ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

**Московский институт электроники и математики им. А.Н. Тихонова**

Фамилия Имя Отчество автора

**НАЗВАНИЕ ТЕМЫ ВКР**

Выпускная квалификационная работа – магистерская диссертация

по направлению 10.04.01 «Информационная безопасность»

студента образовательной программы магистратуры  
«Наименование образовательной программы»

|  |  |
| --- | --- |
| Студент  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  И.О. Фамилия  Рецензент  уч. степень, уч. звание  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  И.О. Фамилия | Научный руководитель  уч. степень, уч. звание  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  И.О. Фамилия  Консультант  уч. степень, уч. звание  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  И.О. Фамилия |

Москва 2025

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

**Московский институт электроники и математики им. А.Н. Тихонова**

**ЗАДАНИЕ**

**на выполнение выпускной квалификационной работы**

студенту группы \_\_\_\_\_ Фамилия Имя Отчество

1. Тема работы

*Тема ВКР приводится в соответствии с приказом об утверждении тем ВКР.*

1. Требования к работе

2.1. Цель работы

*Цель работы формулируется в контексте повышения эффективности некоторого научно-технического решения либо в контексте решения конкретной прикладной задачи. Из формулировки цели ВКР должна быть понятна ее актуальность*.

2.2. Требования к результатам работы

*Перечисляются основные результаты, которые должны быть получены в рамках ВКР, и уточняются их технические характеристики.*

2.3. Требования к документации

Текст ВКР должен быть оформлен в соответствии с ГОСТ.

1. Содержание работы

*Перечисляются основные выполняемые работы, в основном соответствующие структуре ВКР, представленной в пояснительной записке. Обязательные работы приведены в настоящем шаблоне. Все прочие работы определяются студентом совместно с руководителем.*

3.1. Обзор научно-технической литературы по теме работы

3.2. …

3.\_. Подготовка пояснительной записки к ВКР

1. Сроки выполнения этапов работы

|  |  |
| --- | --- |
| Проект ВКР представляется студентом в срок до | «16» февраля 2025 г. |
| Первый вариант ВКР представляется студентом в срок до | «31» марта 2025 г. |
| Итоговый вариант ВКР представляется студентом руководителю до загрузки работы в систему «Антиплагиат» в срок до | «30» апреля 2025 г. |

|  |  |  |
| --- | --- | --- |
| Задание выдано | «20» декабря 2024 г. | \_\_\_\_\_\_\_\_\_\_\_\_ И.О. Фамилия  подпись руководителя |
|  |  |  |
| Задание принято к исполнению | «20» декабря 2024 г. | \_\_\_\_\_\_\_\_\_\_\_\_ И.О. Фамилия  подпись студента |

**Аннотация**

В аннотации кратко описывается решаемая в выпускной квалификационной работе (ВКР) задача, обосновывается ее актуальность, описываются использованные методы и приводятся полученные результаты. При описании полученных результатов указываются их отличительные особенности и преимущества. Работа содержит \_\_ страниц, \_\_ рисунков, \_\_ таблиц, \_\_ источников, \_\_ приложений.

**Abstract**

Перевод аннотации на английский язык.

**СОДЕРЖАНИЕ**

[1 Введение 7](#__RefHeading___Toc3438_1511805350)

[2 Обзор литературы 8](#__RefHeading___Toc3440_1511805350)

[2.1 SCA для Java-проектов с использованием Spring Boot 8](#__RefHeading___Toc3442_1511805350)

[2.1.1 Обзор иностранных опенсорсных SCA-инструментов 10](#__RefHeading___Toc3444_1511805350)

[2.1.2 Обзор отечественных SCA-инструментов 12](#__RefHeading___Toc3446_1511805350)

[2.2 Название подраздела 13](#__RefHeading___Toc3448_1511805350)

[2.2.1 Название пункта 13](#__RefHeading___Toc3450_1511805350)

[2.2.2 Название пункта 13](#__RefHeading___Toc3452_1511805350)

[3 Заключение 14](#__RefHeading___Toc3454_1511805350)

[4 Список использованных источников 15](#__RefHeading___Toc3456_1511805350)

# 1 Введение

Цель данной работы заключается в повышении эффективности разработки и обеспечения безопасности микросервисов, построенных на платформе Java Spring Boot, путем интеграции методов статического и динамического анализа, а также анализа состава программного обеспечения. В частности, акцент будет сделан на использовании инструментов SCA (Software Composition Analysis), SAST (Static Application Security Testing) и DAST (Dynamic Application Security Testing).

