

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное  
учреждение высшего образования  
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»  
(ФГАОУ ВО «ЮФУ»)

Институт компьютерных технологий и информационной безопасности  
Кафедра безопасности информационных технологий

К защите допустить:  
Зав. кафедрой БИТ  
\_\_\_\_\_ Е. С. Абрамов  
«\_\_\_» \_\_\_\_\_ 2020 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

по образовательной программе  
«Анализ безопасности информационных систем»  
специальности  
10.05.03 Информационная безопасность автоматизированных систем  
на тему:

**«СРЕДСТВО ГИБРИДНОГО ТЕСТИРОВАНИЯ ЗАЩИЩЕННОСТИ ВЕБ-  
ПРИЛОЖЕНИЙ НА ОСНОВЕ ГРАФА СВОЙСТВ КОДА»**

Руководитель ВКР:  
доцент кафедры БИТ,  
к. т. н., доцент

\_\_\_\_\_  
(подпись, дата)

Е. С. Абрамов

Нормоконтроль:  
доцент кафедры БИТ,  
к. т. н., доцент

\_\_\_\_\_  
(подпись, дата)

А. В. Некрасов

Консультант по БЧМВ:  
доцент кафедры БИТ,  
к. т. н., доцент

\_\_\_\_\_  
(подпись, дата)

Н. Б. Ельчанинова

Консультант по ТЭО:  
доцент кафедры БИТ,  
к. т. н., доцент

\_\_\_\_\_  
(подпись, дата)

В. И. Струков

Выполнил:  
студент группы КТсо5-5

\_\_\_\_\_  
(подпись, дата)

О. П. Хабаров

Таганрог 2020

# **ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ**

по образовательной программе

«Анализ безопасности информационных систем»

специальности 10.05.03 Информационная безопасность автоматизированных  
систем

студенту группы КТсо5-5

Хабарову Олегу Петровичу

1. Тема выпускной квалификационной работы:

Средство гибридного тестирования защищенности веб-приложений на основе графа свойств кода.

2. Требования и исходные данные к работе:

Задачей ВКР является разработка средства гибридного тестирования защищенности веб-приложений на основе графа свойств кода. Результатом работы должна быть полностью функциональная система, которая последовательно выполняет статический и динамический анализ программного проекта с исходным кодом на языке программирования Java и обнаруживает такие опасные уязвимости как SQL-инъекции, инъекции команд ОС и HTML-инъекции (XSS-уязвимости).

3. Перечень подлежащих разработке вопросов (содержание работы):

1. Изучить общие принципы, лежащие в основе построения инструментов статического анализа.

2. Разработать модуль для построения синтаксической модели (абстрактного синтаксического дерева).

3. Разработать модуль для построения семантической модели (графы потока управления и потоков данных).

4. Разработать веб-приложение для удобного отображения синтаксической и семантической моделей.

5. Разработать модуль для проведения Taint-анализа.

6. Разработать модуль инструментирования исходного кода.

7. Разработать модуль динамического тестирования веб-приложения.

4. Перечень графических материалов:

1. Титульный лист
2. Перечень решаемых задач
3. Модели представления исходного кода
4. Генератор абстрактного синтаксического дерева
5. Генератор графа потока управления
6. Генератор графа потока данных
7. Taint-анализ
8. Динамический анализатор
9. Безопасность человеко-машинного взаимодействия
10. Заключение

5. Консультанты по выпускной квалификационной работе:

- по разделу «Безопасность человеко-машинного взаимодействия»

доцент кафедры БИТ, к.т.н., доцент Н. Б. Ельчанинова

- по разделу «Технико-экономическое обоснование»

доцент кафедры БИТ, к.т.н., доцент В. И. Струков

6. Срок сдачи студентом законченной ВКР руководителю: « 3 » июня 2020 г.

7. Дата выдачи задания: «14» февраля 2020 г.

Руководитель образовательной программы

зав. кафедрой БИТ,

к.т.н., доцент

\_\_\_\_\_  
(подпись, дата)

Е. С. Абрамов

Научный руководитель ВКР:

к.т.н., доцент кафедры БИТ,

\_\_\_\_\_  
(подпись, дата)

Е. С. Абрамов

Исполнитель:

студент группы КТсо5-5

\_\_\_\_\_  
(подпись, дата)

О.П. Хабаров

ВКР.20.КТсо5-5.100503.02

Хабаров Олег Петрович

Выпускная квалификационная работа  
Средство гибридного тестирования  
защищенности веб-приложений на  
основе графа свойств кода  
Южный Федеральный Университет,  
Таганрог, 2020

## **АННОТАЦИЯ**

Выпускная квалификационная работа изложена на 111 странице, содержит 50 рисунков, 24 источников и 6 приложений.

В процессе выполнения выпускной квалификационной работы было спроектировано и программно реализовано средство гибридного тестирования защищенности веб-приложений на основе графа свойств кода. Инструмент работает для веб-приложений, написанных на языке программирования Java. Разработан модуль для построения синтаксической и семантической моделей исходного кода анализируемых веб-приложений. Реализован веб-интерфейс для навигации по этим моделям. Разработан модуль taint-анализа. Разработан в виде библиотеки модуль динамического анализа веб-приложений.

При тестировании работы разрабатываемого средства использовались специально созданные сторонние уязвимые веб-приложения Damn Vulnerable Java Application и WebGoat.

GQW.20.KTso5-5.100503.02  
Khabarov Oleg Petrovich  
Graduation Qualification Work  
Web application hybrid security  
testing tool based on code property  
graph  
Southern Federal University,  
Taganrog, 2020

## **SUMMARY**

The graduation qualification work is presented on 111 pages, contains 50 figures, 24 sources and 6 applications.

In the process of completing the graduation qualification work, a web application hybrid security testing tool based on code property graph was designed and software implemented. The tool works for web applications written in the Java programming language. A module has been developed for constructing syntactic and semantic models of the source code of the analyzed web applications. Implemented a web interface for navigating these models. A taint analysis module has been developed. A module for dynamic analysis of web applications was developed as a library.

When testing the work of the developed tool, specially created third-party vulnerable web applications Damn Vulnerable Java Application and WebGoat were used.

## РЕФЕРАТ

Выпускная квалификационная работа изложена на 111 странице, содержит 50 рисунков, 24 источников и 6 приложений.

SAST, DAST, AST, CFG, DFG, GREMLIN, ИНСТРУМЕНТИРОВАНИЕ, ЗАЩИЩЕННОСТЬ ПРИЛОЖЕНИЙ.

В качестве объекта для проектирования было выбрано средство гибридного тестирования защищенности веб-приложений на основе графа свойств кода. Инструмент работает для веб-приложений, написанных на языке программирования Java. Разработан модуль для построения синтаксической и семантической моделей исходного кода анализируемых веб-приложений. Реализован веб-интерфейс для навигации по этим моделям. Разработан модуль taint-анализа. Разработан в виде библиотеки модуль динамического анализа веб-приложений.

При тестировании работы разрабатываемого средства использовались специально созданные сторонние уязвимые веб-приложения Damn Vulnerable Java Application и WebGoat.

Актуальность данного проекта можно обосновать тем, что на данный момент существует очень мало общедоступных средств анализа защищенности веб-приложений, сочетающих в себе статический и динамический анализ. Кроме того, данный инструмент построен с использованием архитектуры, которая позволяет легко добавлять поддержку других языков программирования, а также поддержку различных веб-фреймворков, на базе которых могут быть построены анализируемые веб-приложения.

## ПЕРЕЧЕНЬ ГРАФИЧЕСКИХ МАТЕРИАЛОВ

№ п/п	Наименование плаката	Количество листов	Обозначение	Формат
1	Титульный лист	1	ВКР.20.КТсо5-5.100503.02	слайд
2	Перечень решаемых задач	1	ВКР.20.КТсо5-5.100503.02	слайд
3	Модели представления исходного кода	2	ВКР.20.КТсо5-5.100503.02	слайд
4	Генератор абстрактного синтаксического дерева	1	ВКР.20.КТсо5-5.100503.02	слайд
5	Генератор графа потока управления	1	ВКР.20.КТсо5-5.100503.02	слайд
6	Генератор графа потока данных	1	ВКР.20.КТсо5-5.100503.02	слайд
7	Taint-анализ	2	ВКР.20.КТсо5-5.100503.02	слайд
8	Динамический анализатор	4	ВКР.20.КТсо5-5.100503.02	слайд
9	Безопасность человеко- машинного взаимодействия	1	ВКР.20.КТсо5-5.100503.02	слайд
10	Заключение	1	ВКР.20.КТсо5-5.100503.02	слайд

## ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ПО – программное обеспечение

ИБ – информационная безопасность

ОС – операционная система

БД – база данных

DevOps – Development Operation - набор практик для повышения эффективности процессов разработки и эксплуатации ПО

SAST – Static Application Security Testing – статическое тестирование защищенности приложений

OSA – Open Source Analysis – анализ компонент с открытым исходным кодом

DAST – Dynamic Application Security Testing – динамическое тестирование защищенности приложений

CVE – Common Vulnerabilities and Exposures – база данных общеизвестных уязвимостей информационной безопасности

AST – Abstract Syntax Tree – абстрактное синтаксическое дерево

CFG – Control Flow Graph – граф потока управления

DFG – Data Flow Graph – граф потока данных

PDG – Program Dependency Graph – граф зависимостей программы

HTTP – HyperText Transfer Protocol – сетевой протокол передачи гипертекста

HTML – HyperText Markup Language – язык гипертекстовой разметки

XSS – Cross-Site Scripting – межсайтовый скриптинг

SQL – Structural Query Language – язык запросов для работы с базой данных

MVC – Model-View-Controller – паттерн проектирования модель-представление-контроллер

JSP – Java Server Pages – страницы Java-сервера, динамически генерируемые веб-сервером для пользователей



## ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ .....	10
АНАЛИЗ ТЕХНИЧЕСКОГО ЗАДАНИЯ .....	12
1 Обзор существующих практик обеспечения ИБ при разработке ПО .....	14
1.1 Статический анализ кода (SAST) .....	14
1.2 Анализ Open Source .....	16
1.3 Динамический анализ (DAST) .....	17
2 Используемые технологии .....	20
2.1 Python .....	20
2.2 ANTLR .....	21
2.3 Flask .....	23
2.4 OrientDB .....	23
2.5 Apache Tinkerpop Gremlin .....	24
2.6 Redis .....	25
3 Модели представления исходного кода .....	26
3.1 Абстрактное синтаксическое дерево .....	26
3.2 Граф потока управления .....	28
3.3 Граф зависимостей программы .....	29
3.4 Граф свойств кода .....	30
4 Уязвимости веб-приложений .....	32
4.1 Межсайтовый скриптинг .....	32
4.2 SQL-инъекции .....	33
4.3 Внедрение команд ОС .....	34
5 Разработка модулей статического анализа .....	36
5.1 Грамматика программирования языка Java .....	36
5.2 Структура данных для хранения графов и их визуализация .....	38
5.4 Генерация графа потока управления .....	41
5.5 Извлечение информации о Java-классах .....	43

5.6 Построение графа потока данных .....	44
5.7 Использование графовой базы данных OrientDB .....	46
5.8 Taint-анализ .....	47
5.9 Веб-интерфейс для навигации по графическим представлениям AST, CFG и DFG ..	48
6 Разработка модулей динамического анализа .....	51
6.1 Извлечение точек входа веб-приложения .....	51
6.2. Формирование списка атакуемых точек входа .....	54
6.3. Подготовка контрольных точек.....	55
6.4. Инструментирование исходного кода .....	56
6.5. Проведение атаки.....	57
6.6. Пользовательский модуль динамического анализа.....	58
7 Тестирование работы разработанного инструмента .....	61
8 Безопасность человеко-машинного взаимодействия .....	63
8.1 Особенности функционального назначения объекта .....	63
8.2 Описание процесса эксплуатации объекта.....	63
8.3 Оценка эргономичности пользовательского интерфейса .....	67
8.4 Оценка напряженности процесса эксплуатации объекта .....	71
8.5 Разработка мер профилактики и повышения безопасности человеко-машинного взаимодействия .....	74
9 Технико-экономическое обоснование .....	75
9.1 Обоснование необходимости и актуальности разработки.....	75
9.2 Обоснование выбора аналога для сравнения .....	76
9.3 Сопоставление информационно-технических параметров аналога и разработки .....	77
9.4 Экономическая оценка разработки ПП .....	79
9.5 Расчет себестоимости ПП .....	80
9.6 Расчет цены ПП.....	83
9.7 Итоговое заключение по ТЭО .....	84
ЗАКЛЮЧЕНИЕ.....	86
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	87

ПРИЛОЖЕНИЕ А .....	89
ПРИЛОЖЕНИЕ Б .....	93
ПРИЛОЖЕНИЕ В.....	95
ПРИЛОЖЕНИЕ Г .....	97
ПРИЛОЖЕНИЕ Д.....	101
ПРИЛОЖЕНИЕ Е.....	106

## ВВЕДЕНИЕ

Проблема безопасной разработки программного обеспечения стоит достаточно остро. От того, что продукт будет содержать дефект безопасности, компания-владелец может понести как финансовые, так и репутационные потери. Также не стоит забывать, что чем раньше выявлена ошибка, тем меньше стоимость ее исправления. Следовательно, возникает задача поиска инструментов и методик по снижению вероятности появления дефектов безопасности и повышения качества кода приложения. Таких методик и инструментов существует уже немало, но популярных всего два.

Первая – жизненный цикл безопасной разработки (англ. Security Development Lifecycle) – концепция разработки, заключающаяся в формировании требований к приложению, безопасном программировании, тестировании, сертификации, эксплуатации и обновлении [1]. SDL – это процесс, который позволяет поддерживать необходимый уровень безопасности системы на этапе разработки, а затем на протяжении всего срока эксплуатации. Эта концепция фокусируется на обеспечении безопасности разрабатываемого приложения, идентификации рисков и управлении ими.

SDL – это процесс, который позволяет убедиться в необходимом уровне безопасности разрабатываемой системы. SDL базируется на основе практик направленных на обучение команды, подготовку отчетности и непосредственные действия, связанные с анализом безопасности разрабатываемой системы и имплементацией механизмов направленных на улучшение безопасности. Эти практики в виде конкретных шагов легко ложатся на привычный спиральный цикл разработки программного обеспечения.

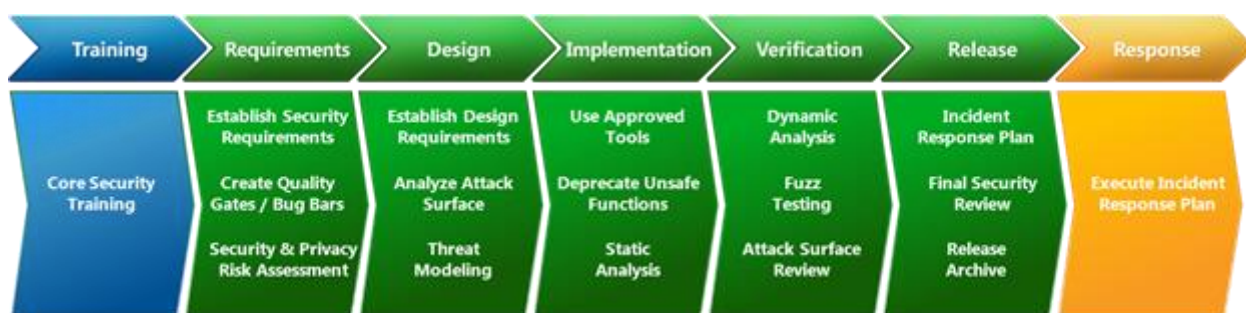


Рисунок 0.1 – Этапы жизненного цикла безопасной разработки

Вторая методология – DevSecOps. Если DevOps – методология, нацеленная на взаимодействие программистов и системных администраторов, которые в тесном взаимодействии разрабатывают продукт, то DevSecOps – это развитие концепции DevOps,

где помимо автоматизации затрагиваются вопросы обеспечения качества и надёжности кода.

