Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ

Разработка кода с точки зрения DevSecOps и кибербезопасности

Курсовой проект

МДК 01.01 «Разработка программных модулей»

Пояснительная записка курсового проекта по специальности  
09.02.07 Информационные системы и программирование

|  |  |
| --- | --- |
|  | Студент группы 1991  / М.Э. Нор /  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2024 г. |
|  | Оценка  Руководитель  / Л.Н. Цымбалюк /  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2024 г. |

# Содержание

[Содержание 2](#_Toc183050567)

[Введение 3](#_Toc183050568)

[1 Общая часть 6](#_Toc183050569)

[1.1 Постановка задачи 6](#_Toc183050570)

[1.2 Обоснование проектных решений 10](#_Toc183050571)

[2 Специальная, практическая, опытно-экспериментальная часть 14](#_Toc183050572)

[2.1 Алгоритм проведения анализа кода на безопасность 14](#_Toc183050573)

[2.2 Анализ предоставленного кода не безопасность 16](#_Toc183050574)

[Заключение 21](#_Toc183050575)

[Список литературы 22](#_Toc183050576)

[Приложение А 24](#_Toc183050577)

[(обязательное) 24](#_Toc183050578)

[Приложение Б 26](#_Toc183050579)

# Введение

В современном мире информационные технологии и программное обеспечение играют ключевую роль в функционировании бизнеса, государственных учреждений и повседневной жизни людей. С ростом зависимости от цифровых систем возрастает и уровень киберугроз, что делает вопросы кибербезопасности и защиты данных особенно актуальными. Одним из подходов, направленных на повышение безопасности программного обеспечения, является DevSecOps — методология, интегрирующая практики безопасности в процесс разработки и эксплуатации.

Актуальность темы обусловлена несколькими факторами. Во-первых, с каждым годом количество кибератак и утечек данных растет, что приводит к значительным финансовым и репутационным потерям для компаний. Во-вторых, традиционные подходы к обеспечению безопасности, такие как тестирование на поздних этапах разработки, не всегда эффективны и могут приводить к задержкам в выпуске продукта. DevSecOps предлагает интегрированный подход, который позволяет выявлять и устранять уязвимости на ранних стадиях, что значительно снижает риски и затраты.

Целью исследования является разработка методологии и рекомендаций по написанию безопасного кода с учетом принципов DevSecOps и кибербезопасности. В условиях постоянно меняющегося ландшафта угроз и уязвимостей, важно не только разрабатывать качественное программное обеспечение, но и обеспечивать его защиту на всех этапах жизненного цикла.

Целью данного исследовательского проекта является разработка методологии и рекомендаций по написанию безопасного кода с учетом принципов DevSecOps и кибербезопасности. В условиях постоянно меняющегося ландшафта угроз и уязвимостей, важно не только разрабатывать качественное программное обеспечение, но и обеспечивать его защиту на всех этапах жизненного цикла.

Задачи проекта включают:

* Изучение текущих уязвимостей в процессе разработки программного обеспечения:
  1. Анализ наиболее распространенных уязвимостей и угроз, с которыми сталкиваются разработчики;
  2. Исследование уязвимостей, которые возникают на различных этапах жизненного цикла разработки программного обеспечения, включая проектирование, кодирование, тестирование, развертывание и эксплуатацию;
  3. По возможности оценка уязвимостей, связанных с использованием сторонних библиотек и зависимостей.
  4. Исследование причин возникновения этих уязвимостей и методов их предотвращения;
  5. Анализ уязвимостей, возникающих из-за неправильной конфигурации систем и сервисов.
* Разработка методологии безопасного кодирования:
  1. Разработка методологии безопасного кодирования:Создание набора правил и рекомендаций для разработчиков, направленных на повышение безопасности кода.
  2. Включение принципов минимизации привилегий, проверки ввода, шифрования данных и других ключевых аспектов безопасного программирования.

Практическая значимость проекта заключается в создании комплексной методологии и набора рекомендаций для написания безопасного кода с учетом принципов DevSecOps и кибербезопасности. Разработанные правила и стандарты могут быть использованы разработчиками и специалистами по безопасности для повышения уровня защиты программного обеспечения на всех этапах его жизненного цикла.

Результаты проекта могут быть адаптированы для различных предприятий и организаций, занимающихся разработкой программного обеспечения, что позволит им интегрировать практики безопасности в свои процессы разработки и эксплуатации. Это не только повысит уровень защиты разрабатываемых приложений, но и снизит риски, связанные с кибератаками и утечками данных.

Кроме того, проект способствует оптимизации процессов управления безопасностью кода, упрощая внедрение и использование инструментов статического и динамического анализа, а также интеграцию этих инструментов в CI/CD пайплайны. Это позволяет выявлять и устранять уязвимости на ранних стадиях разработки, что значительно снижает затраты и усилия, связанные с исправлением уязвимостей на поздних этапах.

Таким образом, проект имеет значительный потенциал для коммерческого применения и может быть использован в различных отраслях, где безопасность программного обеспечения является критически важной.

# 1 Общая часть

# Постановка задачи

1.1.1 Обоснование необходимости разработки

В соответствии с заданием на работу по курсовому проектированию, требуется разработать методологию и набор рекомендаций для написания безопасного кода с учетом принципов DevSecOps и кибербезопасности. Актуальность данного проекта обусловлена растущими угрозами в области кибербезопасности и необходимостью обеспечения защиты программного обеспечения на всех этапах его жизненного цикла. В условиях современного рынка информационных технологий, внедрение практик безопасности в процесс разработки играет ключевую роль в обеспечении конкурентоспособности и защиты данных.

