdk --env=LANG=en_US.UTF-8 --env=LANGUAGE=en_US:en --env=LC_ALL=en_US.UTF-8 --env=JAVA_VERSION=jdk-17.0.9+9 --env=DOCKER_RUNNING=true --env=SONARQUBE_HOME=/opt/sonarqube --env=SONAR_VERSION=10.3.0.82913 -
```

```
-env=SQ_DATA_DIR=/opt/sonarqube/data                                    --
env=SQ_EXTENSIONS_DIR=/opt/sonarqube/extensions                         --
env=SQ_LOGS_DIR=/opt/sonarqube/logs                                     --
env=SQ_TEMP_DIR=/opt/sonarqube/temp                                     --
volume=<SONARQUBE_DATA_DIR_PATH>:/opt/sonarqube/data                    --
volume=<SONARQUBE_EXTENSIONS_DIR_PATH>:/opt/sonarqube/extensions        --
volume=<SONARQUBE_LOGS_DIR_PATH>:/opt/sonarqube/logs                    --
workdir=/opt/sonarqube -p 9000:9000 --runtime=runc --name sonar --
network host-net -d sonarqube:latest
```

4.2. Работа с API-контроллером

4.2.1. CI/CD-пайплайн

Описание

Для работы необходимо выставить переменные окружений для CI:

Переменная	**Описание**
AUTH_TOKEN	Авторизационный токен для API-контроллера
REMOTE_SERVER	Адрес удаленного сервера
REMOTE_SERVER_API_ADDRESS	Адрес API-контроллера
REMOTE_SERVER_TMP_DIR	Директория на сервере для анализируемого проекта
REMOTE_USER	Конфигурации для подключения по SSH к удаленному серверу
SSH_PRIVATE_KEY	
SSH_KNOWN_HOSTS	

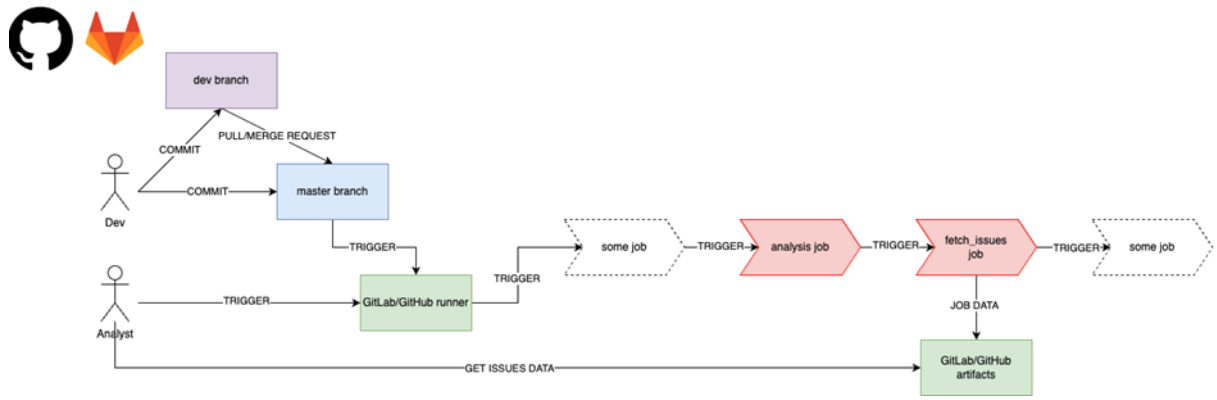
CI-конфигурации для платформ GitLab и GitHub доступны в репозитории.[4]

Далее требуется загрузить код в репозиторий, после этого начнется работа pipeline, либо можно внести новые данные в имеющийся репозиторий в нужную ветку, либо запустить pipeline собственноручно.

При возникновении ошибки pipeline завершится с ошибкой, о чем будет сообщено. Если pipeline выполнился успешно, то ошибок обнаружено не будет.

При окончании pipeline со статусом warning будет доступен файл со списком замечаний безопасности, при любом из вариантов окончания в SonarQube будет создан проект с анализом.