В случае с DevSecOps речь идет о попытке автоматизировать основные задачи безопасности, внедряя контроль этих процессов на раннем этапе DevOps. Этот подход выгодно отличается от того, что было принято до DevSecOps – контроль безопасности являлся заключительным процессом и осуществлялся в конце разработки.

Преимущества DevSecOps выделить просто – автоматизация процессов с самого начала позволяет снизить вероятность неправильного администрирования и ошибок, которые часто приводят к простоям или открывают поверхность для атак. Также автоматизация избавляет ИБ-специалистов от необходимости настраивать консоли вручную.

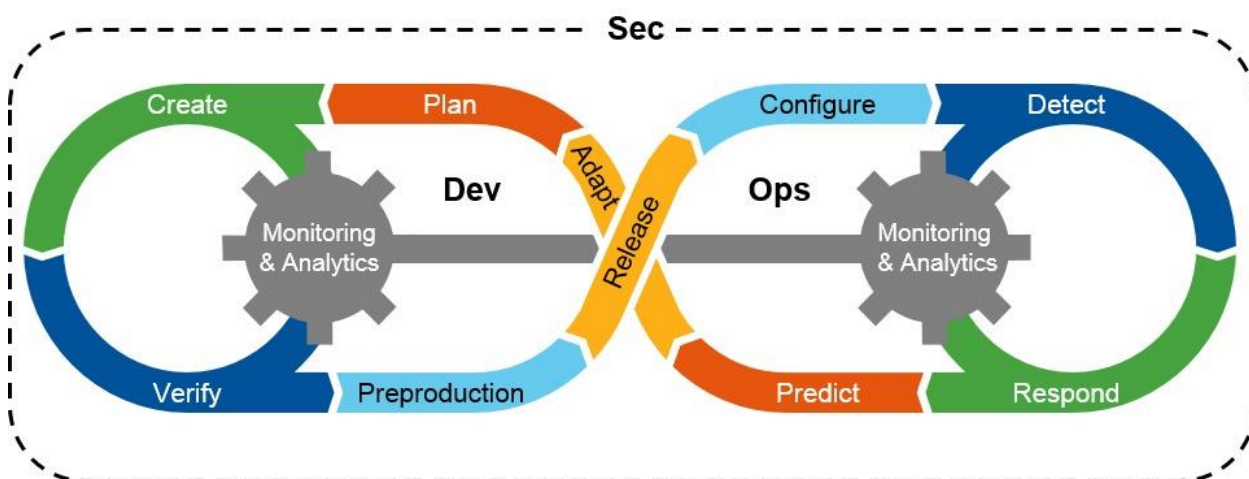


Рисунок 0.2 – Этапы методологии DevSecOps

Также эксперты отмечают, что одно из основных преимуществ DevSecOps заключается в том, что за безопасность отвечает каждая команда, участвующая в разработке. Вследствие этого уменьшается общая вероятность появления дефектов безопасности.

Что в случае с SDL, что и в случае с DevSecOps, создаются и применяются специальные инструменты, направленные на повышение безопасности на разных этапах разработки программного продукта. Эти инструменты различаются как по своему назначению, так и по доступности (коммерческие и некоммерческие). Задачей выпускной квалификационной работы является разработка инструмента способного эффективно интегрироваться в процессы двух уже упомянутых методологий безопасной разработки и заранее обнаруживать опасные уязвимости, такие как межсайтовый скриптинг, SQL-инъекции и инъекции команд ОС.

## АНАЛИЗ ТЕХНИЧЕСКОГО ЗАДАНИЯ

Задачей ВКР является разработка средства гибридного тестирования защищенности веб-приложений на основе графа свойств кода. Результатом работы должна быть полностью функциональная система, которая последовательно выполняет статический и динамический анализ программного проекта с исходным кодом на языке программирования Java и обнаруживает такие опасные уязвимости как SQL-инъекции, инъекции команд ОС и HTML-инъекции (XSS-уязвимости).

При решении поставленной задачи необходимо:

- изучить общие принципы, лежащие в основе построения инструментов статического анализа,
- разработать модуль для построения синтаксической модели (абстрактного синтаксического дерева),
- разработать модуль для построения семантической модели (графы потока управления и потоков данных),
- разработать веб-приложение для удобного отображения синтаксической и семантической моделей,
- разработать модуль для проведения Taint-анализа,
- разработать модуль инструментирования исходного кода,
- разработать модуль динамического тестирования веб-приложения.

При проектировании программного комплекса необходимо тщательно продумать оптимальную архитектуру. В качестве представления исходного кода была выбрана концепция графов свойств кода. Наиболее удачным является выбор следующих технологий:

- генератор синтаксических анализаторов ANTLR v4 (для построения моделей исходного кода),
- веб-фреймворк Flask на языке программирования Python (для разработки веб-приложения для навигации по моделям исходного кода),
- система управления графо-ориентированными базами данных OrientDB (в качестве хранилища моделей в виде графов),
- Apache Tinkerpop Gremlin (в качестве языка обхода графов),
- база данных типа ключ-значение Redis (в качестве хранения промежуточных значений для модуля динамического тестирования).

В результате работы должна быть разработана и программно реализовано средство гибридного тестирования защищенности веб-приложений на основе графа свойств кода, а также разработан удобный интерфейс работы с этим инструментом.

## **1 Обзор существующих практик обеспечения ИБ при разработке ПО**

Сейчас принято использовать следующие инструментальные средства в процессе жизненного цикла ПО для обеспечения ИБ:

- обычные виды тестирования (unit-тестирование, интеграционное тестирование, нагрузочное тестирование), которыми обычно занимаются QA-инженеры,
- статический анализ кода (SAST),
- контроль состава компонент ПО (OSA),
- все виды динамического анализа (DAST, IAST, BAST).

Важно понимать, что нет идеального средства тестирования ПО. Для разных классов программного обеспечения разные методики будут давать разные результаты. Высокого качества программы можно добиться, только сочетая различные методики.

Обычные методы тестирования могут выступать как подготовительные меры обеспечения ИБ. Прежде всего, они предназначены для повышения качества кода и гарантии того, что разрабатываемое приложение работает так, как от него требуется. Однако unit- и другие тесты могут использоваться для «отлова» довольно простых дефектов безопасности. Остальные инструменты требуют более пристального рассмотрения.

### **1.1 Статический анализ кода (SAST)**

Статический анализ кода это процесс выявления ошибок и недочетов в исходном коде программ. Под статическим анализом кода обычно понимается запуск инструментов статического анализа кода, которые пытаются найти потенциальные уязвимости в статическом состоянии, а не в состоянии выполнения анализируемого приложения. Они не проверяют, соответствует ли разрабатываемое приложение спецификации (требованиям заказчика). Вместо этого, они ищут нарушения различных практик программирования. Например, разыменованное нулевого указателя или переполнение массива. Средства статического анализа также могут находить «мертвый» код, который обычно присутствует в одной из веток условного if/else-блока и не должен исполняться из условия, которое всегда будет вычисляться одинаково [2].

Существуют различные технологии, которые могут лежать в основе работы инструментов статического анализа:



- сопоставление с шаблоном (англ. pattern-based analysis): поиск мест в исходном коде, которые похожи на известные шаблоны кода с ошибкой, на основе представления кода как абстрактного синтаксического дерева,
- вывод типов (англ. type inference): анализатор получает полную информацию обо всех переменных и выражениях, встречающихся в программе,
- символьное выполнение (англ. symbolic execution): без фактического выполнения с конкретными входными значениями вычисление значения переменных, которые могут приводить к нарушению правильной работы программы,
- анализ потока данных (англ. data-flow analysis): отслеживание того, как данные, которые были где-то определены, используются в различных частях программы.

У инструментов статического анализа есть свои преимущества. Полное покрытие кода. Статические анализаторы проверяют даже те фрагменты кода, которые получают управление крайне редко. Такие участки кода, как правило, не удастся протестировать другими методами. Это позволяет находить дефекты в обработчиках редких ситуаций, в обработчиках ошибок или в системе логирования.

Статический анализ не зависит от используемого компилятора и среды, в которой будет выполняться скомпилированная программа. Это позволяет находить скрытые ошибки, которые могут проявить себя только через несколько лет. Например, это ошибки неопределенного поведения. Такие ошибки могут проявить себя при смене версии компилятора или при использовании других опций командной строки для оптимизации кода.

Можно легко и быстро обнаруживать опечатки и последствия использования приема «копировать-вставить» (англ. Copy-Paste). Как правило, нахождение этих ошибок другими способами является крайне неэффективной тратой времени и усилий. Обидно после часа отладки обнаружить, что ошибка заключается в выражении вида «strcmp(A, A)». Обсуждая типовые ошибки, про такие ляпы, как правило, не вспоминают. Но на практике на их выявление тратится существенное время.

Однако существуют и недостатки. Статический анализ, как правило, слаб в диагностике утечек памяти и параллельных ошибок. Чтобы выявлять подобные ошибки, фактически необходимо виртуально выполнить часть программы. Это крайне сложно реализовать. Также подобные алгоритмы требуют очень много памяти и процессорного времени. Как правило, статические анализаторы ограничиваются диагностикой простых случаев. Более эффективным способом выявления утечек памяти и параллельных ошибок является использование инструментов динамического анализа.

Программа статического анализа предупреждает о подозрительных местах. Это значит, что на самом деле код, может быть совершенно корректен. Это называется ложнопозитивными срабатываниями или ошибками I рода. Понять, указывает анализатор на ошибку или выдал ложное срабатывание, может только программист. Необходимость просматривать ложные срабатывания отнимает рабочее время и ослабляет внимание к тем участкам кода, где в действительности содержатся ошибки.

Кроме ложноположительных срабатываний существует также класс ошибок, которые называются ошибками II рода. Это значит, что анализатор не выдал сообщение об ошибке, которая на самом деле присутствует. Связано это с тем, что для статического анализатора еще остается большой круг уязвимостей безопасности, которые очень трудно найти автоматически: проблемы аутентификации, проблемы контроля доступа, небезопасное использование криптографии и т.д. Современный уровень развития позволяет таким инструментам автоматически находить только относительно небольшой процент дефектов безопасности приложения. Однако инструменты этого типа продолжают развиваться.

Часто не удастся найти проблемы конфигурации, так как в большинстве случаев они не являются частью кода.

Интерфейс у инструментов статического тестирования защищенности бывает обычно двух типов: консольный и в качестве плагина в среду разработки. Первый вариант самый простой и в то же время удобный, так как его можно легко встроить в процесс непрерывной интеграции и непрерывной доставки. У второго есть очень важное преимущество: так как именно на этапе разработки появляется большое количество дефектов, то очень удобно будет, если проверка будет происходить уже, а не на более поздних этапах тестирования.

## **1.2 Анализ Open Source**

Инструменты этого типа в основном реализуют две функции: поиск уязвимостей в библиотеках и анализ лицензионной чистоты.

Поиск уязвимостей в библиотеках может проходить как на этапе разработки ПО и выбора необходимых библиотек, так и на поздних этапах, когда приложение уже находится в промышленной среде и необходим постоянный мониторинг на предмет внезапно появившихся публичных уязвимостей. Поиск обычно осуществляется по публичному реестру CVE и баг-трекерам. Если одна из библиотек уязвима, то инструмент

выдаст соответствующее предупреждение и предложит использовать другую версию без уязвимостей.

Анализ лицензионной чистоты необходим, так как не существует одной лишь лицензии для открытого ПО. Одни лицензии позволяют свободно использовать исходный код в коммерческих приложениях. Другие же обязывают публиковать исходный код приложения, если оно использует исходный код с этим типом лицензии.

Инструменты OSA могут предоставлять различные возможности: поддержка различных языков программирования и систем сборки, мониторинг компонент в промышленной среде, анализ Docker-контейнеров и др. Тем не менее, при выборе инструмента стоит обратить внимание на его ключевые характеристики: видимость, отчетность и предлагаемые меры исправления.

Хорошая система анализа с открытым исходным кодом дает вам полную видимость всех компонентов с открытым исходным кодом. Можно думать об этом как об инвентаризации. Важно понимать структуру компонентов с открытым исходным кодом, если необходимо управлять, обновлять и защищать их.

Типы отчетов, генерируемых инструментом, являются ключевыми. Среди этих типов можно выделить: уязвимости безопасности, меры по исправлению и служебная информация, необходимая для проведения инвентаризации инфраструктуры разрабатываемого проекта.

После того, как были определены уязвимости с открытым исходным кодом, необходимо их исправить. Если количество найденных известных уязвимостей оказалось большим, то нужен эффективный способ организации устранения этих угроз безопасности. Инструмент должен позволять отфильтровать оповещения с низким приоритетом. Также средство OSA должно автоматически обновлять компоненты с открытым исходным кодом, так как внедрение новых версий в большинстве случаев требует времени.

### **1.3 Динамический анализ (DAST)**

Средство динамического тестирования безопасности приложений (DAST) – это программа, которая взаимодействует с веб-приложением через веб-интерфейс для выявления потенциальных уязвимостей безопасности в веб-приложении и архитектурных недостатков. Оно выполняет тестирование методом черного ящика. В отличие от статических инструментов тестирования безопасности приложений, инструменты DAST

не имеют доступа к исходному коду и, следовательно, обнаруживают уязвимости, имитируя атаки.

Инструменты DAST позволяют выполнять сложные сканирования, обнаруживать уязвимости с минимальным взаимодействием с пользователем после предварительной настройки имени хоста, параметров сканирования и учетных данных для аутентификации. Эти инструменты будут пытаться обнаружить уязвимости в URI-параметрах, заголовках, фрагментах, HTTP-методах (GET / POST / PUT), а также провести DOM-инъекцию.

Инструменты DAST облегчают автоматическую проверку веб-приложения, выявляя уязвимости безопасности, и должны соответствовать различным нормативным требованиям. Сканеры веб-приложений могут искать широкий спектр уязвимостей, таких как проверка ввода: например, межсайтовый скриптинг и SQL-инъекции, конкретные проблемы приложения и ошибки конфигурации сервера.

Преимущества у инструментов динамического анализа несколько. Эти инструменты могут обнаруживать уязвимости в финальных версиях продукта перед его отправкой в промышленную среду. Сканеры, атакуя приложение, имитируют поведение злоумышленника, тем самым позволяя выявлять входные данные, которые не являются частью ожидаемого набора входных данных.

Как инструмент динамического тестирования, веб-сканеры не зависят от языка.

Однако существуют и недостатки. При сканировании с использованием инструмента DAST данные могут быть перезаписаны, или вредоносные данные внедрены в сайт субъекта. Сайты следует сканировать в полупроизводственной, но в непроизводственной среде, чтобы обеспечить точные результаты при защите данных в производственной среде.

Поскольку инструмент реализует метод динамического тестирования, он не может покрыть 100% исходного кода приложения, а, следовательно, и самого приложения. Тестировщик на проникновение должен самостоятельно изучить веб-приложение и составить представление о поверхности атаки, чтобы узнать, правильно ли настроен инструмент.

Инструмент не может реализовать все варианты атак для данной уязвимости. Таким образом, инструменты обычно имеют predetermined вектора атак и не генерируют полезные нагрузки в зависимости от тестируемого веб-приложения. Некоторые инструменты также весьма ограничены в понимании поведения приложений с динамическим содержанием, таким как JavaScript и Flash.

Стоит учесть, что инструменты DAST можно использовать как аналог нагрузочного тестирования. Для этого можно включить динамический сканер в 10-15 потоков и проанализировать надежность тестируемого веб-приложения

Существует еще один класс инструментов DAST – инструменты интерактивного тестирования безопасности приложений (IAST) [3]. Технология IAST анализирует приложение изнутри во время его работы. IAST отслеживает выполнение кода в памяти и ищет конкретные события, которые могут привести к уязвимости. Далее эти события анализируются с целью проверить, не произошло ли каких-либо нарушений.