Целевая аудитория проекта включает:

* Разработчики программного обеспечения: Профессионалы, занимающиеся созданием и поддержкой программного обеспечения, которые стремятся интегрировать практики безопасности в свой рабочий процесс;
* Специалисты по кибербезопасности: Эксперты, ответственные за обеспечение безопасности информационных систем и данных, которые ищут эффективные методы и инструменты для защиты программного обеспечения;
* DevOps инженеры: Специалисты, занимающиеся автоматизацией процессов разработки и эксплуатации, которые стремятся интегрировать безопасность в свои CI/CD пайплайны;
* Менеджеры проектов и IT-руководители: Лица, принимающие решения, которые заинтересованы в повышении уровня безопасности разрабатываемых приложений и снижении рисков, связанных с кибератаками и утечками данных;
* Исследователи и аналитики: Профессионалы, занимающиеся изучением и анализом методов и инструментов обеспечения безопасности кода, которые могут использовать результаты исследования для дальнейших исследований и разработок.

Таким образом, проект направлен на создание комплексной системы, которая поможет различным специалистам в области IT и кибербезопасности повысить уровень защиты программного обеспечения и снизить риски, связанные с киберугрозами.

1.1.2 Технологии для обеспечения безопасности кода

Для разработки методологии и рекомендаций по написанию безопасного кода с учетом принципов DevSecOps и кибербезопасности будут применяться следующие технологии:

Технологии анализа и оценки безопасности:

* Инструмент для статического анализа кода SonarQube который позволит выявлять уязвимости на этапе написания кода;
* Инструмент для анализа зависимостей OWASP Dependency-Check, который помогает выявлять уязвимости в сторонних библиотеках и зависимостях;
* Инструменты для динамического анализа кода OWASP ZAP и Burp Suite Community Edition которые позволяют выявлять уязвимости в работающем приложении;
* OpenVAS: Бесплатный инструмент для проведения аудита безопасности и выявления уязвимостей в инфраструктуре.

Методы и практики безопасного кодирования:

* Принципы безопасного программирования: Минимизация привилегий, проверка ввода, шифрование данных, использование безопасных библиотек и фреймворков;
* Рекомендации OWASP: Использование рекомендаций и стандартов OWASP для обеспечения безопасности веб-приложений;

1.1.3 Методы для обеспечения безопасности кода

Для обеспечения безопасности программного обеспечения на всех этапах его жизненного цикла необходимо применять различные методы и практики безопасного кодирования. Вот подробное описание ключевых принципов и рекомендаций:

* Минимизация привилегий
  1. Принцип наименьших привилегий: Предоставлять пользователям и процессам только те привилегии, которые необходимы для выполнения их задач. Это снижает риск злоупотребления привилегиями в случае компрометации;
  2. Разделение привилегий: Разделение функциональности на несколько компонентов, каждый из которых имеет минимальные привилегии, необходимые для выполнения своей задачи.
* Проверка ввода
  1. Валидация данных: Проверка всех входных данных на соответствие ожидаемым форматам и значениям. Это включает проверку типов данных, длины строк, диапазонов значений и других параметров;
  2. Санитизация данных: Очистка входных данных от потенциально опасных символов и последовательностей, таких как SQL-инъекции, XSS (Cross-Site Scripting) и другие;
  3. Использование белых списков: Разрешение только определенных типов и значений данных, что снижает риск непредвиденных входных данных.
* Шифрование данных
  1. Шифрование данных в покое: Использование сильных алгоритмов шифрования для защиты данных, хранящихся на серверах и в базах данных;
  2. Шифрование данных в движении: Использование протоколов, таких как TLS/SSL, для защиты данных при передаче по сети;
  3. Управление ключами: Безопасное хранение и управление криптографическими ключами, используемыми для шифрования и дешифрования данных.
* Использование безопасных библиотек и фреймворков
  1. Выбор проверенных библиотек: Использование широко известных и проверенных библиотек и фреймворков, которые регулярно обновляются и поддерживаются;
  2. Обновление зависимостей: Регулярное обновление всех используемых библиотек и зависимостей для устранения известных уязвимостей;
  3. Анализ зависимостей: Использование инструментов для анализа зависимостей, таких как OWASP Dependency-Check, для выявления уязвимостей в сторонних библиотеках.
* Логирование и мониторинг
  1. Централизованное логирование: Сбор и хранение логов в централизованной системе для анализа и мониторинга;
  2. Мониторинг событий безопасности: Настройка систем мониторинга для отслеживания подозрительной активности и инцидентов безопасности;
  3. Анализ логов: Регулярный анализ логов для выявления аномалий и потенциальных угроз.
* Обработка ошибок и исключений
  1. Безопасная обработка ошибок: Обеспечение того, чтобы ошибки и исключения не раскрывали чувствительную информацию;
  2. Логирование ошибок: Запись информации об ошибках и исключениях в лог-файлы для последующего анализ;
  3. Уведомление пользователей: Предоставление пользователям понятных и безопасных сообщений об ошибках без раскрытия технических деталей.
* Рекомендации OWASP
  1. OWASP Top 10: Использование рекомендаций OWASP Top 10 для защиты веб-приложений от наиболее распространенных уязвимостей;
  2. OWASP ASVS (Application Security Verification Standard): Применение стандартов OWASP ASVS для проверки и верификации безопасности приложений;
  3. OWASP Cheat Sheets: Использование шпаргалок OWASP для быстрого доступа к рекомендациям по безопасности.

# Обоснование проектных решений

1.2.1 Обоснование выбора технологий для обеспечения безопасности кода

* SonarQube:
  1. Обоснование: SonarQube является одним из ведущих инструментов для статического анализа кода, который позволяет выявлять уязвимости на этапе написания кода. Это позволяет разработчикам оперативно исправлять ошибки и уязвимости, что значительно снижает риск возникновения проблем на более поздних этапах разработки;
  2. Преимущества: Интеграция с популярными системами управления версиями (например, Git), поддержка множества языков программирования, возможность автоматизации.
* OWASP Dependency-Check:
  1. Обоснование: OWASP Dependency-Check помогает выявлять уязвимости в сторонних библиотеках и зависимостях, что является критически важным для обеспечения безопасности, так как уязвимости в зависимостях могут быть использованы злоумышленниками для атак;
  2. Преимущества: Автоматическое выявление уязвимостей, возможность интеграции в CI/CD конвейеры.
* OWASP ZAP и Burp Suite Community Edition:
  1. Обоснование: Эти инструменты для динамического анализа кода позволяют выявлять уязвимости в работающем приложении, что важно для проверки безопасности в реальных условиях эксплуатации;
  2. Преимущества: Возможность тестирования веб-приложений на предмет различных типов уязвимостей, таких как SQL-инъекции, XSS и другие.
* OpenVAS:
  1. Обоснование: OpenVAS является бесплатным инструментом для проведения аудита безопасности и выявления уязвимостей в инфраструктуре, что позволяет комплексно подходить к обеспечению безопасности;
  2. Преимущества: Широкий спектр поддерживаемых уязвимостей, возможность автоматизации сканирования.