Рисунок 1. CI/CD-пайплайн-диаграмма



На диаграмме (Рисунок 1) указан поток для CI/CD-пайплайн:

- Разработчик провоцирует изменения в master-ветке посредством прямого ввода данных, либо через Pull/Merge Request, это инициирует запуск pipeline;
- Либо аналитик инициирует запуск pipeline в ручном режиме;
- Pipeline выполняет работу, обязательно выполняются два шага:
  - Анализ проекта;
  - Сбор и формирование информации об найденных инцидентах;

[4] Ссылка на репозиторий: https://github.com/YuriyVorobyov96/static-analysis-controller/tree/main/ci

- Найденные инциденты помещаются в хранилище артефактов;
- Продолжение pipeline продолжается, если это необходимо;

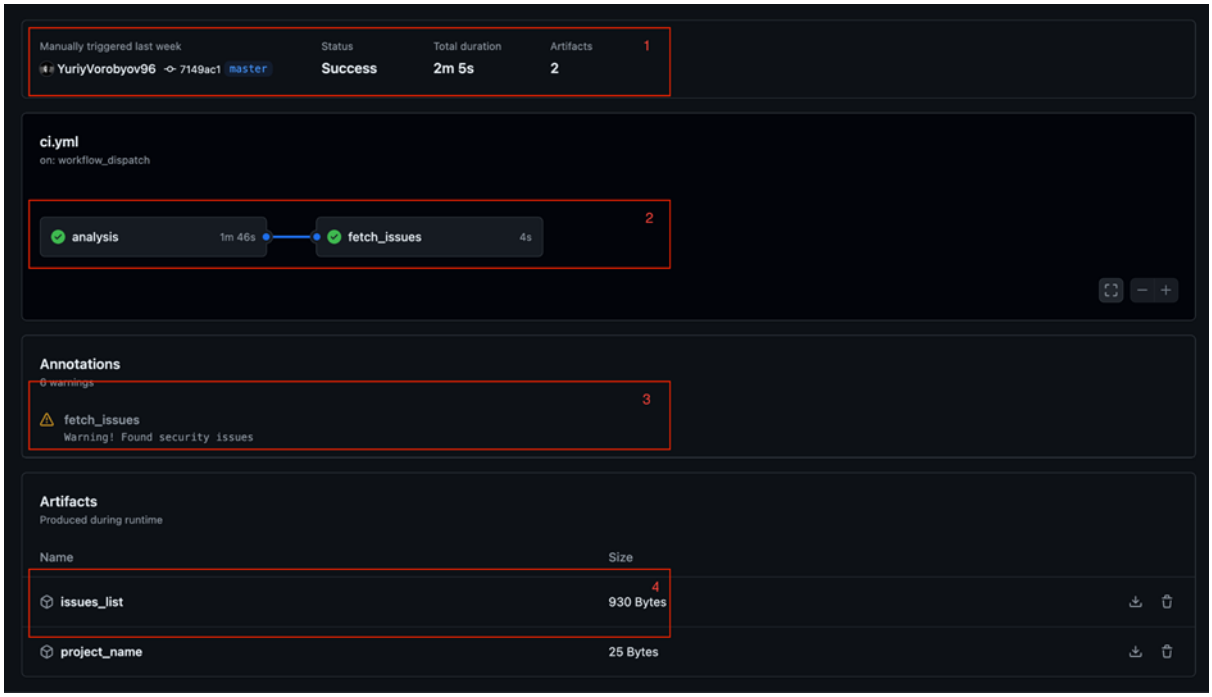
- Аналитик выгружает данные по инцидентам, если они есть.

Примеры работы CI/CD-pipeline

Пример работы pipeline в GitHub приведен на Рисунке 2. На примере видны следующие данные:

1. Статус выполнения пайплайна, число артефактов, время;
2. Этапы пайплайна, при клике можно посмотреть лог;
3. Предупреждение о том, что в коде обнаружены инциденты безопасности;
4. Артефакт с инцидентами безопасности.