В рамках исследования будет изучено применение опенсорсных библиотек, доступных для скачивания в России, что обеспечит доступность и возможность практической реализации предложенных решений. Особое внимание будет уделено поиску и оценке русских аналогов анализируемых инструментов, если таковые найдутся.

В результате работы будет представлена практическая реализация интеграции инструментов SCA и SAST в процесс разработки микросервисов, что позволит продемонстрировать их взаимодополняемость и вклад в повышение уровня безопасности приложений. Кроме того, будет рассмотрено применение DAST в рамках тестирования и эксплуатации, что подчеркнет важность комплексного подхода к обеспечению безопасности. Таким образом, работа направлена не только на решение конкретных прикладных задач, но и на развитие понимания актуальных методов обеспечения безопасности в контексте современного программного обеспечения.

# 2 Обзор литературы

В этой главе я проведу обзор литературы, посвященной инструментам статического и динамического анализа программного обеспечения (SCA, SAST и DAST), с акцентом на их применение в контексте разработки Java-приложений, использующих библиотеку Spring Boot.

Также я буду рассматривать использование этих инструментов в процессах непрерывной интеграции и доставки (CI/CD), что является важной частью практик DevSecOps. Одним из ключевых аспектов внедрения SAST и DAST инструментов должно быть их удобство интеграции в существующие рабочие процессы разработки. Эти инструменты должны быть легко подключяемы к CI/CD конвейерам.

## 2.1 ****SCA для Java-проектов с использованием Spring Boot****

Компоненты с открытым исходным кодом стали основными элементами программного обеспечения в различных областях. Системы SCA помогают контролировать эти компоненты в приложениях, что критично для их безопасности. Всё больше современных приложений строится на базе открытого кода, и по некоторым данным, до 90% их кода может состоять из опенсорса [1]. Анализ компонентов программного обеспечения (SCA) представляет собой процесс идентификации и оценки компонентов, особенно открытого исходного кода, в рамках программного обеспечения, а также проверки их безопасности. Используя SCA, команды разработчиков могут быстро обнаруживать и анализировать опенсорсные компоненты, интегрированные в проект. Инструменты SCA способны выявлять все связанные компоненты и библиотеки, а также прямые и косвенные зависимости. Кроме этого, системы SCA выявляют лицензии программного обеспечения, устаревшие зависимости, уязвимости и потенциальные угрозы.

В результате сканирования создаётся полная спецификация, которая позволяет провести инвентаризацию программных активов проекта. Хотя инструменты SCA существуют уже давно, их значение возросло с увеличением использования открытого исходного кода в последние годы, и они стали ключевыми для повышения безопасности приложений. Современные методологии разработки, такие как DevSecOps, требуют, чтобы SCA был удобен как для разработчиков, так и для специалистов по безопасности, чья роль заключается в поддержке разработчиков на всех этапах жизненного цикла разработки программного обеспечения.

В области разработки программного обеспечения эффективное управление зависимостями является ключевым аспектом в создании надежных и безопасных приложений. Spring Boot, который зарекомендовал себя как предпочтительный инструмент среди разработчиков на языке Java, значительно упрощает процесс разработки, однако не следует ограничиваться лишь его удобством. Критически важно осуществлять мониторинг и актуализацию зависимостей, чтобы обеспечить бесперебойную работу проектов на Spring Boot и их устойчивость к постоянно эволюционирующим угрозам [2].

Одним из наиболее важных аспектов управления зависимостями в рамках Spring Boot является вопрос безопасности. Уязвимости в программном обеспечении регулярно выявляются, и своевременное обновление зависимостей в проекте можно рассматривать как форму цифровой защиты. Устаревшие зависимости представляют собой потенциальные риски, аналогичные незапертым дверям, которые могут способствовать проникновению угроз. Следовательно, необходимо принять меры для предотвращения таких ситуаций и обеспечить безопасность и целостность разрабатываемых приложений [2].

Gradle и Maven — это популярные инструменты для управления проектами и построения программного обеспечения на языке Java. Они упрощают процесс сборки, управления зависимостями и автоматизации рабочего процесса, позволяя разработчикам сосредоточиться на написании кода, а не на решении организационных задач.