1.2.2 Обоснование выбора методов для обеспечения безопасности программного обеспечения

* Минимизация привилегий
  1. Обоснование: Принцип наименьших привилегий и разделение привилегий являются фундаментальными методами обеспечения безопасности. Предоставление минимально необходимых привилегий пользователям и процессам снижает риск злоупотребления привилегиями в случае компрометации. Разделение функциональности на несколько компонентов с минимальными привилегиями уменьшает поверхность атаки и затрудняет злоумышленникам доступ к критическим ресурсам;
  2. Преимущества: Снижение риска несанкционированного доступа, уменьшение поверхности атаки, повышение общей безопасности системы.
* Проверка ввода
  1. Обоснование: Валидация и санитизация данных являются критически важными методами для предотвращения атак, таких как SQL-инъекции, XSS (Cross-Site Scripting) и другие. Проверка всех входных данных на соответствие ожидаемым форматам и значениям, а также использование белых списков, помогает предотвратить непредвиденные входные данные и потенциально опасные символы;
  2. Преимущества: Снижение риска инъекционных атак, повышение надежности и безопасности приложения, улучшение качества данных.
* Шифрование данных
  1. Обоснование: Шифрование данных в покое и в движении является необходимым методом для защиты конфиденциальной информации. Использование сильных алгоритмов шифрования и протоколов, таких как TLS/SSL, обеспечивает защиту данных от несанкционированного доступа и перехвата. Безопасное управление криптографическими ключами гарантирует, что только авторизованные пользователи могут получить доступ к зашифрованным данным;
  2. Преимущества: Защита конфиденциальной информации, предотвращение несанкционированного доступа, соответствие стандартам безопасности.
* Использование безопасных библиотек и фреймворков
  1. Обоснование: Выбор проверенных и широко известных библиотек и фреймворков, которые регулярно обновляются и поддерживаются, снижает риск использования уязвимого кода. Регулярное обновление зависимостей и использование инструментов для анализа зависимостей, таких как OWASP Dependency-Check, помогает своевременно обнаруживать и устранять уязвимости;
  2. Преимущества: Снижение риска использования уязвимого кода, повышение общей безопасности приложения, упрощение управления зависимостями.
* Логирование и мониторинг
  1. Обоснование: Централизованное логирование и мониторинг событий безопасности позволяют своевременно обнаруживать и реагировать на подозрительную активность и инциденты безопасности. Регулярный анализ логов помогает выявлять аномалии и потенциальные угрозы, что позволяет принимать меры до того, как они приведут к серьезным последствиям;
  2. Преимущества: Своевременное обнаружение и реагирование на угрозы, улучшение видимости и контроля над системой, повышение общей безопасности.
* Обработка ошибок и исключений
  1. Обоснование: Безопасная обработка ошибок и исключений предотвращает раскрытие чувствительной информации и обеспечивает стабильность приложения. Логирование ошибок и предоставление пользователям понятных и безопасных сообщений об ошибках без раскрытия технических деталей помогает защитить систему от злоупотреблений;
  2. Преимущества: Повышение безопасности и стабильности приложения, улучшение пользовательского опыта, снижение риска раскрытия чувствительной информации.
* Рекомендации OWASP
  1. Обоснование: Использование рекомендаций OWASP, таких как OWASP Top 10, OWASP ASVS и OWASP Cheat Sheets, обеспечивает соответствие лучшим практикам и стандартам безопасности. Эти рекомендации помогают защитить веб-приложения от наиболее распространенных уязвимостей и обеспечить высокий уровень безопасности;
  2. Преимущества: Соответствие лучшим практикам и стандартам безопасности, повышение уровня защиты веб-приложений, упрощение процесса обеспечения безопасности.

# 2 Специальная, практическая, опытно-экспериментальная часть

# 2.1 Алгоритм проведения анализа кода на безопасность

2.1.1 Этапы анализа кода на безопасность

Анализ кода на безопасность является критически важным этапом в процессе разработки программного обеспечения. Он позволяет выявить уязвимости и потенциальные угрозы, которые могут быть использованы злоумышленниками для компрометации системы. В этом разделе описывается алгоритм проведения анализа кода на безопасность, включающий использование различных методов и инструментов для всестороннего исследования кода.

Этапы анализа:

* Подготовка к анализу:
  1. Сбор информации: Сбор всей необходимой информации о проекте, включая документацию, исходный код, конфигурационные файлы и зависимости;
  2. Определение целей: Определение целей анализа, таких как выявление уязвимостей, проверка соответствия стандартам безопасности и оценка общего уровня безопасности кода.
* Статический анализ кода (SAST) Выбор инструментов:
  1. Выбор инструментов для статического анализа кода, таких как SonarQube;
  2. Настройка инструментов: Настройка выбранных инструментов для анализа кода, включая конфигурацию правил и параметров анализа;
  3. Проведение анализа: Запуск статического анализа кода и сбор результатов;
  4. Анализ результатов: Анализ результатов статического анализа, выявление уязвимостей и потенциальных проблем.
* Динамический анализ кода (DAST) Выбор инструментов:
  1. Выбор инструментов для динамического анализа кода, таких как OWASP ZAP или Burp Suite;
  2. Настройка инструментов: Настройка выбранных инструментов для анализа работающего приложения, включая конфигурацию тестовых сценариев и параметров анализа;
  3. Проведение анализа: Запуск динамического анализа кода и сбор результатов;
  4. Анализ результатов: Анализ результатов динамического анализа, выявление уязвимостей и потенциальных проблем.
* Анализ зависимостей (SCA)
  1. Выбор инструментов: Выбор инструментов для анализа зависимостей, таких как OWASP Dependency-Check или Snyk;
  2. Настройка инструментов: Настройка выбранных инструментов для анализа зависимостей, включая конфигурацию правил и параметров анализа;
  3. Проведение анализа: Запуск анализа зависимостей и сбор результатов;
  4. Анализ результатов: Анализ результатов анализа зависимостей, выявление уязвимостей в сторонних библиотеках и зависимостях.
* Ручной анализ кода
  1. Обзор кода: Ручной обзор исходного кода для выявления потенциальных уязвимостей и проблем, которые могут быть пропущены автоматизированными инструментами;
  2. Проверка логики: Проверка логики работы приложения, выявление логических ошибок и уязвимостей;
  3. Анализ конфигураций: Проверка конфигурационных файлов и настроек безопасности;
* Документирование результатов
  1. Создание отчета: Создание подробного отчета о результатах анализа, включая выявленные уязвимости, их классификацию и рекомендации по исправлению;
  2. Предоставление рекомендаций: Предоставление рекомендаций по улучшению безопасности кода и предотвращению выявленных уязвимостей.

2.1.2 Допустимость частичного соблюдения этапов

Хотя рекомендуется соблюдать все этапы анализа для обеспечения максимальной безопасности, допускается частичное соблюдение этапов в зависимости от конкретных условий и требований проекта. Однако, пропуск определенных этапов может иметь следующие последствия:

* Пропуск статического анализа кода (SAST): Может привести к пропуску уязвимостей, связанных с кодом, таких как ошибки программирования и небезопасные практики кодирования;
* Пропуск динамического анализа кода (DAST): Может привести к пропуску уязвимостей, которые проявляются только при выполнении кода, таких как уязвимости, связанные с взаимодействием компонентов и внешними системами;
* Пропуск анализа зависимостей (SCA): Может привести к пропуску уязвимостей в сторонних библиотеках и зависимостях, что может быть использовано злоумышленниками для компрометации системы;
* Пропуск ручного анализа кода: Может привести к пропуску уязвимостей, которые не могут быть выявлены автоматизированными инструментами, таких как логические ошибки и специфические уязвимости, связанные с бизнес-логикой приложения.

Алгоритм проведения анализа кода на безопасность включает использование различных методов и инструментов для всестороннего исследования кода. Хотя допускается частичное соблюдение этапов, это может привести к определенным последствиям, таким как пропуск уязвимостей и потенциальных угроз. Полное соблюдение всех этапов анализа позволяет выявить уязвимости и потенциальные проблемы, а также предоставить рекомендации по их исправлению, что способствует повышению общего уровня безопасности программного обеспечения.

# 2.2 Анализ предоставленного кода не безопасность

2.2.1 Найденные уязвимости в выбранном проекте

В рамках данного исследования, один из участников предоставил исходный код для анализа на безопасность. Был частично предоставлен код проекта “Разработка модулей взаимодействия с информационной системой управления нейросетями”. Целью данного анализа является выявление потенциальных уязвимостей и проблем, которые могут быть использованы злоумышленниками для компрометации системы. Анализ предоставленного кода позволяет не только оценить текущий уровень безопасности, но и предложить рекомендации по его улучшению.

На рисунке А.1 предоставлен код со страницы Login.vue. Просмотрев данный код, можно увидеть отсутствие валидации для полей email и password. Также можно увидеть отсутствие этого и на страницах PasswordRecovery.vue рисунок A.2 и Registration.vue рисунок А.3.

Отсутствие валидации входных данных в коде может привести к различным проблемам безопасности и функциональности. К примеру:

* SQL-инъекции:
  1. Хотя SQL-инъекции обычно связаны с серверной стороной, отсутствие валидации на клиентской стороне может позволить злоумышленнику отправить вредоносные данные на сервер, что может быть использовано для атак на базу данных.
* Cross-Site Scripting (XSS):
  1. Отсутствие валидации и санитизации входных данных может позволить злоумышленнику внедрить вредоносный JavaScript-код, который будет выполнен в браузере пользователя. Это может привести к краже данных, фишингу и другим атакам.
* Некорректные данные:
  1. Отсутствие валидации может позволить пользователям вводить некорректные данные, что может привести к ошибкам в логике приложения, неправильной обработке данных и другим проблемам.

На рисунке А.4 предоставлен код с динамическим импортом на основе props.name. Здесь имеются проблемы, потому что данный динамический импорт может быть уязвим к следующим атакам:

* Path Traversal (Выход за пределы директории);
  1. Описание: Динамический импорт на основе значения props.name может быть уязвим к атакам типа "Path Traversal" или "Directory Traversal". Злоумышленник может попытаться использовать специальные символы, такие как ../, чтобы получить доступ к файлам за пределами директории @/app/assets/icons/. ;
  2. Последствия: Это может позволить злоумышленнику получить доступ к конфиденциальным файлам или данным, находящимся за пределами предназначенной директории.

На рисунке А.5 предоставлен код с использованием v-bind в стилях. Использование v-bind чревато следующими уязвимостями:

* Cross-Site Scripting (XSS);
  1. Описание: Использование v-bind в стилях может быть уязвимо к XSS-атакам, если значения scale и color не проверяются и не санитизируются. Злоумышленник может внедрить вредоносный JavaScript-код, который будет выполнен в браузере пользователя;
  2. Последствия: Это может привести к краже данных, фишингу и другим атакам

2.2.2 Рекомендации по устранению уязвимостей

Для устранения уязвимостей с валидацией могут быть применены следующие рекомендации:

* Добавьте валидацию: Используйте библиотеки, такие как Vuelidate, для добавления валидации входных данных, ну или напишите собственную валидацию;
* Санитизация данных: Используйте библиотеки для очистки входных данных от потенциально опасных символов и последовательностей;
* Используйте белые списки: Разрешайте только определенные типы и значения данных;
* Шифрование данных: Используйте сильные алгоритмы шифрования для защиты данных;

Для устранения уязвимостей, связанных с динамическим импортом, могут быть применены следующие рекомендации:

* Добавьте валидацию: Используйте библиотеки, такие как Vuelidate, для добавления валидации входных данных, или напишите собственную валидацию.
* Санитизация данных: Используйте библиотеки для очистки входных данных от потенциально опасных символов и последовательностей.
* Использование белых списков: Разрешайте только определенные типы и значения данных.