Рисунок 2. Работа pipeline в GitHub



Пример работы pipeline в GitLub приведен на Рисунке 3. На примере видны следующие данные:

1. Статус выполнения pipeline,

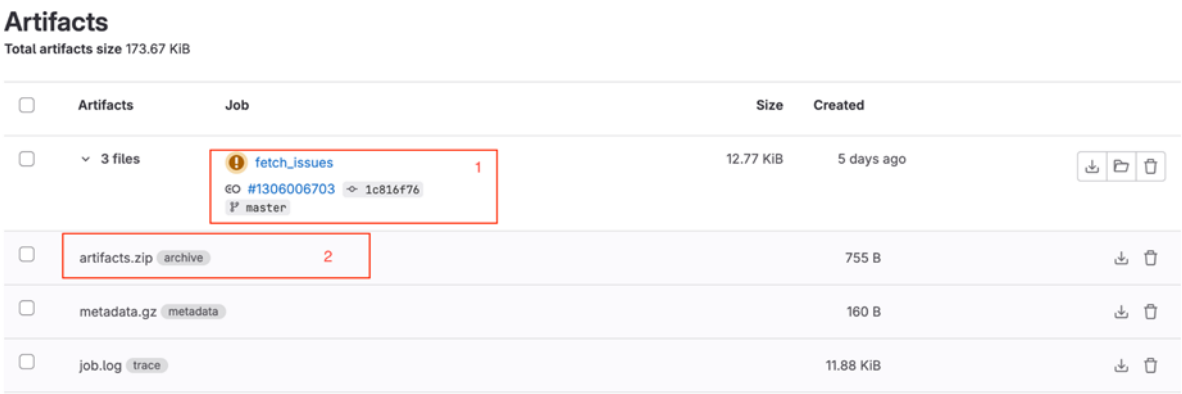
2. Этапы pipeline, с возможностью просмотра лога.

Рисунок 3. Работа pipeline в GitLub

В модуле Artifacts можно просматривать данные по артефактам (Рисунок 4). Основные поля с данными:

1. Принадлежность к pipeline;
2. Архив с артефактами инцидентов безопасности

Рисунок 4. Артефакты pipeline в GitLub



Пример артефакта с инцидентами безопасности (расширен комментариями):

```
[
  {
    "key": "a6a03c97-944e-42df-8acb-6dc326904165",
    "rule": "php:S2755",
    // Тяжесть (приоритет) инцидента
    "severity": "BLOCKER",
    // Полный путь до файла
    "component":
"4cb9ac2f33c23af43e416af4:tmp/4cb9ac2f33c23af43e416af4/Backen
d/src/api/application/controllers/ContactUs.php",
    // Проект
    "project": "4cb9ac2f33c23af43e416af4",
    "line": 52,
    "hash": "f7072de506872a70c9c119bf6bf1d418",
    // Расположение в файле
    "textRange": {
      "startLine": 52,
      "endLine": 52,
      "startOffset": 36,
      "endOffset": 65
    },
    "flows": [],
    "status": "OPEN",
    // Сообщение об ошибке
    "message": "Disable access to external entities in XML