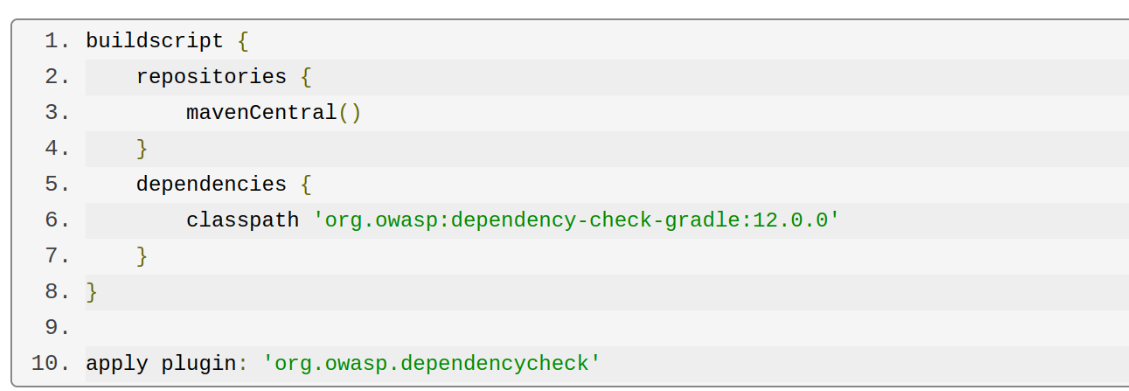
В ходе обзора средств анализа компонентов программного обеспечения (SCA) для Java-приложений я отметила, что фреймворки SCA различаются по удобству использования для разработчиков. К менее удобным инструментам можно отнести те, которые требуют локальной установки для анализа кода, что подразумевает необходимость запуска исполняемого файла приложения. В то же время существуют более удобные SCA-инструменты, которые позволяют легко добавлять зависимости в Gradle или Maven, после чего их можно использовать непосредственно через эти системы управления проектами.

В следующей части работы будет представлен обзор международных опенсорсных SCA-инструментов, которые я бы захотела использовать благодаря их удобству и эффективности. Также мне удалось обнаружить перспективный отечественный SCA-инструмент, о котором стоит рассказать.

### 2.1.1 **Обзор иностранных опенсорсных SCA-инструментов**

1. *OWASP dependency check*. Одним из популярных решений является OWASP Dependency-Check, который можно использовать в качестве плагина для Gradle или Maven. При выполнении он сравнивает все зависимости приложения с базой данных NIST NVD и индексом Sonatype OSS. Этот инструмент позволяет подавлять предупреждения и генерировать отчеты, а также легко интегрируется в CI-пайплайн. Основным недостатком является то, что он иногда выдает ложные срабатывания, так как база данных NIST NVD не предоставляет данные в идеальном формате. Кроме того, первый запуск занимает много времени, так как необходимо загрузить всю базу данных уязвимостей [3]. Использую Gradle, поэтому приведу пример как добавить плагин OWASP Dependency-Check в build.gradle (Рисунок 1), запустить сканирование (Рисунок 2) [4].

Запустить сканирование можно так (смотрите рисунок 2). Посмотреть результаты сканирования можно в папке ${buildDir}/reports.

Рисунок 1 - Инсталляция из центрального репозитория Maven [4]

1. *Snyk*. Snyk Gradle Plugin - это плагин для системы сборки Gradle, который позволяет интегрировать функциональность Snyk прямо в процесс сборки Java-проекта. Этот плагин позволяет использовать Snyk для сканирования зависимостей проекта и исходного кода на наличие уязвимостей и других проблем безопасности. Он включает в себя функционал Snyk Open Source и Snyk Code.

Рисунок 2 - Запуск плагина [4]

Snyk Open Source: Сканирует зависимости проекта (библиотеки и пакеты с открытым исходным кодом) на наличие известных уязвимостей. Предлагает решения по устранению найденных уязвимостей (например, обновление до более безопасной версии или применение патча).

Snyk Code: Анализирует исходный код на наличие уязвимостей безопасности и проблем с качеством кода (например, «code smells», ошибки логики и т. д.). Пример конфигурации в build.gradle (на примере сканирования зависимостей, Рисунок 3) [5].