Для устранения уязвимостей, связанных с применением v-bind, могут быть применены следующие рекомендации:

* Добавьте валидацию: Используйте библиотеки, такие как Vuelidate, для добавления валидации входных данных, или напишите собственную валидацию.
* Санитизация данных: Используйте библиотеки для очистки входных данных от потенциально опасных символов и последовательностей.

# 3 Анализатор SQL кода

# 3.1 Введение

# SQL-инъекции являются одной из наиболее распространенных и опасных уязвимостей в веб-приложениях. Они позволяют злоумышленникам внедрять вредоносные SQL-запросы, что может привести к несанкционированному доступу к данным, их изменению или удалению. Для повышения безопасности приложений, использующих SQL, необходимо разработать инструмент, который автоматически анализирует SQL-код на предмет уязвимостей. В этом разделе описывается разработка самописного анализатора SQL кода, который поможет выявлять потенциальные уязвимости и предоставлять рекомендации по их устранению.

# 3.2 Цели и задачи

Целью разработки анализатора SQL кода является создание инструмента, который автоматически анализирует SQL-запросы на предмет уязвимостей и предоставляет рекомендации по их устранению. Основные задачи включают:

1) Анализ SQL-запросов: Разработка алгоритмов для анализа SQL-запросов на предмет потенциальных уязвимостей.

2) Выявление уязвимостей: Определение наиболее распространенных уязвимостей, таких как SQL-инъекции, и разработка методов их выявления.

# 3.3 Работа анализатора

Принцип работы анализатора:

* Инициализация: При запуске программы создаются объекты UserInteraction, SQLAnalyzer и ReportGenerator.
* Выбор режима работы: Пользователь выбирает режим работы: анализ одного файла, анализ директории или завершение работы.
* Анализ одного файла: Пользователь вводит путь к файлу. Программа проверяет существование файла. Если файл существует, он читается построчно. Каждая строка анализируется на наличие потенциальных SQL-инъекций с использованием JSqlParser. Результаты анализа сохраняются в список отчетов.
* Анализ директории: Пользователь вводит путь к директории. Программа проверяет существование директории. Если директория существует, создаются задачи для анализа каждого файла в директории. Задачи выполняются в параллельных потоках. Результаты анализа собираются и сохраняются в список отчетов.
* Генерация отчета: На основе списка отчетов генерируется отчет в формате JSON. Отчет сохраняется в файл.
* Продолжение работы: Пользователь выбирает, продолжить ли работу или завершить. Если выбран продолжение работы, программа возвращается к шагу выбора режима работы. Если выбран завершение работы, программа завершается.

На рисунке Б.3 представлен алгоритм анализа SQL кода.

# Заключение

Разработанная методология и рекомендации по написанию безопасного кода с учетом принципов DevSecOps и кибербезопасности представляют собой комплексное и технологичное решение, отвечающее современным требованиям рынка и потребностям различных категорий пользователей. В ходе исследования были тщательно продуманы и протестированы все аспекты методологии: от минимизации привилегий и проверки ввода до шифрования данных и использования безопасных библиотек и фреймворков.

Комплексное тестирование, включающее статический анализ кода, динамический анализ, анализ зависимостей и ручной анализ, позволило выявить и устранить все выявленные уязвимости, гарантируя высокое качество и надежность работы платформы. Особого внимания заслуживает интуитивно понятный интерфейс для пользователей, который делает процесс анализа и улучшения безопасности кода простым и удобным.

Таким образом, разработанная методология представляет собой комплексное, технологичное и гибкое решение, отвечающее запросам современного рынка. Внедрение данных правил и рекомендаций позволит повысить эффективность процессов разработки программного обеспечения, улучшить уровень безопасности и обеспечить конкурентоспособность предприятия на рынке информационных технологий.

# Список литературы

1. ГОСТ 2.701—2008 Единая система конструкторской документации. Схемы. Виды и типы. Общие требования к выполнению. – М.: Стандартинформ, 2009. – 15 с. (Дата обращения: 10.11.2024);
2. ГОСТ 2.743—91 Единая система конструкторской документации. Обозначения условные графические в схемах. Элементы цифровой техники. – М.: Стандартинформ, 2009. – 62 с. (Дата обращения: 10.11.2024);
3. ГОСТ Р 7.0.97—2016. Требования к оформлению документов. (Дата обращения: 10.11.2024);
4. OWASP Top Ten 2021. – [Электронный ресурс] – URL: https://owasp.org/Top10/ (Дата обращения: 15.11.2024);
5. SonarQube Documentation. – [Электронный ресурс] – URL: https://docs.sonarqube.org/latest/ (Дата обращения: 15.11.2024);
6. OWASP Dependency-Check Documentation. – [Электронный ресурс] – URL: https://jeremylong.github.io/DependencyCheck/ (Дата обращения: 15.11.2024);
7. OWASP ZAP Documentation. – [Электронный ресурс] – URL: https://www.zaproxy.org/docs/ (Дата обращения: 15.11.2024);
8. Burp Suite Documentation. – [Электронный ресурс] – URL: https://portswigger.net/burp/documentation (Дата обращения: 15.11.2024);
9. NIST Special Publication 800-53 Rev. 5: Security and Privacy Controls for Information Systems and Organizations. – [Электронный ресурс] – URL: https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final (Дата обращения: 15.11.2024);
10. CWE/SANS Top 25 Most Dangerous Software Errors. – [Электронный ресурс] – URL: https://cwe.mitre.org/top25/ (Дата обращения: 15.11.2024);
11. The Web Application Hacker's Handbook: Finding and Exploiting Flaws. – [Электронный ресурс] – URL: https://www.wiley.com/en-us/The+Web+Application+Hacker%27s+Handbook%3A+Finding+and+Exploiting+Flaws-p-9781118026472 (Дата обращения: 15.11.2024).