parsing.",
    "effort": "15min",
    "debt": "15min",
    "author": "",
    "tags": [
      "cwe"
    ],
    "creationDate": "2024-05-26T17:06:01+0000",
    "updateDate": "2024-05-26T17:06:01+0000",
    "type": "VULNERABILITY",
    "scope": "MAIN",
    "quickFixAvailable": false,
    "messageFormattings": [],
    "codeVariants": [],
    "cleanCodeAttribute": "COMPLETE",
    "cleanCodeAttributeCategory": "INTENTIONAL",
    "impacts": [
      {
        "softwareQuality": "SECURITY",
        "severity": "HIGH"
      }
    ],
    "issueStatus": "OPEN"
  }
]
```

4.2.2. API-платформа

Для работы с API реализована Postman-коллекция, доступная в репозитории GitHub.[5]

Описание

Для работы были выставлены переменные окружений для API-запросов:

Переменная	Описание
host	Адрес API-контроллера
auth-token	Авторизационный токен для API-контроллера

Коллекции разделены на две папки:

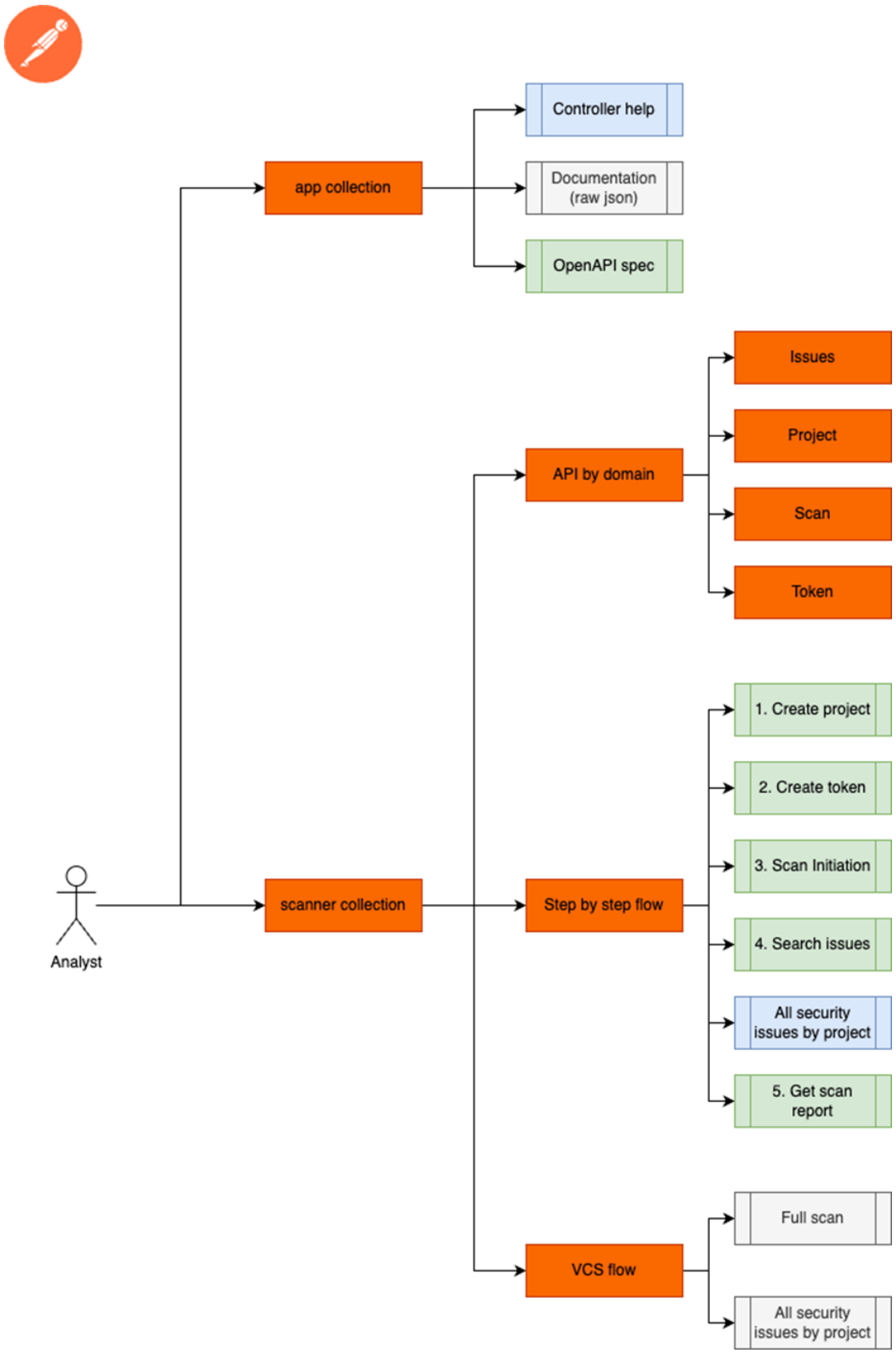
1. app - общие запросы к контроллеру, например, получения документации;
2. scanner - запросы на анализ к контроллеру.

На Рисунке 5 представлена API-диаграмма сценариев Postman, где блоки определены цветами:

- оранжевый - коллекции;
- синий - необязательные вспомогательные действия;
- серый - отладочные запросы;
- зеленый - аналитические сценарии.