## 2 Используемые технологии

### 2.1 Python

Python - популярный язык программирования, используемый как для разработки самостоятельных программ, так и для создания прикладных сценариев в самых разных областях применения. Это мощный, переносимый, простой в использовании и свободно распространяемый язык. Программисты, работающие в самых разных областях, считают, что ориентация Python на эффективность разработки и высокое качество программного обеспечения дает ему стратегическое преимущество, как в маленьких, так и в крупных проектах.

Python отличается удобочитаемостью, ясностью и более высоким качеством, по сравнению с другими скриптовыми языками. Благодаря тому, что код на языке Python читается легче, становится гораздо проще его использовать и обслуживать. Кроме того, Python поддерживает парадигму объектно-ориентированного кода, что позволяет разрабатывать большие проекты, которые обычно не характерны для языков сценариев.

То, что Python не является компилируемым и строго типизированным языком, таким как Java, C/C++, во много раз повышается скорость разработки. Код на языке Python в три или даже пять раз может быть меньше эквивалентного кода на языке C++ или Java. Это означает огромное уменьшение количества трудозатрат программиста не только на написание кода, но и на его отладку.

Интерпретаторы Python выпущены под многие платформы и операционные системы, что делает программы на языке Python переносимыми или кросс-платформенными. Да, не весь код можно просто так взять и перенести на другую операционную систему, однако это не будет стоить больших усилий.

Стандартная библиотека языка Python обладает колоссальным количеством возможностей, начиная от поиска текста по шаблону с использованием регулярных выражений и заканчивая сетевыми функциями. Кроме того, существует огромное количество сторонних библиотек, поддерживаемых Python-сообществом. Это и веб-фреймворки для создания веб-приложений, и библиотеки, предоставляющие доступ к алгоритмам машинного обучения, а также инструменты, позволяющие разрабатывать компьютерных игр.

Язык Python может выступать не только как единственный язык при написании приложения. У него есть возможность вызывать функции, написанные на языках C и C++, а также существует огромное число так называемых биндингов (англ. binding –

связующий), которые позволяют использовать его из других языковых окружений. Кроме того, существует много программ, например графический редактор Gimp, в которых есть возможность расширения своего функционала с помощью плагинов на языке Python.

Однако у языка Python есть один недостаток. Он обладает низкой скоростью выполнения по сравнению с компилируемыми языками, такими как C и C++. Несмотря на то, что Python не является чисто интерпретируемым языком и его код сначала транслируется в байт-код и затем выполняется на виртуальной машине Python VM, не создается двоичный машинный код, который уже непосредственно выполняется процессором. Но не все так печально. Для «значительных» задач, например, когда обрабатывается файл или создается графический интерфейс, программа фактически выполняется со скоростью, которую способен дать язык C. Это достигается за счет компилированного Си-кода, лежащего в недрах интерпретатора Python. Поэтому всегда, когда важна скорость выполнения, можно решить проблему путем написания скомпилированных расширений [4].

Существует две основные версии языка Python – Python 2 и Python 3. Для разработки программного продукта использовалась третья версия, так как поддержка Python 2 заканчивается в 2020 году [5].

## **2.2 ANTLR**

ANTLR v4 - это мощный генератор синтаксических анализаторов, который можно использовать для чтения, обработки, выполнения или трансляции произвольного структурированного текста или двоичных файлов. Он широко используется в научных сфере и в промышленности для создания предметно-ориентированных языков, инструментов и сред. Поиск в Твиттере использует ANTLR для анализа запросов, с более чем 2 миллиардами запросов в день. Все языки для Hive и Pig, а также системы хранения и анализа данных для Hadoop используют ANTLR. Oracle использует ANTLR в среде IDE SQL Developer и ее инструментах для миграций баз данных. Среда IDE NetBeans анализирует C++ с помощью ANTLR. Язык HQL в инфраструктуре ORM Hibernate построен с использованием ANTLR [6].

Помимо этих громких проектов, вы можете создавать различные полезные инструменты, такие как программы чтения файлов конфигурации, конвертеры legacy-кода, средства визуализации разметки вики и анализаторы JSON.

Из описания формального языка, называемого грамматикой, ANTLR генерирует синтаксический анализатор для этого языка, который может автоматически создавать деревья разбора. Они служат структурой данных оптимальной для описания синтаксиса программы, ANTLR также автоматически генерирует классы для обхода деревьев (англ. tree walkers), которые можно использовать для выполнения кода, специфичного для приложения, при посещении того или иного узла дерева.

Синтаксический анализ намного проще рассматривать, если его разбить на две похожие, но разные задачи или этапы.

Процесс группировки символов в слова или символы (токены) называется лексическим анализом или просто токенизацией. Программу, которая токенизирует ввод, называют лексером. Лексер может группировать связанные токены в классы токенов или типы токенов, такие как INT (целые числа), ID (идентификаторы), FLOAT (числа с плавающей запятой) и так далее. Лексер группирует словарные символы по типам, когда анализатор заботится только о типе, а не об отдельных символах. Токены состоят как минимум из двух частей информации: тип токена (идентифицирующий лексическую структуру) и текст, соответствующий группе символов для этого токена.

Второй этап - это работа синтаксического анализатора, который принимает на вход токены для дальнейшего распознавания структуры участка кода, например, оператора присваивания. По умолчанию сгенерированные ANTLR синтаксические анализаторы создают структуру данных, называемую деревом разбора, в котором записывается, как синтаксический анализатор распознал структуру входного предложения и его компонентных фраз. На рисунке 2.1 представлена примерная схема анализа оператора присваивания.

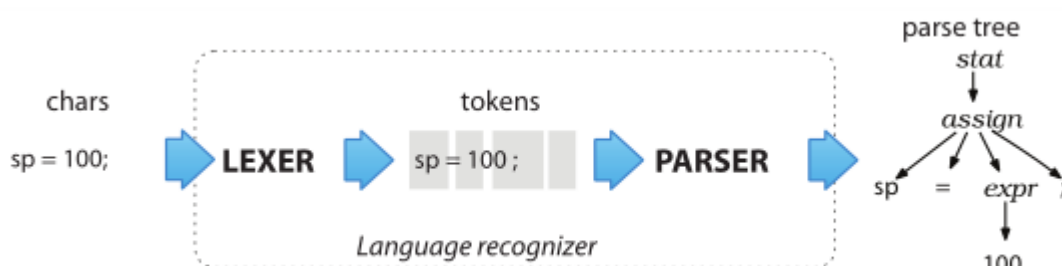


Рисунок 2.1 – Процесс распознавания инструкции языка программирования

Создавая дерево разбора, парсер предоставляет удобную структуру данных для остальной части разрабатываемого приложения. Дерево содержит полную информацию о том, как синтаксический анализатор сгруппировал символы во фразы. Деревья просты в



обработке на последующих этапах и хорошо поняты программистам. А главное, парсер может генерировать деревья разбора автоматически.

Поскольку мы задаем структуру фраз с помощью набора правил, корни поддерева дерева разбора соответствуют именам правил грамматики. Ниже частично приведен пример правила грамматики, соответствующего первому уровню поддерева оператора присваивания из рисунка 2.2.

```
assign : ID '=' expr ';' ; // match an assignment statement like "sp = 100;"
```

Рисунок 2.2 – Пример правила грамматики

## 2.3 Flask

Flask – это микрофреймворк для создания вебсайтов на языке Python. Он называется микрофреймворком, потому что основная идея его разработчиков – это сохранить ядро простым, но расширяемым. В нем нет ORM для баз данных, валидации форм и того, что есть в других веб-фреймворках, таких как Django. Однако при необходимости могут быть добавлены Flask-расширения, которые реализуют подобный функционал. Эта главная идея позволяет писать компактные веб-приложения без лишних компонентов, которые не планируется использовать [7].

Flask поставляется со встроенным веб-сервером и многочисленными инструментами отладки, включая инструмент in-browser. Поэтому вам даже не понадобится NGINX или Apache для тестирования и выявления ошибок в вашем приложении [8].

Flask настолько прост, что работа в нем позволяет опытному программисту, пишущему на Python, создавать проекты в очень сжатые сроки

## 2.4 OrientDB

OrientDB – это система управления базами данных, которая объединяет в себе возможности документо-ориентированной и графо-ориентированной БД. Даже при работы с документ-ориентированными данными взаимодействие между документами обрабатывается как в графо-ориентированной БД с определением прямых связей между записями. При этом пройти по цепочке содержимого деревьев и графов, как целиком, так

и частями, можно в считанные миллисекунды. Дополнительно поддерживается интерфейс объектно-ориентированной БД, который работает поверх документо-ориентированного слоя. Код OrientDB написан на языке Java и распространяется под лицензией Apache [9].

OrientDB имеет консольный и веб-интерфейс (OrientDB WebStudio). Веб-интерфейс обладает следующими возможностями:

- управления правами доступа и Базами Данных,
- просмотра и изменения структур данных (Классы и Ребра) и самих Данных,
- просмотра и изменения Данных в виде визуального построения графов,
- составления и выполнения запросов к БД посредством newSQL,
- составления и выполнения запросов к БД посредством HTTP REST,
- написания и выполнения Хранимых Процедур (SQL, Java Script),
- профилирования выполнения запросов к OrientDB (только в Enterprise Edition),
- управление конфигурацией и мониторинг Кластера (только в Enterprise Edition).

Кроме того, OrientDB имеет биндинги для различных языков программирования. Один из биндингов есть для языка Python – модуль PyOrient, который и будет использоваться в разрабатываемом инструменте.

## 2.5 Apache Tinkerpop Gremlin

Apache TinkerPop - это открытая среда графо-ориентированных вычислений с открытым исходным кодом. Когда система данных поддерживает TinkerPop, ее пользователи могут моделировать свою предметную область в виде графа и анализировать этот граф с помощью языка обхода графа Gremlin. Кроме того, все системы с поддержкой TinkerPop интегрируются друг с другом, что позволяет им легко расширять свои возможности, а также позволяет пользователям выбирать подходящую технологию графов для своего приложения. Иногда приложение лучше обслуживать базой данных транзакционного графа. Иногда с этим справится база данных распределенных графов с несколькими машинами. Или, возможно, приложению требуется база данных распределенных графов для запросов в реальном времени и параллельно процессор больших (графовых) данных для пакетной аналитики. Какими бы ни были требования приложения, существует много графо-ориентированных систем с поддержкой TinkerPop для удовлетворения его потребностей.

Gremlin - это язык обхода графов Apache TinkerPop. Он помогает ориентироваться в вершинах и ребрах графа. По сути, он является языком запросов для построения графо-

ориентированных баз данных, как SQL является языком запросов к реляционным базам данных. Чтобы сообщить Gremlin, как он должен «пройти» по графу (то есть, что пользователь хочет, чтобы запрос делал), нужен способ предоставить ему команды на понятном ему языке – и, конечно, этот язык называется «Gremlin» [10].

Каждый Gremlin-запрос (обход) состоит из последовательности атомарных операций – шагов. Шаг выполняет элементарную операцию над потоком данных. Каждый шаг – это либо шаг-отображение (преобразование объектов в потоке), либо шаг-фильтр (удаление объектов из потока), либо шаг побочного эффекта (вычисление статистики о потоке). Стандартная библиотека шагов Gremlin расширяет эти три фундаментальные операции, чтобы предоставить пользователям обширную коллекцию шагов, из которой они могут составить любой запрос, предназначенный для решения задач, связанных с обработкой данных.

## 2.6 Redis

Redis (от англ. remote dictionary server) – резидентная система управления базами данных класса NoSQL с открытым исходным кодом, работающая со структурами данных типа «ключ – значение». Используется как для баз данных, так и для реализации кэшей, брокеров сообщений [11].

Все данные Redis хранит в виде словаря, в котором ключи связаны со своими значениями. Одно из ключевых отличий Redis от других хранилищ данных заключается в том, что значения этих ключей не ограничиваются строками. Поддерживаются следующие абстрактные типы данных: строки, списки, множества, хеш-таблицы, упорядоченные множества. Тип данных значения определяет, какие операции (команды) доступны для него; поддерживаются такие высокоуровневые операции, как объединение и разность множеств, сортировка наборов.

Дополнительные функции, такие как репликация, постоянство и сегментирование на стороне клиента, позволяют Redis масштабироваться от удобного способа создания прототипа системы до сотен гигабайт данных и миллионов запросов в секунду.

Для разработчиков уже создано много библиотек по работе с Redis. В разрабатываемом инструменте применяется библиотека для языка программирования Scala.

### 3 Модели представления исходного кода

Для того чтобы проводить анализ исходного кода приложения, недостаточно одного лишь его представления в виде текста. Необходимы более структурированные представления, или модели. Таких моделей существует несколько и каждая из них служит для решения своих задач. Разработчики инструментов статического анализа выделяют следующие модели:

- абстрактное синтаксическое дерево,
- граф зависимостей,
- граф потока управления,
- граф зависимостей программы,
- граф свойств кода.

Рассмотрим каждое из этих представлений более подробно.

#### 3.1 Абстрактное синтаксическое дерево

В основе абстрактного синтаксического дерева лежит одноименная и одна из наиболее широко распространённых структур данных в информатике, эмулирующая древовидную структуру в виде набора связанных узлов – дерево. Дерево состоит из узлов, соединенных ребрами.

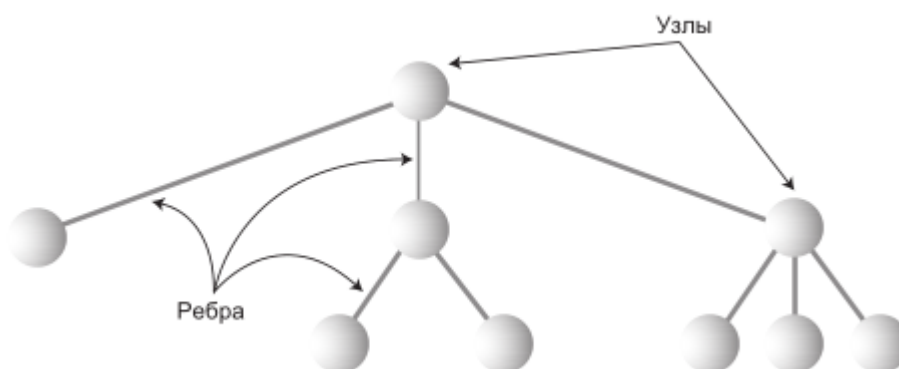


Рисунок 3.1 – Графическое представление структуры данных «дерево»

Абстрактное синтаксическое дерево – конечное помеченное ориентированное дерево, в котором внутренние вершины сопоставлены (помечены) с операторами языка

программирования, а листья – с соответствующими операндами. Таким образом, листья являются пустыми операторами и представляют только переменные и константы [12].

Абстрактные синтаксические деревья обычно являются первыми среди промежуточных представлений, производимых парсерами компиляторов, и, следовательно, выступают фундаментом для генерации многих других представлений кода.

Например, на рисунке 3.2 показано абстрактное синтаксическое дерево для алгоритма Евклида, написанного на языке программирования Pascal и приведённого ниже:

```
while b ≠ 0
  if a > b
    a := a - b
  else
    b := b - a
return a
```

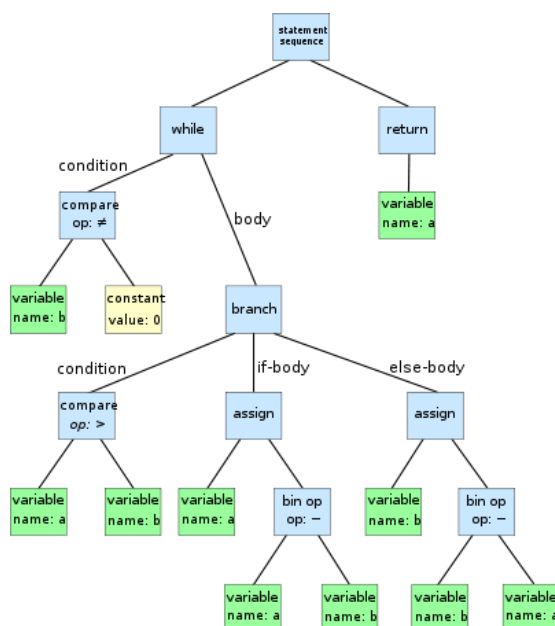


Рисунок 3.2 – Пример абстрактного синтаксического дерева

Эти деревья точно кодируют то, как операторы и выражения вложены друг в друга внутри программы. В отличие от деревьев разбора с их избыточностью абстрактные синтаксические деревья больше не отображают конкретный синтаксис языка программирования, а хранят только существенную информацию.