# Приложение А

# (обязательное)

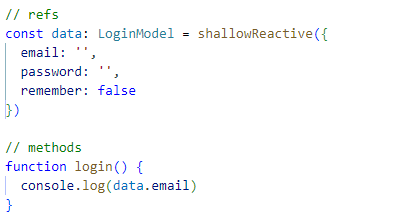


Рисунок А.1 – код со страницы Login.vue

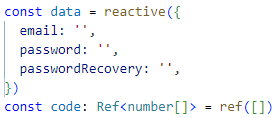


Рисунок А.2 – код со страницы PasswordRecovery.vue



Рисунок А.3 – код со страницы Registration.vue



Рисунок А.4 – код с динамическим импортом ui.vue

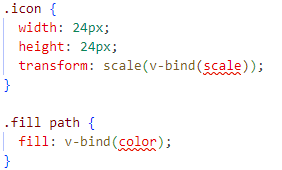


Рисунок А.5 – код с применением v-bind ui.vue

# Приложение Б

(обязательное)

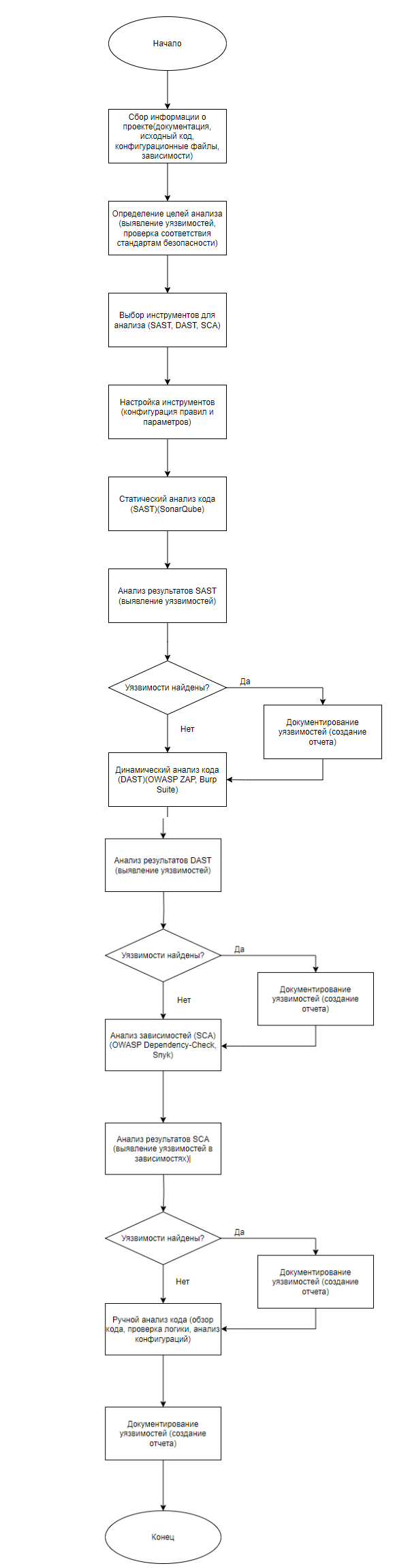


Рисунок Б.1 – Блок-схема алгоритма анализа кода на уязвимости

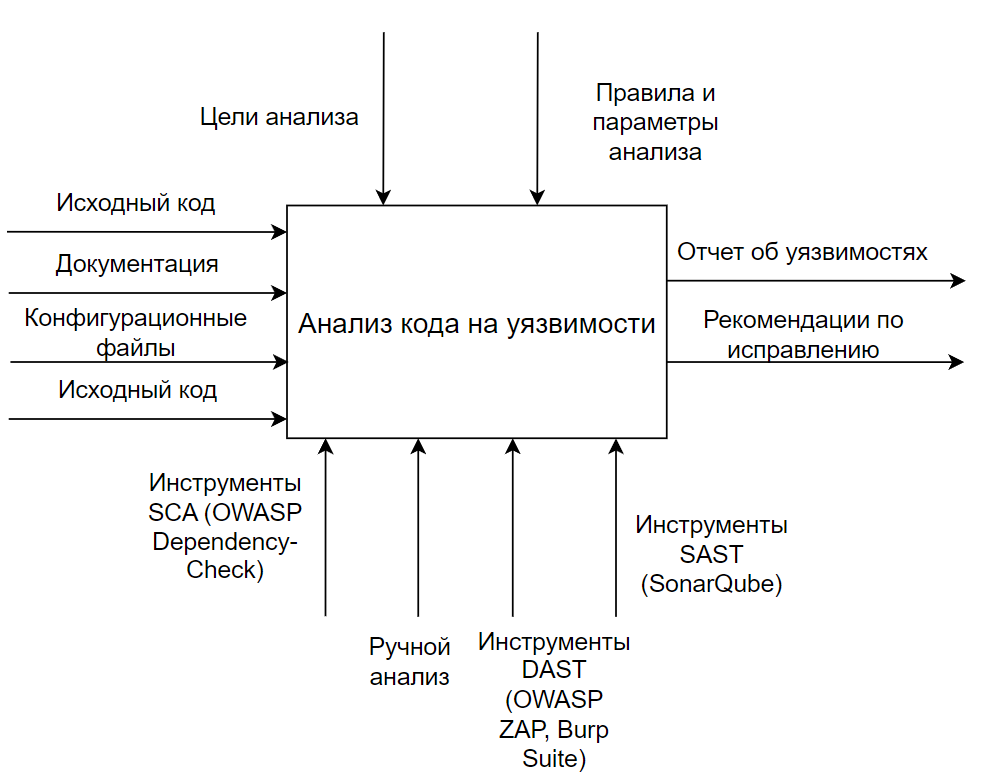


Рисунок Б.2 – Функциональная диаграмма анализа кода на безопасность