[5] Ссылка на репозиторий: https://github.com/YuriyVorobyov96/static-analysis-controller/tree/main/postman

Рисунок 5. API-диаграмма сценариев Postman

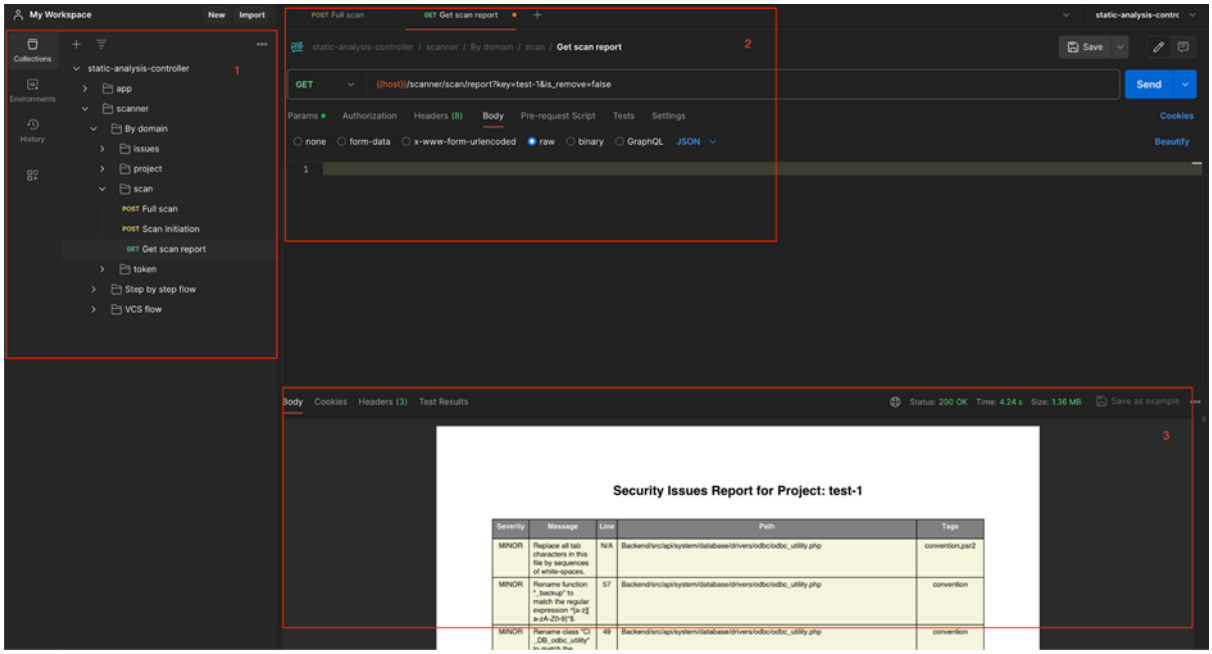


Пример работы с API-контроллером с помощью Postman представлен на Рисунке 6.

На рисунке выделены основные модули:

1. Структура коллекции;
2. Поле запроса с адресом и параметрами запроса (адрес, тело, заголовки и т.п);
3. Тело ответа.

Рисунок 6. Пример работы с API-контроллером с помощью Postman



5. ПОИСК ИНЦИДЕНТОВ БЕЗОПАСНОСТИ

В ходе анализа были выявлены уязвимости разной степени серьезности, в данном разделе будут рассмотрены основные группы.

5.1. Отладочное логирование

Проблема:

Отладочная логирование явно включено. Это может привести к раскрытию конфиденциальной информации и никогда не должно быть активировано в сборках релизов.

Критичность ВЫСОКАЯ

Расположение

Путь до файла: Backend/src/api/index.php

Пример кода

Рекомендации

Добавить логику проверки окружения, и включать debug логи только в тестовой и разработческой среде. Например, задавать через переменную окружения.