Хотя они хорошо подходят для простых преобразований кода и используются для идентификации семантически подобного кода, они не применимы для более продвинутого

анализа кода, такого как обнаружение мертвого кода или неинициализированных переменных. Причина этого недостатка состоит в том, что абстрактные синтаксические деревья не хранят информацию ни о передаче потока управления, ни о потоках данных.

### 3.2 Граф потока управления

Граф потока управления (CFG) явно отображает порядок, в котором выполняются инструкции (операторы), а также условия (предикаты), которые должны быть соблюдены для выбора определенного пути исполнения. С этой целью операторы и предикаты представляются узлами, которые соединены направленными ребрами для отображения передачи управления. Так как из узлов могут выходить больше одного ребра, то необходимо всем ребрам назначать соответствующие метки «True», «False» или «EPS» (epsilon). В частности, узел-оператор имеет одно исходящее ребро, помеченное как «EPS», тогда как узел предиката имеет два исходящих ребра, соответствующих истинному или ложному значению предиката. Пример графа потока управления представлен на рисунке 3.3.

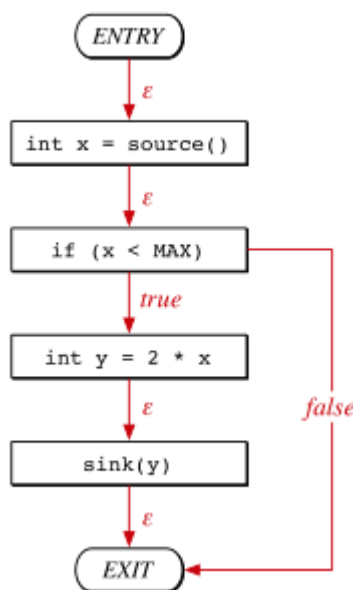


Рисунок 3.3 – Граф потока управления

Графы потока управления используются при решении разных задач в контексте безопасности, например обнаружение различных модификаций известного вредоносного ПО [13] и контроль выполнения работы инструментов фаззинг-тестирования [14]. Кроме того, такое представление кода стало стандартным при решении задач обратного инжиниринга, чтобы помочь в понимании программ. Однако, хотя они содержат

информацию о передаче управления внутри программы, графы потоков управления не могут предоставить информацию о потоках данных. В частности, для анализа уязвимостей это означает, что графы потоков управления не могут быть легко использованы для идентификации инструкций, которые обрабатывают данные и на которые воздействует злоумышленник.

### 3.3 Граф зависимостей программы

Граф зависимостей программы был изначально разработан для проведения «слайсинга». Этот вид анализа нужен для определения всех инструкций и предикатов, которые влияют на значение переменной в каком-то определенном операторе. Граф зависимостей программы явно представляет зависимости между операторами и предикатами. В частности, граф строится с использованием двух типов ребер: ребер зависимостей данных, отражающих влияние одной переменной на другую, и ребер передачи управления, соответствующих влиянию предикатов на значения переменных. Ребра графа зависимостей программы можно рассчитать из графа потока управления, сначала определив общий набор переменных и набор переменных, используемых каждым оператором, и вычислить достижимость определений каждой такой переменной до каждого оператора и предиката. Такая операция является стандартной задачей при проектировании компиляторов.

На рисунке 3.4 показан график зависимости программы для примера кода, граф потока управления которого показан на рисунке 3.3. Стоит обратить внимание на то, что ребра зависимостей управления – это не просто ребра потоков управления и, в частности, что порядок, в котором выполняются операторы, больше не может быть определен из графа. Однако зависимости между операторами и предикатами отчетливо видны.

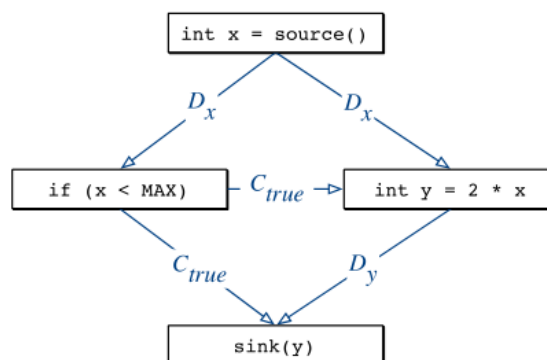


Рисунок 3.4 – Граф зависимостей программы

### 3.4 Граф свойств кода

Каждая из перечисленных выше моделей обеспечивает уникальный взгляд на исходный код программы, подчеркивая различные аспекты базовой программы. Чтобы объединить эти представления в единое представление, в том числе для обнаружения уязвимостей, используется концепция графов свойств, которое является базовым представлением структурированных данных во многих графовых базах данных, как, например, ArangoDB, Neo4J и OrientDB [15].

Граф свойств это направленный, с метками ребер, атрибутированный мультиграф, у которого ребра имеют метки, и который определяется следующей формулой (3.1):

$$G = (V, E, \lambda, \mu), \quad (3.1)$$

где  $V$  – множество вершин,  $E$  – множество направленных ребер,  $\lambda: E \rightarrow \Sigma$  функция присвоения меток из алфавита  $\Sigma$  каждому ребру. Свойства могут быть назначены ребрам и вершинам функцией  $\mu: (V \cup E) \times K \rightarrow S$ , где  $K$  – это множество названий свойств и  $S$  – множество значений свойств.

Рисунок 3.5 показывает простой граф свойств с четырьмя вершинами. Стоит обратить внимание, что граф свойств – мультиграф, и, следовательно, две вершины могут быть соединены несколькими ребрами, как, например, вершины А и В на рисунке 3.5. Более того, в этом примере свойство с названием  $k \in K$  присваивается каждой вершине, где только вершины А и В содержат значения свойства из множества значений свойств  $S = \{x, w\}$ . На рисунке у вершин С и D свойство  $k$  принимает значение  $\epsilon$ : это означает, что свойство  $k$  не задано.

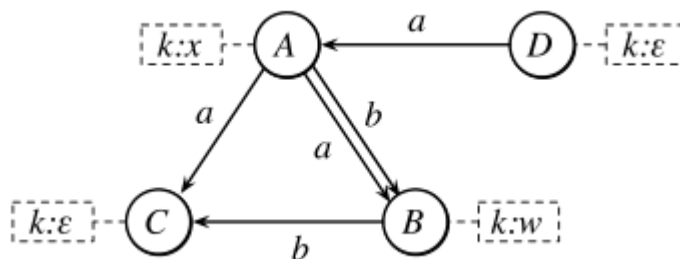


Рисунок 3.5 – Пример графа свойств. Свойства, присвоенные вершинам, помечены пунктирной линией.



Основным инструментом для извлечения информации в графах свойств являются так называемые обходы графов или короткие обходы, которые можно использовать для перемещения по ребрам графа в зависимости от меток и свойств, как вершин, так и ребер.

Перевод перечисленных ранее моделей исходного кода, а именно в абстрактного синтаксического дерева, графа потока управления и графа потока данных, в соответствующие графы свойств производится довольно просто. Граф зависимостей программы использовался частично, так как для хранения информации о передаче управления хватило графа потока управления.

Единственное представление, предлагающее подробную декомпозицию исходного кода в языковые конструкции – это AST. Следовательно, мы начинаем построение совместного представления, выражая AST как граф свойств формулой (3.2):

$$G_A = (V_A, E_A, \lambda_A, \mu_A), \quad (3.2)$$

где вершины  $V_A$  берутся из вершин дерева и ребра  $E_A$  это соответствующие ребра дерева с метками AST-ребер в соответствии с функцией меток  $\lambda$ . Дополнительно мы присваиваем каждой вершине свойство «код», используя  $\mu$  такое, что значение свойства соответствует оператору или операнду, которые представляет вершина. Также мы присваиваем свойство «строка» (номер строки) для отображения местоположения конкретной синтаксической конструкции языка. В результате ключи свойств графа –  $K_A = \{\text{код, строка}\}$ , а множество значений свойств  $S_A$  задается всеми операторами и операндами и натуральными числами.

В качестве следующего шага мы подготовим CFG для включения в совместное представление. Для этой цели мы выразим CFG как граф свойств формулой (3.3):

$$G_C = (V_C, E_C, \lambda_C, \mu_C), \quad (3.3)$$

где  $V_C$  это в основном AST-вершины с типом STMT (оператор) и PRED (предикат). Кроме того, мы определим функцию меток ребер, которая присваивает метку из множества  $\Sigma_C = \{true, false, \epsilon\}$  всем ребрам в графе свойств.

При переводе графа потока данных вершины остаются те же, что и у графа потока управления. Таким образом, граф свойств в этом случае имеет вид, представленный формулой (3.4):

$$G_D = (V_C, E_D, \lambda_D, \mu_D). \quad (3.4)$$

## **4 Уязвимости веб-приложений**

В современном мире веб-приложения стали настолько популярны, что без них сложно представить компанию из сегмента малого или крупного бизнеса. Обычно это компании, предоставляющие услуги или продающие какие-то товары. Потребители предпочитают узнавать об этих услугах и товарах онлайн, ознакомившись с ассортиментом компании на ее сайте. Сайты есть не только у частных компаний, но и у государственных структур. Это обстоятельство мотивирует владельцев веб-приложений поддерживать их на высоком технологическом уровне, который невозможен без должного внимания к защите от кибератак.

Специалисты компании Positive Technologies в ходе исследования за 2019 год [16] обнаружили, что в 19% веб-приложений есть уязвимости, позволяющие злоумышленнику получить контроль, как над самим приложением, так и над ОС сервера. Если сервер находится на периметре организации, злоумышленник может проникнуть во внутреннюю сеть компании. Наиболее распространены уязвимости связанные с недостаточной авторизацией, возможностью загрузки или чтения произвольных файлов, а также с возможностью внедрения SQL-кода. В то же время высока доля веб-приложений с уязвимостями «Межсайтовое выполнение сценариев» (Cross-Site Scripting, XSS), которая в 2019 году стала еще внушительней (88,5% против 77,9% в позапрошлом году).

Рассмотрим уязвимости, обнаружение которых будет главной целью работы разрабатываемого средства анализа защищенности веб-приложений.

### **4.1 Межсайтовый скриптинг**

Межсайтовый скриптинг – это атака, при которой злоумышленник может внедрить код JavaScript в приложение. Точнее, цель состоит в том, чтобы этот код JavaScript выполнялся в браузере потенциальной жертвы. Поскольку JavaScript имеет полный доступ к отображаемому в данный момент документу, это позволяет злоумышленнику контролировать браузер жертвы в контексте уязвимого приложения [17].

Существует три основных вида XSS-атак.

Отраженные XSS (или непостоянные) – сервер считывает данные непосредственно из HTTP-запроса и отражает их обратно в HTTP-ответе. Наиболее распространенным механизмом доставки вредоносного контента является включение его в качестве параметра в URL-адрес, который где-нибудь публикуется или отправляется по

электронной почте непосредственно жертве. Созданные таким образом URL-адреса составляют основу многих фишинговых схем, в результате чего злоумышленник убеждает жертву посетить URL-адрес, относящийся к уязвимому сайту. После того, как сайт отразит контент злоумышленника обратно к жертве, контент исполняется браузером жертвы.

Хранимые XSS (или постоянные) – приложение хранит опасные данные в базе данных, журнале посещений или другом надежном хранилище данных. Позже опасные данные впоследствии считываются обратно и участвуют в генерации динамического контента. С точки зрения злоумышленника, оптимальным местом для внедрения вредоносного контента является такое место веб-приложения, которое посещают либо многие пользователи, либо особо интересные. Интересные пользователи обычно имеют повышенные привилегии в приложении или взаимодействуют с конфиденциальными данными, которые ценны для злоумышленника. Если у одного из этих пользователей исполняется вредоносный контент, злоумышленник может выполнить привилегированные операции от имени пользователя или получить доступ к конфиденциальным данным, принадлежащим пользователю. Например, злоумышленник может внедрить XSS в сообщение журнала, которое может не обрабатываться должным образом, когда администратор просматривает журналы.

DOM-модели XSS: уязвимость возникает в коде на стороне клиента, а не на стороне серверного кода. XSS в DOM-модели представляет собой вариант как хранимой и отраженной XSS-атаки. В этой XSS-атаке вредоносная строка не обрабатывается браузером жертвы, пока настоящий JavaScript веб-сайта не выполнится [18].

## **4.2 SQL-инъекции**

SQL-инъекции - это уязвимости, в которых злоумышленник использует уязвимость в приложении для внедрения SQL-команд по своему выбору. Хотя, в зависимости от базы данных, точный синтаксис немного отличается, общая концепция одинакова для всех механизмов базы данных. Чаще всего он выполняет определение СУБД, а также извлекает таблицы с наиболее «интересными» именами (например «users»). После этого, в зависимости от привилегий, с которыми запущено уязвимое приложение, он может обратиться к защищенным частям бэкенда веб-приложения (например, прочитать файлы на стороне хоста или выполнить произвольные команды).

Различают пять основных классов SQL-инъекций [19].

Инъекции через UNION-запрос. Классический вариант внедрения SQL-кода, когда в уязвимый параметр передается выражение, начинающееся с "UNION ALL SELECT". Эта техника работает, когда веб-приложения напрямую возвращают результат вывода команды SELECT на страницу.

Инъекции, основанные на сообщениях об ошибках. В случае этой атаки сканер заменяет или добавляет в уязвимый параметр синтаксически неправильное выражение, после чего парсит HTTP-ответ (заголовки и тело) в поиске ошибок СУБД, в которых содержалась бы заранее известная инъектированная последовательность символов и где-то «рядом» вывод на интересующий нас подзапрос. Эта техника работает только тогда, когда веб-приложение по каким-то причинам (чаще всего в целях отладки) раскрывает ошибки СУБД.

Инъекции через стек запросов. Сканер проверяет, поддерживает ли веб-приложение последовательные запросы, и, если они выполняются, добавляет в уязвимый параметр HTTP-запроса точку с запятой «;» и следом внедряемый SQL-запрос. Этот прием в основном используется для внедрения SQL-команд, отличных от SELECT, например, для манипуляции данными (с помощью INSERT или DELETE).

Обычные слепые инъекции (boolean-based blind). Реализация так называемой слепой инъекции: данные из БД в «чистом» виде уязвимым веб-приложением нигде не возвращаются. Прием также называется дедуктивным. При атаке добавляется в уязвимый параметр HTTP-запроса синтаксически правильно составленное выражение, содержащее подзапрос SELECT (или любую другую команду для получения выборки из базы данных). Для каждого полученного HTTP-ответа выполняется сравнение заголовков и тела страницы с ответом на изначальный запрос — таким образом, утилита может символ за символом определить вывод внедренного SQL-выражения.

Слепая инъекция на основе задержек времени. Полностью слепая инъекция. Точно так же как и в предыдущем случае, атакующий работает с уязвимым параметром. Но в этом случае в подзапросе использует одну из команд, которая приводит к паузе работы СУБД на определенное количество секунд (например, с помощью команды SLEEP).

### **4.3 Внедрение команд ОС**

Внедрение команд — это тип атаки, целью которого является несанкционированное выполнение команд операционной системы на удаленном сервере через уязвимое веб-приложение.