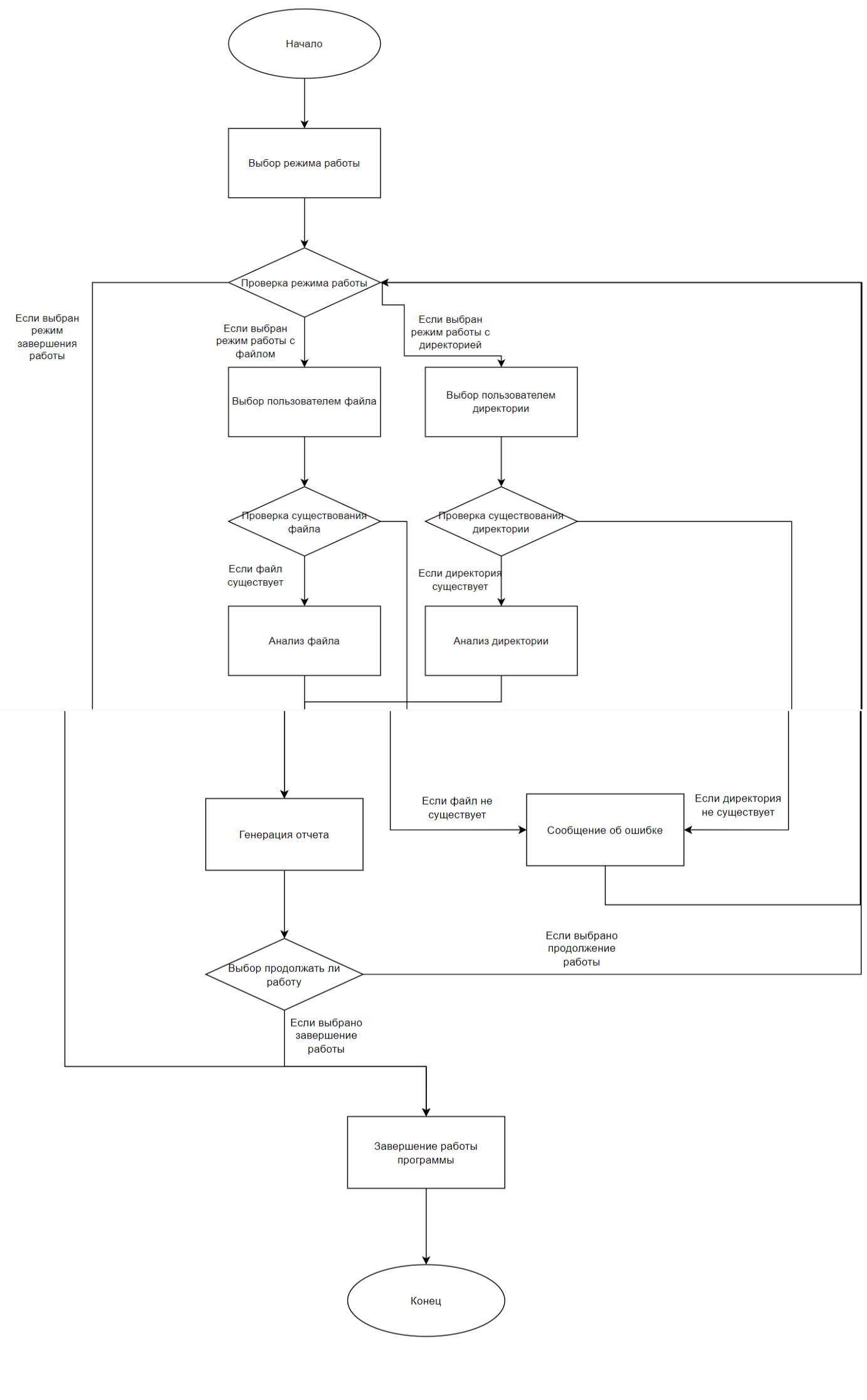


Рисунок Б.3 – Алгоритм программы анализа SQL кода

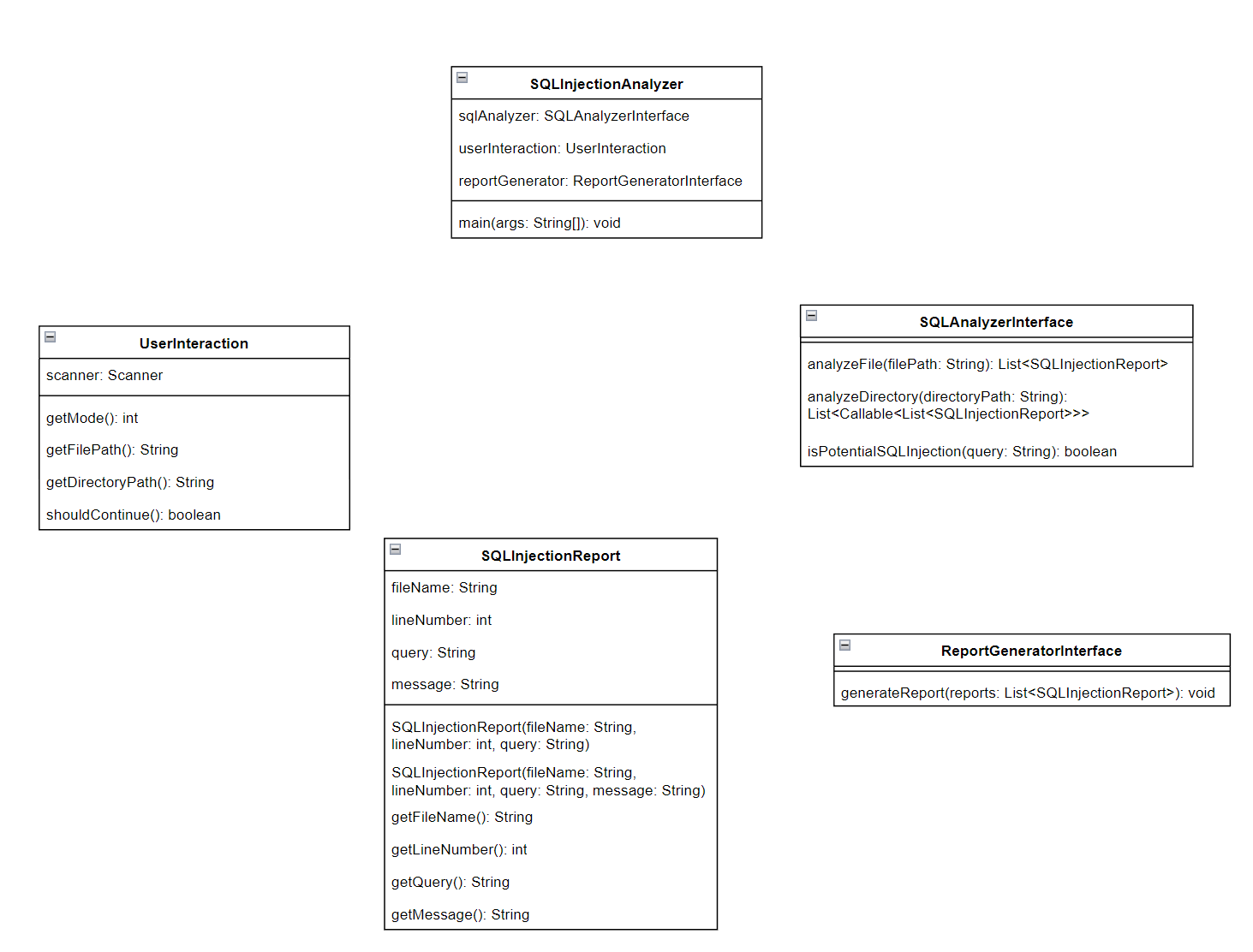


Рисунок Б.4 – Диаграмма анализатора SQL-кода

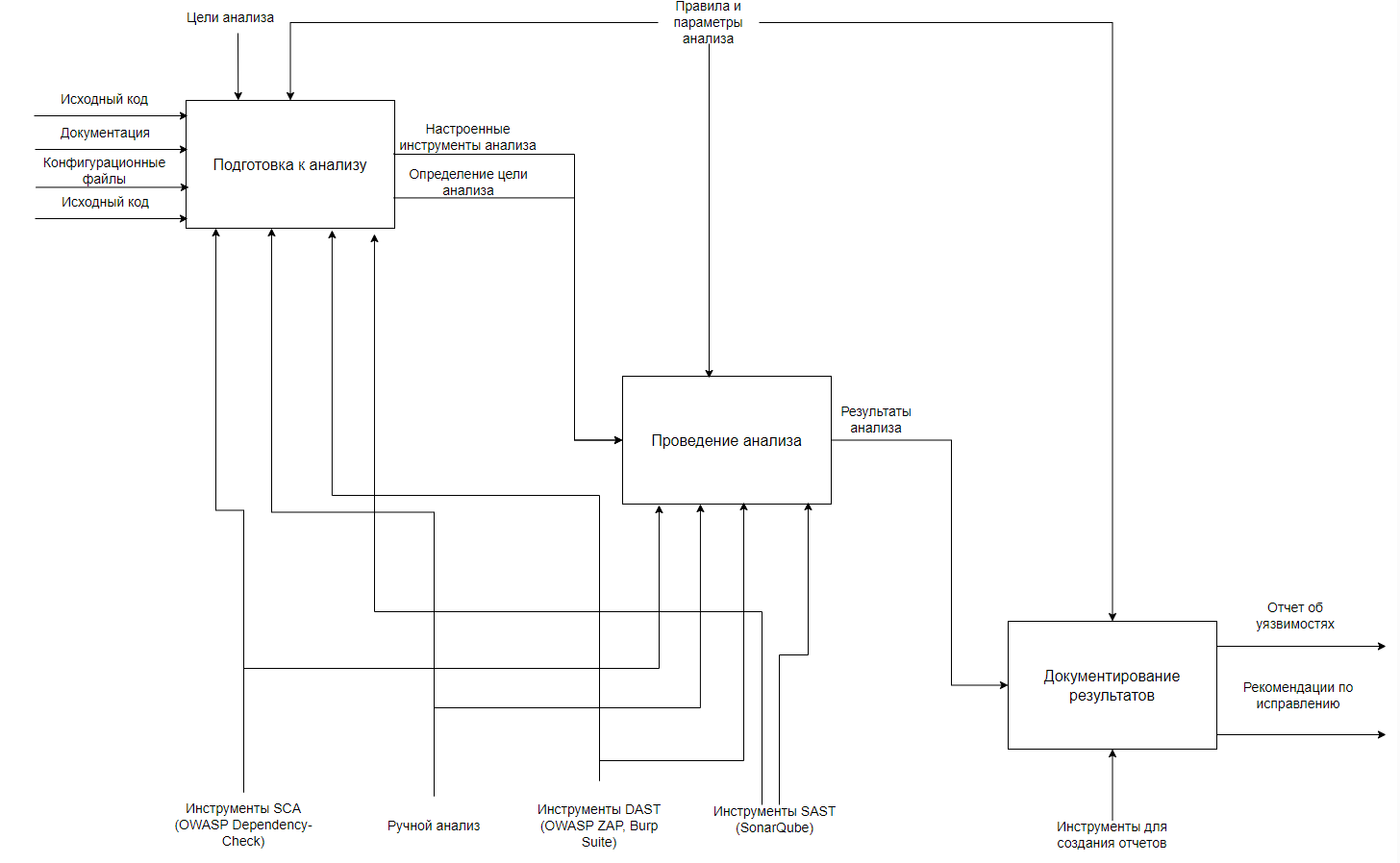


Рисунок Б.5 – Декомпозиция функциональной диаграммы анализа кода на безопасность