Вспомогательные ссылки

CWE-489: Active Debug Code.[6]

5.2. Использование MCRYPT

Проблема

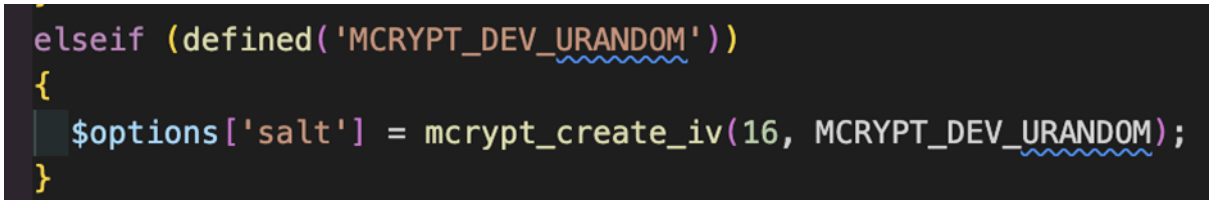
Шифрование на основе Mcrypt перешло в deprecated по причине отсутствия обновлений с 2007 года.

Критичность ВЫСОКАЯ

Расположение

Путь до файла: Backend/src/api/system/core/compat/password.php

Пример кода:



[6] https://cwe.mitre.org/data/definitions/489.html

Путь	Строки
Backend/src/api/system/core/compat/password.php	133 | $options['salt'] = mcrypt_create_iv(16, MCRYPT_DEV_URANDOM);
Backend/src/api/system/libraries/Encrypt.php	298 | $init_size = mcrypt_get_iv_size($this->_get_cipher(), $this->_get_mode()); 299 | $init_vect = mcrypt_create_iv($init_size, MCRYPT_DEV_URANDOM); 300 | return $this->_add_cipher_noise($init_vect.mcrypt_encrypt($this->_get_cipher(), $key, $data, $this->_get_mode(), $init_vect), $key); 315 | $init_size = mcrypt_get_iv_size($this->_get_cipher(), $this->_get_mode()); 325 \| return rtrim(mcrypt_decrypt($this->_get_cipher(), $key, $data, $this->_get_mode(), $init_vect),

Backend/src/api/system/libraries/Encryption.php	264 | mcrypt_module_close($this->_handle); 352 \| return mcrypt_create_iv($length, MCRYPT_DEV_URANDOM); 422 | mcrypt_module_close($params['handle']); 432 | $block_size = mcrypt_enc_get_block_size($params['handle']); 452 | mcrypt_generic_deinit($params['handle']); 455 \| mcrypt_module_close($params['handle']); 595 | mcrypt_module_close($params['handle']); 601 | $data = mdecrypt_generic($params['handle'], $data); 608 \| mcrypt_generic_deinit($params['handle']); 611 | mcrypt_module_close($params['handle']); 742 \| return mcrypt_module_open($cipher, '', $mode, '');

Рекомендации

Рассмотреть возможность использования Sodium или OpenSSL.

Вспомогательные ссылки

CVE-2012-4527;[7]

CVE-2012-4426;[8]

CVE-2012-4409.[9]

5.3. SQL-инъекции

Проблема

[7] https://nvd.nist.gov/vuln/detail/CVE-2012-4527

[8] https://nvd.nist.gov/vuln/detail/CVE-2012-4426

[9] https://nvd.nist.gov/vuln/detail/CVE-2012-4409

Пользовательские данные попадают в SQL-строку, которая создана вручную. SQL-строки, построенные вручную — это возможный индикатор SQL-инъекции, которая может позволить злоумышленнику украсть данные из базы данных или манипулировать ими.