Большинство веб-серверов поддерживают функциональность, которая позволяет данным взаимодействовать с серверной операционной системой. Эти функции могут быть полезными при создании приложений, возможности которых уже реализованы в ОС. В результате не нужно писать новую программу, можно просто передавать команду с дополнительными опциями в систему и отображать результаты на веб-сайте [20].

Эффективно используя эту уязвимость, атакующий может получить чувствительные данные, такие как:

- файлы паролей операционной системы,
- конфигурационные файлы операционной системы,
- исходный код приложения.

## 5 Разработка модулей статического анализа

### 5.1 Грамматика программирования языка Java

Файлы грамматики языка Java были взяты из открытого github-репозитория проекта ANTLR и немного доработаны для того, чтобы было удобней обрабатывать правила операторов (statements) и правила выражений (expressions).

Разработка проекта велась в интегрированной среде разработки PyCharm Community, у которой есть сторонний плагин, упрощающий работу с инструментом ANTLR. С помощью этого плагина можно настроить автоматическую генерацию лексера и синтаксического анализатора, а также «Слушателя» (Listener, Walker) и «Посетителя» (Visitor). Слушатель и Посетитель – это базовые классы, которые определяют интерфейс обхода дерева разбора. Разработчик, наследуясь от этих классов, переопределяет их методы, тем самым реализует логику разрабатываемого приложения. При разработке необходимо выбрать один из них, так как реализуют они обход дерева разбора различным образом.

В Посетителе для обработки потомков какого-либо узла необходимо вручную вызывать методы их обхода. При этом если родитель имеет три потомка, и вызвать методы только для двух узлов, то часть поддерева вообще не будет обработана. Обход дерева разбора Посетителем показан на рисунке 5.1.

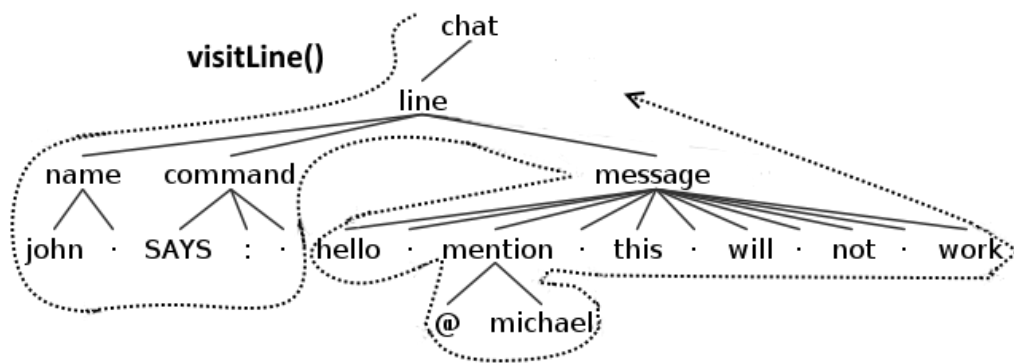


Рисунок 5.1 – Обход дерева, используя Посетитель

В Слушателе методы посещения всех потомков вызываются автоматически. Он определяет метод, вызываемый в начале посещения узла (enterNode) и метод, вызываемый после посещения узла (exitNode). Эти методы отлично подходят для реализации механизма событий. Обход дерева разбора Слушателем показан на рисунке 5.2.

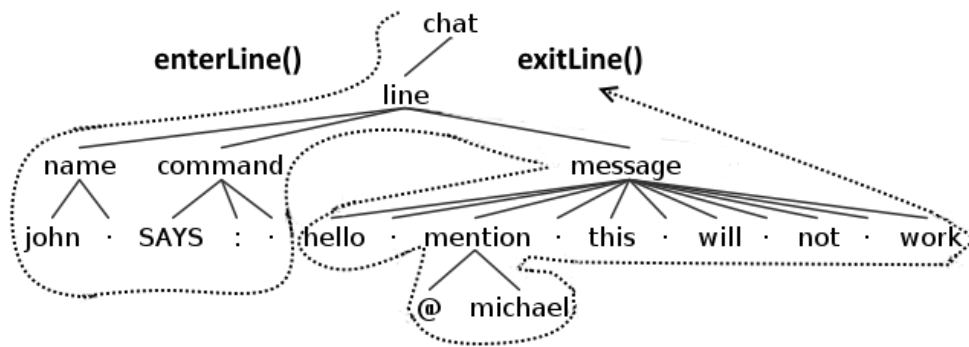


Рисунок 5.2 – Обход дерева, используя Слушатель

Для реализации построения AST, CFG и DFG из-за своей простоты и большего контроля над обходом дерева был использован паттерн Посетитель.

У плагина ANLR для IDE PyCharm есть еще одна удобная возможность – смотреть какое дерево разбора создается для определенного правила, если на вход подать строку (или целый файл) соответствующую грамматике какого-либо языка. Например, если протестировать правило `statement` и подать на вход небольшой фрагмент Java-кода (см. рисунок 5.3), то увидим представление дерева разбора как на рисунке 5.4:

```

JavaParser.g4 start rule: statement  Input File
1  if (a > 10) {
2    ... System.out.println("a > 10");
3  }
4  |

```

Рисунок 5.3 – Фрагмент Java-кода

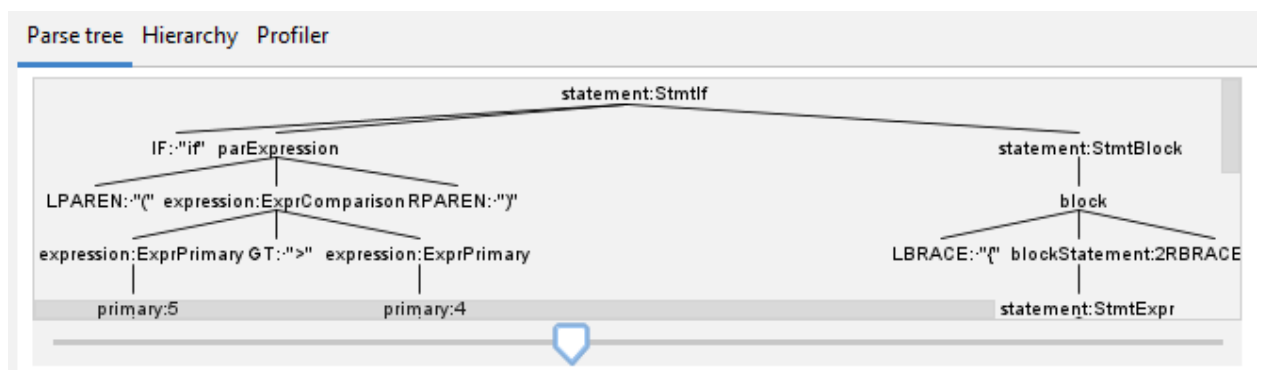


Рисунок 5.4 – Дерево разбора, выведенное с помощью ANTLR-плагина PyCharm

Это дерево разбора также можно экспортировать во внешний файл как графический файл формата PNG.

## 5.2 Структура данных для хранения графов и их визуализация

Для того чтобы создавать и хранить абстрактное синтаксическое дерево, граф потока управления и граф потока данных, необходима соответствующая структура данных, а именно ориентированный граф. В стандартной библиотеке языка Python нет готовой структуры данных, но есть сторонняя библиотека NetworkX. Несмотря на то, что этот модуль обладает обширным функционалом, способ работы с ребрами и вершинами не всегда является удобным. Поэтому было принято решение самостоятельно реализовать ориентированный граф. Вершины (узлы) будут храниться в простом списке. Хранение ребер реализовано чуть сложнее. Есть список `allEdges`, в котором будут храниться все ребра. Кроме этого списка есть два словаря `inEdges` и `outEdges` для хранения входящих и исходящих ребер соответственно. Ключом в этих словарях будет соответствующая вершина (идентификатор вершины), в которую будут входить или из которой будут выходить ребра. Значениями же будут списки ребер.

Само ребро будет представлять собой структуру данных, состоящую из трех полей: начального узла, метки и конечного узла.

Узел представляет собой структуру, у которой имеются следующие поля:

- внутренний идентификатор графа, к которому он принадлежит,
- свойства, это могут быть свойства «код» и «номер строки»,
- общий идентификатор для обеспечения связанности с другими графами.

Хранение графов реализовано с помощью простой файловой базы данных в формате JSON. Для того чтобы эффективно работать с такой базой данных, используется Python-модуль – `pickleDB`. В базе данных принято решение хранить объекты, описание и название которых представлено в таблице 5.1.

Таблица 5.1 – Объекты базы данных

Имя объекта (коллекции)	Описание объекта (коллекции)
Словарь ASTs	Хранение абстрактных синтаксических деревьев. Здесь ключ – это так называемое «квалификационное имя» (англ. <i>qualified name</i> ), состоящее из полного названия пакета, который пишется вначале Java-файла и имени этого файла без расширения. Значения – ориентированные графы, представляющие соответствующие абстрактные синтаксические деревья.



Продолжение таблицы 5.1

Имя объекта (коллекции)	Описание объекта (коллекции)
Словарь CFGs	Хранение графов потока управления. Здесь ключ – это также «квалификационное имя», состоящее из полного названия пакета, класса метода и имени метода. Значения – соответствующие графы потоков управления.
Словарь DFGs	Хранение графов потоков данных. По структуре аналогичен предыдущему словарию, так как в основе графа потока данных лежит граф потоков управления.
Словарь JavaClasses	Хранение информации о Java-классах и их методах. Здесь ключ – это «квалификационное имя», состоящее из полного имени пакета и имени класса, а значение – объект с полями, описывающими типичный Java-класс. У этого объекта присутствуют следующие поля: пакет, модификаторы доступа, аннотации, свойства, методы и т.д.

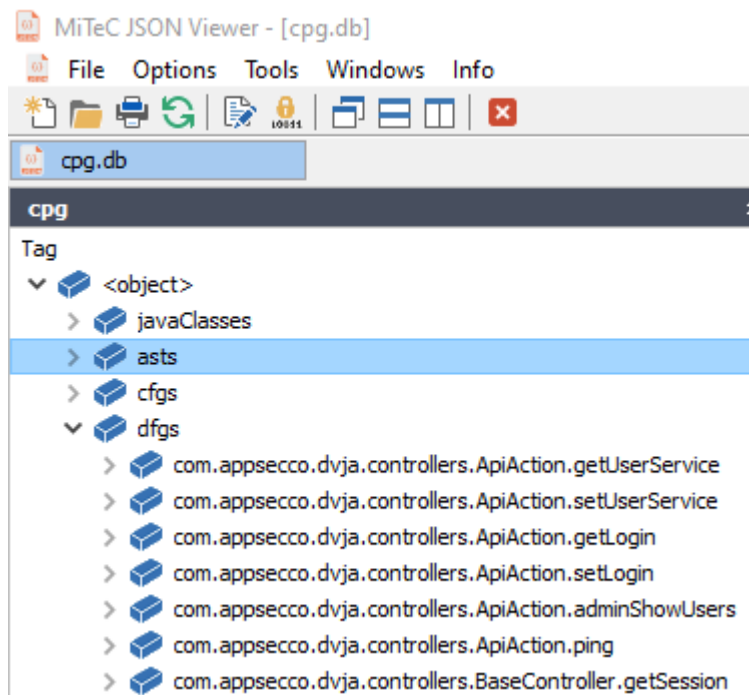


Рисунок 5.5 – База данных для хранения графов

Так как данные хранятся в БД в JSON-формате, то необходимо заняться вопросом сериализации/десериализации объектов, которые представляют графы, ребра и узлы. Для решения этого вопроса был использован Python-модуль `marshmallow`.

Затем с помощью библиотеки `graphviz` происходит экспорт в файл формата `SVG`, который можно открыть в веб-браузере и увидеть графическое представление либо абстрактного синтаксического дерева, либо графа потока управления, либо графа потока данных.

### 5.3 Генерация абстрактного синтаксического дерева

В первую очередь реализуем обход дерева разбора для генерации абстрактного синтаксического дерева. Будем его создавать для файла в целом, то есть корнем AST будет узел с типом `ROOT` и свойством «код» со значением имени файла. Для того чтобы реализовать иерархичность дерева нам понадобится вспомогательная структура данных – «стек родителей» (`parentStack`). В самом начале перед обходом положим на стек корневой узел. Начинается обход и остальные узлы создаются по следующему принципу.

Во-первых, создается узел с соответствующим типом в создающееся AST. Этому узлу назначается свойство «`line`» (от англ. – строка) со значением номера строки, где располагается этот синтаксический элемент. Узлу назначается общий идентификатор «`sharedId`» для того, чтобы можно было связать в самом конце узлы AST, CFG и DFG. Для некоторых узлов кроме типа им назначается свойство «`code`» (код). Значение у этого свойства может быть разным. Если это арифметический или логический оператор, то значением будет знак этого оператора. Например, для сложения – символ «`+`». Если это числовой литерал или идентификатор, то значением будет число или имя соответственно.

Во-вторых, этот узел добавляется в AST.

В-третьих, добавляется ребро, которому нужно указать начальный узел, метку и конечный узел. Начальный узел обычно берется со стека узлов-родителей, метка – «AST», а конечный узел – это созданный в самом начале узел.

В-четвертых, если предполагается, что текущий узел не будет являться листом и у него есть потомки, которые представляют вложенные синтаксические конструкции, то необходимо вызвать обход и для них. Например, если это узел, который соответствует условному IF-оператору. Для этого вначале на стек родителей кладется текущий узел. Вызывается в общем случае метод `visit`, которому передается контекст потомков, которые необходимо обойти и добавить их в AST. После того, как этот вызов произошел, со стека родителей снимается созданный в самом начале узел.

После того, как Посетитель обошел дерево разбора и AST был создан, мы сохраняем AST в базу данных и при необходимости может экспортировать его в DOT-

формат, который затем переведем в формат векторной графики SVG и выведем на экран. Пример графического представления абстрактного синтаксического дерева показан на рисунке 5.6.

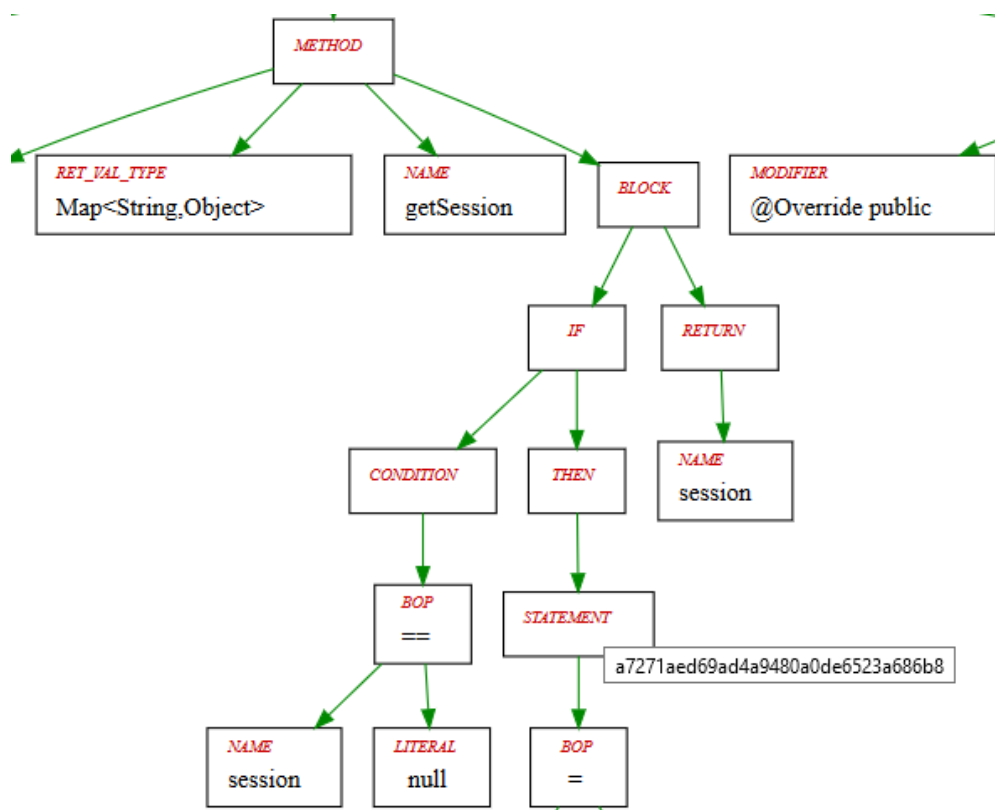


Рисунок 5.6 – Визуализация абстрактного синтаксического дерева

## 5.4 Генерация графа потока управления

Построение графа потока управления будет реализовано не из созданного абстрактного синтаксического дерева, а также с помощью Посетителя, сгенерированного с помощью ANTLR. Это связано с тем, что в Посетителе уже описаны все методы, которые нам нужно просто переопределить и не надо создавать своего посетителя для обхода абстрактного синтаксического дерева. Хотя на его вход будет подаваться весь файл, граф потока управления будет создаваться отдельно для каждого Java-метода. Это достигается тем, что создание CFG будет происходить либо в методе visitMethodDeclaration (объявлении метода), либо visitConstructorDeclaration (объявлении конструктора). В этих методах и будет происходить сброс текущего графа потока управления, чтобы другие вложенные в методы синтаксические конструкции при их посещении не были добавлены не в тот граф.