Бесплатная версия Snyk предоставляет ограниченное количество ежемесячных сканирований, так как доступ к её возможностям осуществляется через API ключ (настройка apiToken, Рисунок 5), полученный при регистрации. Поэтому постоянный запуск тестов на каждом этапе сборки может привести к быстрому исчерпанию лимита. Рекомендуется настроить запуск сканирования Snyk реже, например, только перед релизом или по расписанию.

Рисунок 3 -Пример конфигурации плагина Snyk

# 

### 2.1.2 **Обзор отечественных SCA-инструментов**

# *CodeScoring*. Платформа композиционного анализа программного обеспечения для анализа безопасности и качества кода, который помогает разработчикам выявлять уязвимости и недостатки в программном обеспечении. Он предоставляет автоматизированные проверки кода, позволяя командам разработчиков быстро находить и исправлять проблемы до того, как они станут серьезными угрозами. Единственный инструмент отечественный инструмент SCA, который мне удалось найти.

Здесь используется Агент CLI (Command Line Interface). Это инструмент, который работает через командную строку и выполняет определенные задачи, такие как анализ, сканирование или управление проектами.

Агент CLI Johnny поставляется с локальной версией CodeScoring. Агент представляет собой исполняемый бинарный файл, который анализирует манифесты известных менеджеров пакетов, сканирует образы Docker и ищет прямые включения библиотек с открытым исходным кодом, используя их хеши. Агент может работать как независимо от установки, так и в связке с ней, получая данные о настроенных политиках и сохраняя результаты сканирования в существующие проекты.

При работе с Java для запуска сканирования с CLI Johnny необходимо дополнительно создать артефакты (файл в формате txt), которые содержат полную структуру зависимостей проекта. После создания артефактов вам необходимо использовать команду сканирования файла на полученном артефакте (смотрите Рисунок 4).



# **2.2  SAST **для Java-проектов с использованием Spring Boot****

**SAST (Static Application Security Testing) — это метод анализа безопасности программного обеспечения, который исследует исходный код приложения для выявления уязвимостей и недостатков в безопасности до его компиляции. Он позволяет разработчикам заранее находить и устранять проблемы, снижая риски в процессе разработки и увеличивая общую безопасность приложения [7].**

**SAST (Static Application Security Testing) может быть интегрирован в процесс CI/CD (Continuous Integration/Continuous Deployment). Это позволяет автоматически проверять код на наличие уязвимостей и нарушений безопасного программирования еще на этапе разработки. Вот несколько ключевых аспектов интеграции SAST в CI/CD:**

**Автоматизация: Инструменты SAST могут быть настроены для автоматического запуска тестов при каждом коммите или перед слиянием в основную ветку. Это помогает быстро выявлять и исправлять уязвимости.**

**Результаты в реальном времени: Разработчики могут получать результаты анализа кода в реальном времени, что позволяет им оперативно реагировать на найденные проблемы.**

**Формирование отчетов: Инструменты SAST могут автоматически генерировать отчеты о найденных уязвимостях, что упрощает процесс ревью кода и планирования исправлений.**

**Интеграция с другими инструментами: SAST может быть интегрирован с другими инструментами DevOps, такими как системы отслеживания задач, репозитории кода и платформы CI/CD, что улучшает общий процесс разработки.**

**SAST-анализаторы осуществляют проверку исходного кода на наличие распространённых уязвимостей, включая те, которые перечислены в списке OWASP Top Ten. Они способны выявлять не только сами уязвимости, но и указывать на конкретные фрагменты кода, которые являются их источником [8].**

**Данный вид анализа также известен как проверка методом «белого ящика» (White Box Testing), поскольку анализатор имеет доступ к внутренней структуре приложения. Важно учитывать, что SAST-анализаторы осуществляют проверку исходного кода без его запуска, что может привести к ложным срабатываниям и может не выявить некоторые виды уязвимостей. Поэтому не рекомендуется полагаться исключительно на SAST-анализ [8].**