Критичность ВЫСОКАЯ

Расположение

Путь до файла: Backend/src/api/application/models/Model_loan.php

Пример кода:

Путь	Строки
Backend/src/api/application/models/Model_loan.php	64 | $this->db->query("INSERT INTO loan_details (`user_id_fk`, `amount`, `roi`,`type`,`tenure`,`AppliedDate`) VALUES (?,?,?,?,?,?) ", array($accountId, $data['amount'], $data['roi'], $unserialised->logdata, $data['tenure'], date('Y-m-d')));
Backend/src/api/application/models/LoginModuleHandler.php	100 |$sql = "INSERT INTO session_master (session_id, last_access, cust_id) VALUES (?, ?, ?)"; 96 | $sql = "UPDATE session_master SET session_id = ? , last_access = ? WHERE cust_id = ?";
Backend/src/api/application/models/Model_beneficiary.php	247 | $stmt = $this->db->query("INSERT INTO beneficiary_details (user_id_fk, beneficiary_alias,

	beneficiary_account_no, bank_code) VALUES (?,?,?,?)",

Рекомендации

Пользовательские данные можно безопасно встраивать в SQL-строки с помощью подготовленных операторов или объектно-реляционного связующего (ORM). Вместо прямой вставки пользовательского ввода необходимо использовать подготовленные операторы ($mysqli->prepare("INSERT INTO test(id, label) VALUES (?, ?)");) или безопасную библиотеку.

Вспомогательные ссылки

A03:2021 – Injection[10].

5.4. Использование UNLINK

Проблема

Использование пользовательского ввода при удалении файлов с помощью unlink() может привести к тому, что злоумышленник может использовать эту уязвимость для изменения или доступа к файлам, на которые у него нет прав.

Критичность СРЕДНЯЯ

Рекомендации

Пользователи сервиса должны проходить аутентификацию с помощью имени пользователя и пароля. Для запоминания можно использовать сеанс PHP, и для хранения информации о владельце файла следует использовать таблицу базы данных или текстовый файл на сервере.

Вспомогательные ссылки

[10] https://owasp.org/Top10/A03_2021-Injection/

A01:2021 – Broken Access Control.[11]

5.5. Пользовательское имя файла

Проблема

Вызов unserialize() с использованием пользовательского ввода может привести к выполнению произвольного кода.

Критичность СРЕДНЯЯ

Расположение

Путь до файла: Backend/src/api/application/controllers/Loan.php

Пример кода:

```
if ($is_valid_param['status_code'] == "ALLOK1") {
    $unserialized = unserialize(base64_decode($parsed['data']['type']), ["LogWrite"]);
    $this->Model_loan->saveLoanDetails($account_id, $parsed['data'], $unserialized);
```

Рекомендации

Рассмотреть возможность использования JSON или структурированных данных.

Вспомогательные ссылки

A8:2017-Insecure Deserialization.[12]

5.6. Ложнопозитивные срабатывания

В ходе анализа были выявлены проблемы, которые инцидентами безопасности не являются, однако требуют внимания. К ним относятся:

1) использование exec - в исходном коде image_lib есть операции exec, но они не содержат пользовательский ввод и не могут быть скомпрометированы;

2) использование eval - в коде loader есть операция eval, но пользовательский ввод отсутствует;

3) использование openssl_decrypt без обработки false поведения - в исходном коде есть функция с данным поведением, но она не используется;

---

[11] https://owasp.org/Top10/A01_2021-Broken_Access_Control/

[12] https://owasp.org/www-project-top-ten/2017/A8_2017-Insecure_Deserialization.html

4) также для проблем пользовательского файла есть некоторые ложноположительные срабатывания, стоит обратить внимание только на тот файл и строку кода, которая указана выше.

ЗАКЛЮЧЕНИЕ

Команда выполнила анализ кода для ОПО UnSafe Bank. Были найдены уязвимости разных уровней критичности. Информация о причинах уязвимостей, их расположении в исходном коде, рекомендациям по исправлению были отражены в отчете.

Для дальнейшего анализа и получения упрощенных отчетов был реализован API-контроллер на базе SonarQube, который позволяет проводить быстрый анализ и получать всю необходимую информацию.

Были подготовлены CI-конфигурации, которые могут быть использованы в рамках систем GitLab и GitHub, данные конфигурации могут быть расширены или встроены в процесс CI/CD.

Для аналитиков подготовлены коллекции Postman для простого взаимодействия с API, также имеет OpenAPI спецификация с описанием действий и ссылками на оригинальный API SonarQube.

Поставленные задачи в техническом задании выполнены в полном объеме.