Процесс добавления узла в граф немного отличается от того, когда была работа с AST. В общем, он имеет следующий вид, но для некоторых конструкций он может немного отличаться:

- 1) создание узла с определенным типом (ему присваиваются такие свойства как номер строки и код),
- 2) добавление узла в граф,
- 3) создание ребра, где начальный узел – это узел из стека предыдущих узлов, метка берется из стека с ребрами, а конечный узел – текущий,
- 4) добавление в стек ребра с меткой (обычно это «EPS», но в случае с предикатами из операторов IF, While или Do-While может быть и «True» или «False»),
- 5) добавление текущего узла в стек предыдущих узлов.

Также как с абстрактным синтаксическим деревом происходит сохранение созданных графов потока управления в базе данных и их экспорт для дальнейшей визуализации (рисунок 5.7).

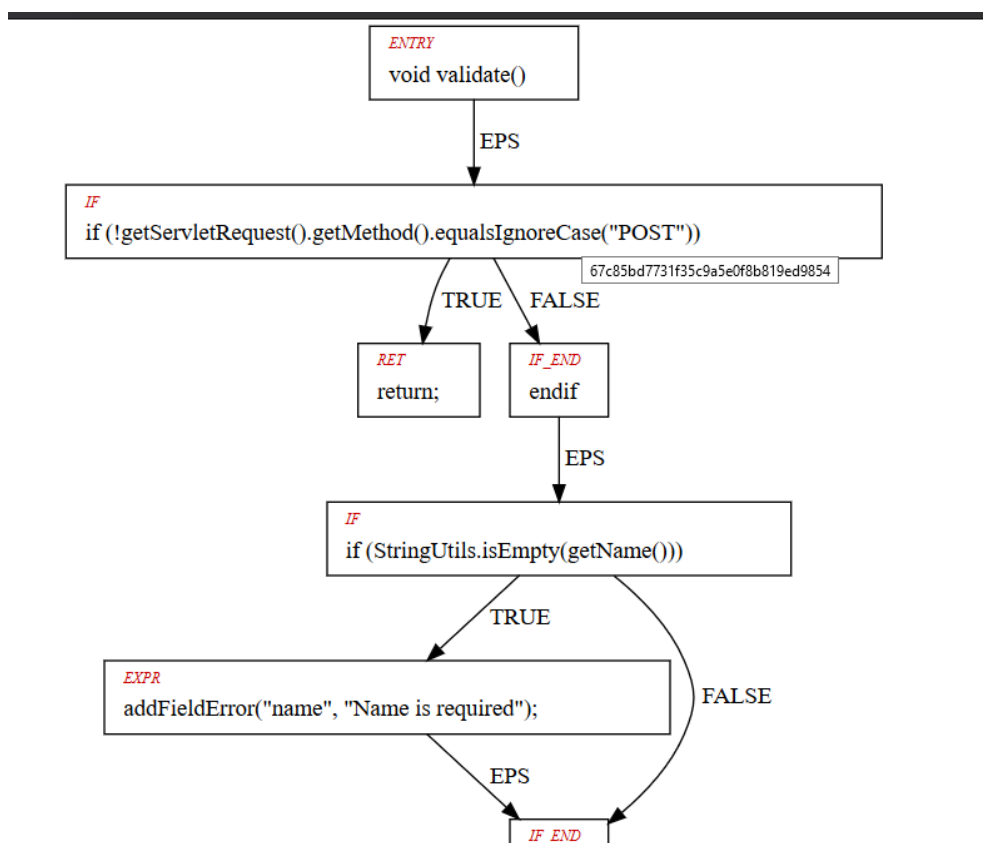


Рисунок 5.7 – Визуализация графа потока управления

## 5.5 Извлечение информации о Java-классах

Прежде чем приступить к генерированию графа потока данных необходимо собрать информацию о методах и, соответственно, об их классах. Это связано с тем, что для межпроцедурного анализа потоков данных необходима информация о параметрах вызываемого метода. Для решения этой задачи был реализован Посетитель, который и собирает всю необходимую информацию. Среди этой информации не только типы и имена параметров, но и модификаторы доступа, возвращаемый тип и др., которые могут понадобиться в дальнейшем. Собранная информация о Java-классах сохраняется в базе данных в коллекции JavaClasses (см. рисунок 5.8). Данные о Java-методах не хранятся обособленно, а организованы внутри классов на тот случай, если имена методов разных классов могут совпадать.

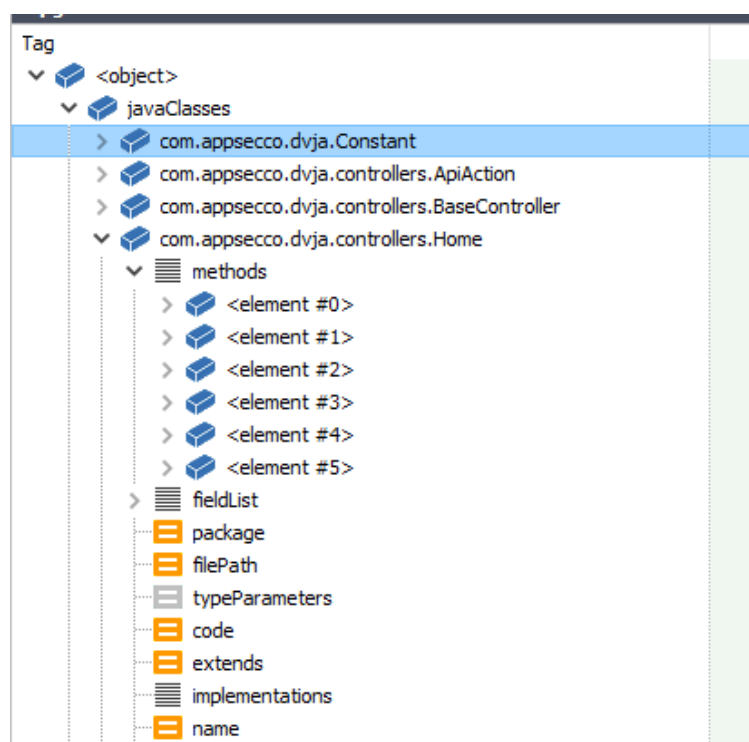


Рисунок 5.8 – Представление Java-класса в БД

```

▼ methods [6]
  ▼ 0 {8}
    isAbstract : ☐ false
    isStatic : ☐ false
    ▼ args [0]
      (empty array)
    retType : ProductService
    modifier : public
    line : 13
    name : getProductService
    sharedId : 498a649380070d322314374c77a5cc5a

```

Рисунок 5.9 – Представление Java-метода в БД

На рисунке 5.9 можно видеть, что у метода есть поле `sharedId` – это общий идентификатор, который соответствует узлу с объявлением метода. В абстрактном синтаксическом дереве это узел с типом `METHOD`, а в графе потока управления – узел с типом `ENTRY`.

## 5.6 Построение графа потока данных

Как уже было сказано, граф потока данных базируется на графе потока управления. В данной реализации потоки данных – это ребра DEF-USE (от англ. «definition-use» – определение-использование). У этих ребер начальный узел – узел, в котором происходит определение переменной, конечный узел – узел, в котором она используется, а метка ребра – имя переменной.

Построение графа потока данных происходит в два этапа. На первом происходит итеративное добавление узлов с присвоением каждому узлу свойств DEF и USE. И если DEF – это множество переменных, которые определяются в данном узле, то USE – переменные, которые используются. Основной функционал по наполнению этих двух множества сосредоточен в функции `analyzeDefUse`. На вход она принимает два аргумента: первый – узел, у которого в случае чего будут обновлены DEF- и USE-множество, второй – выражение в узле, которое нужно дополнительно проанализировать на предмет наличия использований переменных.

Построение графа потока данных происходит в два этапа. Первый этап действует по следующему принципу:

1) обход дерева разбора путем вызова visit-методов: в этом visit-методе в зависимости от синтаксической конструкции происходит наполнение списков определений (defList) и использований (useList),

2) вызов функции analyseDefUse для наполнения свойств узлов DEF и USE.

На втором этапе создаются ребра – потоки данных – путем обхода графа соответствующего метода путем анализа свойств DEF и USE.

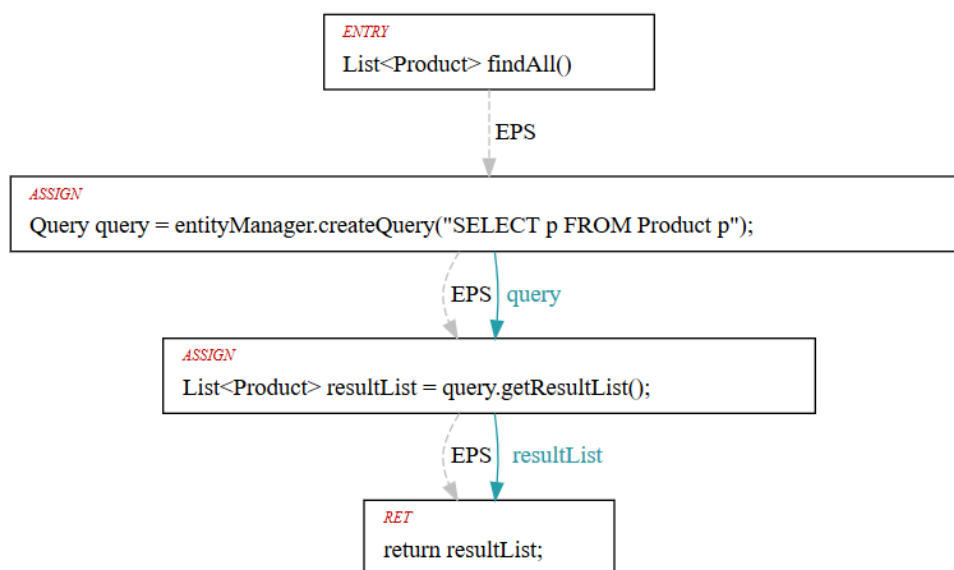


Рисунок 5.10 – Визуализация графа потоков данных

Для межпроцедурного анализа добавляется несколько дополнительных шагов. Этот анализ начинается в visit-методе, который соответствует вызову метода (visitMethodCall). Сначала из БД извлекается информация о вызываемом методе. Затем идет обработка аргументов вызова. Если на предыдущем шаге метод был найден, то DFG-узлу, который соответствует вызову метода, добавляется свойство IP\_DEFs (англ. interprocedural definitions – межпроцедурные определения). У этого свойства значением является словарь с полем общего идентификатора, который соответствует первому (entry) узлу в графе потока управления вызываемой функции. Межпроцедурные потоки данных добавляются в самом конце, когда для каждого метода в анализируемом проекте были созданы свои графы потоки данных. Эти потоки данных представляют собой ребро со свойством «kind» (тип) и значением «INTER», в то время как остальные ребра имеют тип «INTRA» (внутренний).

## 5.7 Использование графовой базы данных OrientDB

Несмотря на то, что были созданы все три представления кода, которые вместе можно объединить в общий граф свойств кода, способ их хранения еще не позволяет применять различные инструменты обхода графов. Для того чтобы двигаться дальше необходимо уже готовые графы, сериализованные с помощью Python-модуля `marshmallow`, экспортировать в систему управления графовыми базами данных OrientDB. Для каждого анализируемого веб-приложения пользователю необходимо вручную создать новую графовую БД. После этого всю остальную работу по взаимодействию с OrientDB сделает уже реализованный класс `OrientDB` из файла `OrientDBDriver.py`. Стоит упомянуть, что все узлы и ребра кладутся в одну базу данных, но обеспечивается различие между узлами AST, CFG и DFG через классы: `ASTNode`, `CFGNode`, `DFGNode`, `ASTEdge`, `CFGEdge` и `DFGEdge`.

Далее необходимо настроить сервер Apache Tinkerpop, который позволяет производить обход графов, используя язык Gremlin. Для этого нужно совершить следующие три операции.

В каталоге `orientdb-tp3\config` создать файл со свойствами графа. Его можно создать на основе файла `demodb.properties`. Например, он может называться `dvja.properties`.

```
gremlin.graph=org.apache.tinkerpop.gremlin.orientdb.OrientEmbeddedFactory
orient-db-name=dvja-analysis
orient-user=admin
orient-pass=admin
```

Рисунок 5.11 – Пример конфигурационного файла со свойствами для OrientDB

В каталоге `orientdb-tp3\config` изменить файл-скрипт `cpg.groovy`, который будет создавать объект обхода (traversal object) для БД из OrientDB.

```
// define the default TraversalSource to bind
globals << [
  cpg : cpgGraph.traversal(),
  dvja : dvjaGraph.traversal()
]
```

Рисунок 5.12 – Фрагмент файла `dvja.groovy`

Изменить файла `orientdb-tp3\config\gremlin-server.yaml`, добавив в него имена traversal-объекта и файла свойств из пунктов 1 и 2.



```

graphs: {
  cpgGraph: ../config/cpg.properties,
  dvjaGraph: ../config/dvja.properties
}
scriptEngines: {
  gremlin-groovy: {
    plugins: { org.apache.tinkerpop.gremlin.server.jsr223.GremlinServerGremlinPlugin: {},
               org.apache.tinkerpop.gremlin.orientdb.jsr223.OrientDBGremlinPlugin: {},
               org.apache.tinkerpop.gremlin.jsr223.ImportGremlinPlugin: {classImports:
               [java.lang.Math], methodImports: [java.lang.Math#*]},
               org.apache.tinkerpop.gremlin.jsr223.ScriptFileGremlinPlugin: {files: [../
               config/cpg.groovy]}}}
  }
}
serializers:
- { className: org.apache.tinkerpop.gremlin.driver.ser.GryoMessageSerializerV3d0,

```

Рисунок 5.13 – Фрагмент конфигурационного файла gremlin-server.yaml

## 5.8 Taint-анализ

Taint-анализ (от англ. taint – оскверненный) – это общий и популярный подход для проверки нарушений целостности и конфиденциальности. Для проверки дефектов целостности taint-анализ отслеживает потоки данных из недоверенного источника ввода source (англ. - источник) к чувствительным инструкциям, называемых sink (англ. – приемник). Например, фрагмент Java-кода, представленный ниже, принимает на вход пользовательский ввод и выполняет запрос, который состоит из уже заданного программистом кода и пользовательского ввода (переменная input):

```

public void getInformation(){
    String input = getUserInput();
    String q = 'SELECT * FROM ' + input;
    int len = 20;
    if (q.length() > len){
        String ans = query(q);
    }
}

```

Source в этом случае – это вызов функции getUserInput, а sink – функция query. Злоумышленник может использовать потенциальный поток данных из getUserInput() в query() для запуска произвольных запросов. Надлежащим способом защиты от атак будет поиск участков кода в программе, в которых пользовательский ввод влияет на чувствительные инструкции, и принятие мер по его фильтрации.