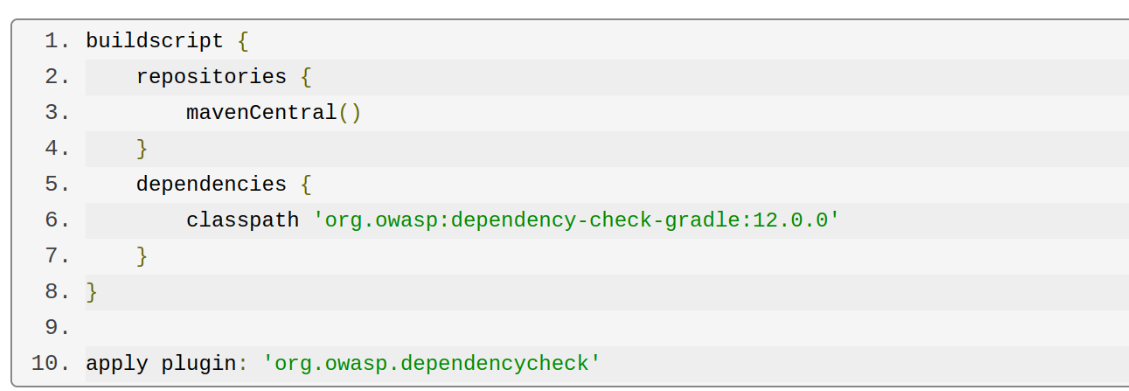
Важные критерии выбора [7]:

1. Способность обнаруживать уязвимости по следующим критериям: OWASP Top Ten
2. Точность: уровни ложных срабатываний/ложных отрицаний; оценка по стандартам OWASP Benchmark
3. Способность понимать библиотеки/фреймворки, которые вам нужны.
4. Требование к доступности исходного кода для компиляции.
5. Возможность работы с двоичными файлами (вместо исходного кода).
6. **Простота настройки/использования.**

### 2.2.1 ****Обзор иностранных опенсорсных SAST-инструментов****

1. *OWASP dependency check*. Одним из популярных решений является OWASP Dependency-Check, который можно использовать в качестве плагина для Gradle или Maven. При выполнении он сравнивает все зависимости приложения с базой данных NIST NVD и индексом Sonatype OSS. Этот инструмент позволяет подавлять предупреждения и генерировать отчеты, а также легко интегрируется в CI-пайплайн. Основным недостатком является то, что он иногда выдает ложные срабатывания, так как база данных NIST NVD не предоставляет данные в идеальном формате. Кроме того, первый запуск занимает много времени, так как необходимо загрузить всю базу данных уязвимостей [3]. Использую Gradle, поэтому приведу пример как добавить плагин OWASP Dependency-Check в build.gradle (Рисунок 1), запустить сканирование (Рисунок 2) [4].

Запустить сканирование можно так (смотрите рисунок 2). Посмотреть результаты сканирования можно в папке ${buildDir}/reports.

Рисунок 3 - Инсталляция из центрального репозитория Maven [4]

### 2.2.1 DefectDojo

Текст.

### 2.2.2 Название пункта

Текст.

# 3 Заключение

Приводится краткое описание результатов ВКР.

# 4 Список использованных источников

1. Обзор рынка инструментов SCA (Software Composition Analysis) – URL: https://www.anti-malware.ru/analytics/Market\_Analysis/Software-Composition-Analysis (дата обращения 15.01.2025).

2. Handling security vulnerabilities in Spring Boot | Snyk. – URL: https://snyk.io/blog/security-vulnerabilities-spring-boot/ (дата обращения 15.01.2025).

3. Dealing with Java CVEs: Discovery, Detection, Analysis, and Resolution. – URL: https://www.infoq.com/articles/dealing-with-java-cves/ (дата обращения 16.01.2025).

4. OWASP/ Dependency-Check/ documentation/ Usage – URL: <http://jeremylong.github.io/DependencyCheck/dependency-check-gradle/index.html> (дата обращения 16.01.2025)

5. Detect Security Vulnerabilities with Snyk – URL: <https://www.baeldung.com/java-snyk-security-risks> (дата обращения 19.01.2025)

6. Resolving dependencies in the build environment – URL: <https://docs.codescoring.ru/agent/resolve.en/> (дата обращения 30.01.2025)

7. Source Code Analysis Tools  
- URL: <https://owasp.org/www-community/Source_Code_Analysis_Tools> (дата обращения 09.03.2025)

8. Этапы DevSecOps‑пайплайна: pre‑commit, pre‑build и post‑build – URL: <https://yandex.cloud/ru/blog/posts/2023/06/devsecops-steps-part-1> (дата обращения 09.03.2025)

9 OWASP Benchmark Project – URL: <https://owasp.org/www-project-benchmark/>(дата обращения 09.03.2025)