В реализуемом проекте поиск source и sink осуществляется по заранее реализованным Gremlin-запросам. Эти паттерны source и sink можно найти в файлах sourcePatterns.py и sinkPatterns.py соответственно. Например, это может быть поиск по

имени метода `createQuery`, который используется для выполнения SQL-запросов. Поиск source реализован немного сложнее. Так как веб-приложения на языке Java могут быть написаны с использованием различных веб-фреймворков, то и способ ввода данных в веб-приложения может быть различным. В реализуемом инструменте пока существует поддержка двух распространенных веб-фреймворков на языке программирования Java – Spring Web MVC и Struts2. Обычно поиск с использованием Gremlin-запросов осуществляется по абстрактным синтаксическим деревьям и может в дальнейшем расширен для нахождения новых sources и sinks.

После того как были найдены sources и sinks происходит перекрестная проверка на достижимость, т.е. существует ли прямой или косвенный (составной) поток данных из source в sink. Если существует, то найденный taint-поток записывается в базу данных в коллекцию `taintFlows`.

## 5.9 Веб-интерфейс для навигации по графическим представлениям AST, CFG и DFG

Веб-интерфейс представляет собой веб-приложение, написанное на языке Python с использованием веб-фреймворка Flask. Для отображения графических представлений AST, CFG и DFG используются полученные на предыдущих шагах графические файлы формата SVG, а также информация о Java-классах из общей базы данных. На рисунке 5.14 можно наблюдать главную страницу веб-интерфейса.




Главная. Список Java-классов			
Полное имя класса	 Абстрактное синтаксическое дерево	 Методы	 Исходный код
<code>com.appsecco.dvja.Constant</code>	<a href="#">Открыть</a>	<a href="#">Открыть</a>	<a href="#">Открыть</a>
<code>com.appsecco.dvja.controllers.ApiAction</code>	<a href="#">Открыть</a>	<a href="#">Открыть</a>	<a href="#">Открыть</a>
<code>com.appsecco.dvja.controllers.BaseController</code>	<a href="#">Открыть</a>	<a href="#">Открыть</a>	<a href="#">Открыть</a>
<code>com.appsecco.dvja.controllers.Home</code>	<a href="#">Открыть</a>	<a href="#">Открыть</a>	<a href="#">Открыть</a>
<code>com.appsecco.dvja.controllers.Login</code>	<a href="#">Открыть</a>	<a href="#">Открыть</a>	<a href="#">Открыть</a>

Рисунок 5.14 – Главная страница веб-интерфейса

На главной странице представлен список всех Java-классов анализируемого веб-приложения. На ней можно открыть графическое представление абстрактного

синтаксического дерева (см. рисунок 5.15), список методов и страницу с исходным кодом (см. рисунок 5.16).

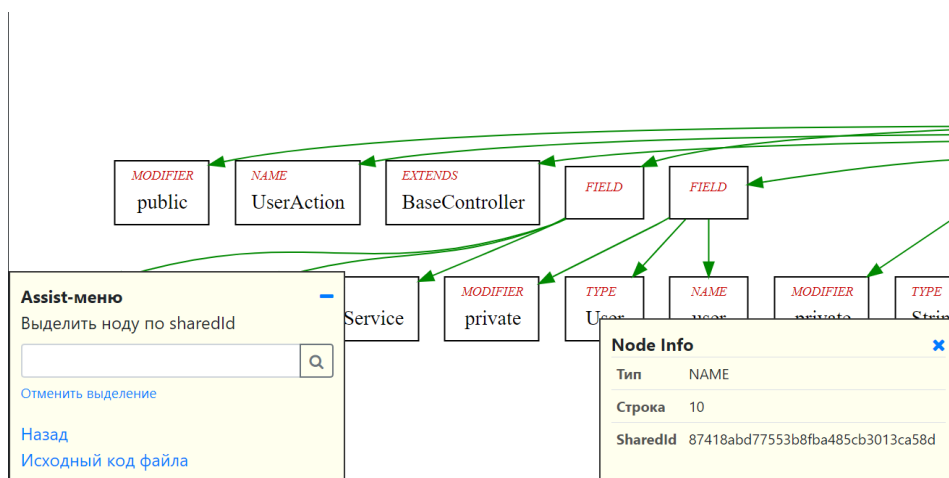


Рисунок 5.15 – Страница с отображением абстрактного синтаксического дерева

На этой странице можно не только изучать сгенерированное абстрактное синтаксическое дерево, но также найти определенный узел по общему идентификатору `sharedId` с помощью окошка Assist-меню. Кроме того, можно получить краткую информацию об узле в окошке справа внизу, если по узлу нажать два раза левой кнопкой мыши. В случае если Assist-меню мешает отображению AST, то его можно скрыть – на месте окошка появится маленькая кнопка, нажав на которую Assist-меню снова восстановится.

Страница с исходным кодом, которую можно открыть как с главной страницы, так и со страницы с AST, представлена на рисунке 5.16. Она имеет функцию подсветки синтаксиса и показа номеров строк.

Path: F:\tmp\dvja\src\main\java\com\appsecco\dvja\controllers\UserAction.java

```
1 package com.appsecco.dvja.controllers;
2
3 import com.appsecco.dvja.models.User;
4 import com.appsecco.dvja.services.UserService;
5 import org.apache.commons.lang.StringUtils;
6
7 public class UserAction extends BaseController {
8
9     private UserService userService;
10    private User user;
11    private String login;
12    private String password;
13    private String passwordConfirmation;
14    private String email;
15    private int userId;
```

Рисунок 5.16 – Страница с исходным кодом

С главной страницы также можно перейти на страницу со списком методов, на которой можно открыть соответствующий граф потока управления или граф потока данных.

## Список методов

### Класс `com.appsecco.dvja.controllers.UserAction`

[← К списку классов](#)



Название метода	 CFG	 DFG
<code>getUserService</code>	<a href="#">Открыть</a>	<a href="#">Открыть</a>
<code>setUserService</code>	<a href="#">Открыть</a>	<a href="#">Открыть</a>
<code>getUser</code>	<a href="#">Открыть</a>	<a href="#">Открыть</a>
<code>setUser</code>	<a href="#">Открыть</a>	<a href="#">Открыть</a>

Рисунок 5.17 – Страница со списком методов Java-класса

Страницы с отображением CFG и DFG аналогичны странице с отображением AST.

## 6 Разработка модулей динамического анализа

Динамический анализатор представляет собой библиотеку на языке Python. Исследователь разрабатывает на ее основе модуль, который и будет тестировать анализируемое веб-приложение. Несмотря на то, что от того, что исследователю необходимо писать модуль, основной функционал уже заключен в библиотеке, ему остается реализовать только некоторые аспекты, которые индивидуальны для каждого веб-приложения.

### 6.1 Извлечение точек входа веб-приложения

Для того чтобы начать тестирование веб-приложения, необходимо определиться с точками входа или то, куда отправлять потенциально опасные входные данные. Веб-приложения на языке программирования Java пишутся с использованием различных веб-фреймворков, у которых структура организации компонентов различается. Разрабатываемый инструмент тестирования защищенности пока реализует анализ структуры только двух распространенных фреймворков Struts 2 и Spring Web MVC, которые позволяют разрабатывать веб-приложения с использованием архитектурного паттерна MVC.

В веб-фреймворке Struts2 информация о точках входа в основном содержится в файле маршрутизации struts.xml и заключена в XML-тегах action.

```
<action name="userSearch" class="com.appsecco.dvja.controllers.UserAction" method="search">
  <interceptor-ref name="authenticationStack"/>
  <result name="input">/WEB-INF/dvja/UserSearch.jsp</result>
  <result>/WEB-INF/dvja/UserSearch.jsp</result>
</action>
```

Рисунок 6.1 – Фрагмент файла маршрутизации struts.xml

Здесь важными являются следующие атрибуты:

- name – имя действия, которое встречается в файлах представлений (HTML- и JSP-файлах) в тегах HTML-форм,
- class – полное имя класса, который реализует контроллер (контроллеры в фреймворке Struts2 наследуются от базового класса ActionSupport),

– method – имя метода-обработчика из вышеупомянутого класса. Это необязательный атрибут, и если он отсутствует, то обработка запроса передается методу с именем execute.

Однако в файле маршрутизации отсутствует информация о передаваемых параметрах. Чтобы ее получить, был реализован парсинг файлов представлений, содержащих HTML-формы с соответствующими тегами.

```
<s:form action="userSearch" method="post" theme="bootstrap">
  <s:textfield label="Login" name="login" placeholder="Enter login to search"/>
  <s:submit cssClass="btn btn-primary"/>
</s:form>
```

Рисунок 6.2 – Фрагмент файла представлений (JSP-файла)

На рисунке 6.2 можно видеть, что действию userSearch, которому есть соответствие в файле struts.xml, передается параметр с именем login через тег s:textfield. При разработке с использованием веб-фреймворка Struts2 в файлах представлений может использоваться специальный синтаксис для определения форм для большей интеграции с другими файлами веб-приложения, в частности с теми, которые реализуют логику. Так s:textfield транслируется в HTML-тег input.

Результатом парсинга веб-приложения, написанного с использованием веб-фреймворка Struts2, является информация, представленная на рисунках 6.3 и 6.4.

```
"userSearch": {
  "class": "com.appsecco.dvja.controllers.UserAction",
  "method": "search"
},
"editUser": {
  "class": "com.appsecco.dvja.controllers.UserAction",
  "method": "edit"
},
```

Рисунок 6.3 – Фрагмент информации о точках входа, собранной из файла маршрутизации struts.xml

```

    "dvja\\UserSearch.jsp": {
        "action": "userSearch",
        "names": [
            "login"
        ]
    },
    "dvja\\a10_redirect\\description.jsp": {},
    "dvja\\a10_redirect\\reference.jsp": {},

```

Рисунок 6.4 – Фрагмент информации о точках входа, собранной из JSP-файлов

В веб-фреймворке Spring Web MVC определение точек входа реализовано другим способом. Здесь файлы представлений играют второстепенную роль. Гораздо важнее Java-файлы классов-контроллеров. Класс делает контроллером специальная аннотация – «@Controller» или «@RestController».

```

@RestController
@AssignmentHints(value = {"SqlStringInjectionHint5b1", "SqlStri
public class SqlInjectionLesson5b extends AssignmentEndpoint {

    private final DataSource dataSource;

    public SqlInjectionLesson5b(DataSource dataSource) { this.d

    @PostMapping("/SqlInjection/assignment5b")
    @ResponseBody
    public AttackResult completed(@RequestParam String userid,
        return injectableQuery(login_count, userid);
    }

```

Рисунок 6.5 – Фрагмент Java-файла с классом-контроллером

Метод-обработчик назначается через аннотацию метода - @PostMapping, @GetMapping, @RequestMapping и т.п. Эта аннотация принимает аргумент – адрес точки входа. Параметры точки входа определяются также через аннотации – аннотации аргументов метода-обработчика. На рисунке 6.5 имя параметра – userid.

Так как мы уже проанализировали Java-классы и записали их в базу данных, то нет необходимости снова парсить Java-файлы анализируемого веб-приложения. Результат работы этого этапа в случае использования фреймворка Spring Web MVC представлен на рисунке 6.6.

```

{
    "method": "completed",
    "route": "\"/SqlInjection/attack4\"",
    "params": [
        "query"
    ],
    "class": "org.owasp.webgoat.sql_injection.introduction.SqlInjectionLesson4"
},

```

Рисунок 6.6 – Фрагмент информации о точках входа, собранной из классов-контроллеров

В итоге для извлечения точек входа были реализованы два класса `Struts2EndpointExtractor` и `SpringMVCEndpointExtractor` для приложений на базе фреймворков `Struts2` и `SpringMVC` соответственно.

## 6.2. Формирование списка атакуемых точек входа

После того, как был сформирован список всех точек входа анализируемого веб-приложения, необходимо среди них отобрать те, которые подвергнуться тестированию на анализ защищенности. Выполняется это с использованием уже готовых к этому моменту результатов Taint-анализа.

Для формирования списка атакуемых точек входа также были реализованы два класса `Struts2AttackEndpointExtractor` и `SpringMVCAccessEndpointExtractor` из-за того, что результаты предыдущего этапа отличаются в зависимости от используемого веб-фреймворка. Конечным этапом является информация, которая имеет структуру как на рисунке 6.7.

```

{
    "uri": "/userSearch",
    "params": [
        "login"
    ],
    "sourceSharedId": "8a87b000ebffca0be78936292e2fd19b",
    "sinkSharedId": "ca61890b2f90fb82de91ac309c4a44c4"
}

```

Рисунок 6.7 – Фрагмент списка атакуемых точек входа



Здесь `uri` – это адрес, на который будет идти HTTP-запрос в ходе дальнейшей работы, `params` – список имен параметров, `sourceSharedId` и `sinkSharedId` – общие идентификаторы источника и приемника, необходимые в дальнейшем для отображения информации о местоположении в коде в случае обнаружения уязвимости.

### 6.3. Подготовка контрольных точек

Для того чтобы определить фильтруются входные данные или нет, необходимо отслеживать значение, которое передается в чувствительные инструкции. Для этого перед ними можно поставить так называемые контрольные точки. В реализуемом инструменте контрольные точки представляют собой обращение к специально созданному классу `CheckPointStorage` и вызов метода этого класса `setValue()`. Пример контрольной точки представлен на рисунке 6.8.

```
public User findByLoginUnsafe(String login) {
    new CheckPointStorage().setValue(
        checkpointId: "2",
        value: "SELECT u FROM User u WHERE u.login = '" + login + "'"
    );
    Query query = entityManager.createQuery( s: "SELECT u FROM User u WHERE
    List<User> resultList = query.getResultList();
```

Рисунок 6.8 – Фрагмент Java-кода с контрольной точкой

В качестве первого параметра передается идентификатор точки останова – для каждой чувствительной инструкции он свой. Вторым параметром передается потенциально уязвимое значение аргумента чувствительного вызова, который можно наблюдать на следующей строке.

Класс `CheckPointStorage` написан на языке Scala, и он скомпилирован и встроен в JAR-файл. Этот JAR-файл необходимо вручную переместить в папку, которая есть в составе переменной среды операционной системы `CLASS_PATH`. Здесь стоит уточнить, что для веб-приложений, которые запускаются с использованием сервера Apache Tomcat, переменная `CLASS_PATH` отличается от одноименной переменной среды ОС. В таком случае Java-библиотеки, необходимые для корректной работы запускаемого веб-приложения, располагаются в папке `target\<имя проекта>\WEB-INF\lib`. Код класса `CheckPointStorage` приведен на рисунке 6.9.

```

package org.khbr

import com.redis._

class CheckPointStorage {
    val r = new RedisClient( host = "localhost", port = 6379)

    def setValue(checkpointId: String, value: Any): Unit = {
        value match {
            case strings: Array[String] =>
                r.set( key = "ch_" + checkpointId, strings.mkString(" "))
            case _ =>
                r.set( key = "ch_" + checkpointId, value)
        }
    }
}

```

Рисунок 6.9 – Реализация класса CheckPointStorage

Хранилищем значений контрольных точек служит база данных типа «ключ-значение» Redis. Перед запуском динамической стадии анализа необходимо убедиться, что сервер Redis работает.

Функционал по управлению и отслеживанию контрольных точек сосредоточен в классе CheckpointManager. Этот класс имеет два метода prepare и getCheckpointValue. Первый предназначен для инициализации контрольных точек в БД Redis для каждой чувствительной инструкции. Под инициализацией понимается создание записей с ключом вида ch\_<идентификатор> и значением пустой строки. Второй метод предназначен для получения текущего значения контрольной точки и понадобится на следующих стадиях динамического анализа.

#### 6.4. Инструментирование исходного кода

Инструментирование – это процесс модификации исследуемой программы с целью ее дальнейшего анализа [21]. Модификация в рамках работы разрабатываемого инструмента будет заключаться во вставке уже упомянутых в предыдущем разделе контрольных точек. Для того чтобы процесс инструментирования происходил безболезненно для анализируемого веб-приложения, а именно для того чтобы после анализа веб-приложение функционировало в своем штатном режиме, применяется система контроля версий Git. Так как разработка ведется на языке программирования

Python, то для взаимодействия с системой Git используется сторонний модуль GitPython. Весь процесс инструментации реализован в методе `GitManager.prepare()`, и его можно описать следующим образом:

- 1) сохранение имени текущей ветки в отдельном промежуточном файле `saved_branch.txt`,
- 2) создание новой ветки с именем «instrument» и переход на нее,
- 3) индексирование всех текущих изменений и создание коммита с сообщением «init state» (от англ. – начальное состояние),
- 4) вставка контрольных точек в виде создания объекта `CheckPointStorage` и вызова его метода `setValue`. Кроме того, также происходит добавление в начале модифицируемого файла строки с импортом класса `CheckPointStorage`.

В дальнейшем, когда динамический анализ закончится, произойдет откат всех изменений с помощью метода `InstrumentTool.revert()`:

- 1) отмена модификации, связанных с добавлением контрольных точек,
- 2) переход на ранее сохраненную ветку,
- 3) удаление ветки «instrument».

## **6.5. Проведение атаки**

После того, как было проведено инструментирование, модуль динамического анализа делает паузу для того, чтобы пользователь запустил инструментированное веб-приложение. После того, как он его запустит, он нажимает любую клавишу для продолжения.

Запускается тестирование веб-приложения, в ходе которого перебираются атакуемые точки входа. Весь процесс тестирования сосредоточен внутри класса `Attacker` и его метода `attack`. Вначале отсылается HTTP-запрос с безопасным значением параметров, которые не могут привести к непредвиденному поведению приложения. С помощью менеджера контрольных точек достается значение аргумента для чувствительной инструкции и передается в метод `setSafeQuery()` объекта `InjectionChecker`, который является частью модуля динамического анализа и отвечает за проверку на наличие уязвимости в случае, если входные данные недостаточно фильтруются. После отправляется HTTP-запрос уже с потенциально опасным содержимым. Например, если тестируется точка входа на предмет SQL-инъекции, то в параметре может содержаться кавычка. Снова достается значение контрольной точки и передается в метод

InjectionChecker.checkPotentialUnsafeQuery, в котором происходит сравнение значений контрольной точки при безопасном параметре и при параметре, содержащем вектор атаки. В случае SQL-инъекции проверка реализуется с помощью синтаксического анализатора языка запросов SQL. Если входная строка содержала один из служебных символов, то после лексического анализа количество токенов изменится по сравнению с входной строкой при передаче безопасного значения параметра. Если проверка определила возможность инъекции, то пользователю выводится соответствующее сообщение на консоль, в котором также содержится информация о местонахождении чувствительной инструкции, аргумент которой не подвергся должной фильтрации.

## 6.6. Пользовательский модуль динамического анализа

Как было уже сказано, каждое веб-приложение обладает некоторыми индивидуальными аспектами. Это может быть механизм аутентификации, дополнительная логика, которая может быть реализована серией HTTP-запросов и т.д. Поэтому пользователю самостоятельно необходимо реализовать модуль, который будет использовать упомянутую выше библиотеку. Основной код по работе с библиотекой представлен на рисунке 6.10.

```
analyzer = DynamicAnalyzer(projectConfig=projectConfig)
analyzer.setEndpointExtractor(
    Struts2EndpointExtractor(projectConfig=projectConfig)
)
analyzer.setAttackEndpointExtractor(
    Struts2AttackEndpointExtractor(projectConfig=projectConfig)
)
analyzer.setWebDriver(webDriver=DVJADriver())
analyzer.run()
```

Рисунок 6.10 – Интерфейс работы с библиотекой

Вначале создается главный объект динамического анализатора на основе класса DynamicAnalyzer, в конструктор которого передается конфигурационный объект проекта. Затем настраиваются объекты, которые будут извлекать общие и атакуемые точки входа в зависимости от веб-фреймворка. Далее настраивается веб-драйвер, который будет использоваться в классе Attacker, когда будут отсылаться безопасные и опасные HTTP-запросы. И в самом конце методом run запускается сам процесс динамического анализа.

Остановимся подробнее на конфигурационном объекте проекта и веб-драйвере. Первый представляет собой словарь, пример которого можно наблюдать на рисунке 6.11.

```
projectConfig = {  
    "DB": r"F:\tmp\dvja-analysis\dvja-analysis.db",  
    "REPO": r"F:\tmp\dvja\.git",  
    "SAVED_BRANCH_FILE": "saved_branch.txt",  
    "JSPFilesDir": r"F:\tmp\dvja\src\main\webapp\WEB-INF",  
    "STRUTS_XML": r"F:\tmp\dvja\src\main\resources\struts.xml",  
    "VIEWS_DIR": "",  
    "base_url": "http://localhost:8080",  
    "auth_login": "khbr",  
    "auth_password": "passwordis1337"  
}
```

Рисунок 6.11 – Пример конфигурации

Рассмотрим каждое из свойств:

- DB – путь к файлу БД, который был создан на этапе статического анализа,
- REPO – путь к git-директории анализируемого веб-приложения,
- JSPFilesDir – путь к директории, где лежат файлы представлений (обязателен только тогда, когда анализируемое приложение построено на базе веб-фреймворка Struts2),
- STRUTS\_XML – путь к файлу маршрутизации (обязателен только тогда, когда анализируемое приложение построено на базе веб-фреймворка Struts2),
- VIEWS\_DIR – путь к директории, где лежат файлы представлений (обязателен только тогда, когда анализируемое приложение построено на базе веб-фреймворка Spring Web MVC),
- base\_url – базовый адрес, по которому доступно анализируемое веб-приложение,
- auth\_login – имя пользователя для аутентификации в анализируемом веб-приложении,
- auth\_pass – пароль пользователя для аутентификации в анализируемом веб-приложении.

От этого файла конфигурации зависит работа каждого компонента библиотеки динамического анализа.

Веб-драйвер реализуется пользователем и наследуется от базового класса WebDriver, в котором уже реализованы методы для отправки GET- и POST- запросов. Также в нем содержатся пустые методы аутентификации (authenticate) и преднастройки

(prepare), которые пользователь может переопределить. Например, на рисунке 6.12 представлен пример пользовательского веб-драйвера.

```
class DVJADriver(WebDriver):  
    def authenticate(self):  
        payload = {  
            "login": projectConfig["auth_login"],  
            "password": projectConfig["auth_password"]  
        }  
        baseURL = projectConfig["base_url"]  
        self.s.post(f"{baseURL}/login", data=payload)
```

Рисунок 6.12 – Пример пользовательского веб-драйвера

Данный веб-драйвер реализует аутентификацию, переопределяя метод `authenticate` и отправляя POST-запрос с заданными именем пользователя (`login`) и паролем (`password`) по адресу `http://localhost:8080/login`.

## 7 Тестирование работы разработанного инструмента

Главным результатом разработанного средства анализа защищенности веб-приложений является отчет, с указанием уязвимой точки входа, необрабатываемого или недостаточно обрабатываемого параметра пользовательских входных данных и местоположение чувствительной инструкции, которая использует потенциально опасные данные, в коде анализируемого приложения. Отчет представляет собой файл с именем report и в формате JSON, пример которого можно видеть на рисунке 7.1. Этот файл располагается в той же директории, откуда запускается модуль динамического анализа.



```
1  {
2    "XSS": [
3      {
4        "uri": "/addEditProduct",
5        "param": "product.name",
6        "line": 86,
7        "file": "F:\\tmp\\dvja\\src\\main\\java\\com\\appsecco\\dvja\\controllers\\ProductAction.java",
8        "code": "productService.save(product);"
9      },
10     {
11       "uri": "/addEditProduct",
12       "param": "product.code",
13       "line": 86,
14       "file": "F:\\tmp\\dvja\\src\\main\\java\\com\\appsecco\\dvja\\controllers\\ProductAction.java",
15       "code": "productService.save(product);"
16     }
17   ]
18 }
```

Рисунок 7.1 – Файл отчета с примером обнаруженных XSS-уязвимостей

```

''
"SQL": [
  {
    "uri": "/listProduct",
    "param": "searchQuery",
    "line": 48,
    "file": "F:\\tmp\\dvja\\src\\main\\java\\com\\appsecco\\dvja\\services\\ProductService.java",
    "code": "query = entityManager.createQuery(\"SELECT p FROM Product p WHERE p.name LIKE '%" + name + "%'\")"
  },
  {
    "uri": "/userSearch",
    "param": "login",
    "line": 75,
    "file": "F:\\tmp\\dvja\\src\\main\\java\\com\\appsecco\\dvja\\services\\UserService.java",
    "code": "query = entityManager.createQuery(\"SELECT u FROM User u WHERE u.login = '\" + login + \"'\")"
  }
],
"CI": [
  {
    "uri": "/ping",
    "param": "address",
    "line": 46,
    "file": "F:\\tmp\\dvja\\src\\main\\java\\com\\appsecco\\dvja\\controllers\\PingAction.java",
    "code": "process = runtime.exec(command)"
  }
]

```

Рисунок 7.2 – Файл отчета с обнаруженными уязвимостями SQL-инъекций и ввода команд

При анализе на защищенность веб-приложения DVJA удалось обнаружить 7 XSS-уязвимостей, две уязвимости SQL-инъекции и одну уязвимость ввода команд. Среди них не было найдено ложных срабатываний, а также инструмент не пропустил другие возможные уязвимости трех классов, то есть при анализе не было допущено ошибок второго рода.



## **8 Безопасность человеко-машинного взаимодействия**

### **8.1 Особенности функционального назначения объекта**

Разработанный инструмент тестирования защищенности веб-приложений позволяет получать информацию о возможных уязвимостях в коде анализируемых веб-приложений. Также для разработчика системы предусмотрен внутренний веб-интерфейс, благодаря которому можно проводить отладку правильности построения синтаксической и семантических моделей исходного кода.

С точки зрения пользователя, разработанный инструмент – это консольная программа, если не считать веб-интерфейс по навигации по моделям исходного кода, который является больше вспомогательным, чем основным.

### **8.2 Описание процесса эксплуатации объекта**

Процесс работы с разрабатываемым инструментом проходит в три этапа:

- 1) предварительная настройка и запуск необходимых вспомогательных серверов,
- 2) инициализация проекта по анализу веб-приложения,
- 3) запуск статического анализатора,
- 4) реализация и запуск пользовательского модуля динамического анализа.

Рассмотрим каждый из этапов более подробно.

Разрабатываемый инструмент зависит от работы двух сторонних серверов: OrientDB и Redis. Для обеспечения работы второго, его достаточно просто запустить. Однако перед тем как запускать OrientDB, необходимо создать новую графовую БД и провести настройку Apache Tinkerpop, как это было описано в разделе 5.7. Использование графово-ориентированной базы данных OrientDB.

Прежде чем запускать главные модули по анализу веб-приложения, необходимо предварительно организовать место на диске, где будут храниться конфигурация и промежуточные результаты анализа. Процесс запуска инициализации выполняется командой `init`, которая представлена на рисунке 8.1.

```
(venv) F:\tmp>python ..\Workspace\CPG-Creator\interface2.py init dvja-analysis
Creating a project with name 'dvja-analysis'...

(venv) F:\tmp>
```

Рисунок 8.1 – Инициализация проекта

В текущей директории будет создана папка dvja-analysis, в котором пока лежит только один файл конфигурации config.json. Некоторые поля в нем уже заполнены, однако для дальнейшей работы необходимо заполнить и остальные. Пример изначального файла конфигурации представлен на рисунке 8.2.

```
{
  "name": "dvja-analysis",
  "target-dir": "",
  "DB": "dvja-analysis.db",
  "orientdb-name": "dvja-analysis",
  "orientdb-user": "root",
  "orientdb-pass": "toor",
  "web-framework": "",
  "JSPFilesDir": "",
  "STRUTS_XML": "",
  "VIEWS_DIR": ""
}
```

Рисунок 8.2 – Изначальный файл конфигурации проекта

Рассмотрим каждую из настроек более подробно:

- name – общее название проекта по анализу,
- target-dir – путь к директории с анализируемым веб-приложением,
- DB – путь к файлу общей базы данных (по умолчанию, это текущая директория),
- orientdb-name – имя графовой БД,
- orientdb-user – имя пользователя графовой БД,
- orientdb-pass – пароль пользователя графовой БД,
- web-framework – веб-фреймворк анализируемого веб-приложения (допустимые значения – «Struts2» и «SpringMVC»).

После того, как конфигурационный файл будет настроен, можно переходить к запуску модуля статического анализа.

Запуск статического анализатора представлен на рисунке 8.3.

```
(venv) F:\tmp\dvja-analysis-2>python ../../Workspace\CPG-Creator\interface2.py run-static
Obtaining Java classes info...
Handling: F:\tmp\dvja\src\main\java\com\appsecco\dvja\Constant.java
Handling: F:\tmp\dvja\src\main\java\com\appsecco\dvja\controllers\ApiAction.java
Handling: F:\tmp\dvja\src\main\java\com\appsecco\dvja\controllers\BaseController.java
Handling: F:\tmp\dvja\src\main\java\com\appsecco\dvja\controllers\Home.java
Handling: F:\tmp\dvja\src\main\java\com\appsecco\dvja\controllers>Login.java
Handling: F:\tmp\dvja\src\main\java\com\appsecco\dvja\controllers\PingAction.java
Handling: F:\tmp\dvja\src\main\java\com\appsecco\dvja\controllers\ProductAction.java
Handling: F:\tmp\dvja\src\main\java\com\appsecco\dvja\controllers\RedirectAction.java
Handling: F:\tmp\dvja\src\main\java\com\appsecco\dvja\controllers\Register.java
Handling: F:\tmp\dvja\src\main\java\com\appsecco\dvja\controllers\ResetPassword.java
Handling: F:\tmp\dvja\src\main\java\com\appsecco\dvja\controllers\UserAction.java
Handling: F:\tmp\dvja\src\main\java\com\appsecco\dvja\interceptors\AuthenticationInterceptor.java
Handling: F:\tmp\dvja\src\main\java\com\appsecco\dvja\models\Product.java
```

Рисунок 8.3 – Запуск статического анализатора

После того, как предыдущая команда завершит свою работу, в папке анализа появится файл общей базы данных с основными результатами статического анализа, а также папка plots, содержащая графические представления AST, CFG и DFG в формате SVG. Для более удобной навигации по этим графическим представлениям можно воспользоваться веб-интерфейсом, который работает через веб-сервер. Команда запуска веб-сервера представлена на рисунке 8.4.

```
(venv) F:\tmp\dvja-analysis-2>python ../../Workspace\CPG-Creator\interface2.py web
* Serving Flask app "web.app" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Рисунок 8.4 – Запуск веб-сервера для веб-интерфейса

Веб-интерфейс станет доступным по адресу <http://127.0.0.1:5000/>, который можно открыть в веб-браузере. Главная страница веб-интерфейса представлена на рисунке 8.5.

## Главная. Список Java-классов




Полное имя класса	 Абстрактное синтаксическое дерево	 Методы	 Исходный код
<a href="#">com.appsecco.dvja.Constant</a>	<a href="#">Открыть</a>	<a href="#">Открыть</a>	<a href="#">Открыть</a>
<a href="#">com.appsecco.dvja.controllers.ApiAction</a>	<a href="#">Открыть</a>	<a href="#">Открыть</a>	<a href="#">Открыть</a>
<a href="#">com.appsecco.dvja.controllers.BaseController</a>	<a href="#">Открыть</a>	<a href="#">Открыть</a>	<a href="#">Открыть</a>
<a href="#">com.appsecco.dvja.controllers.Home</a>	<a href="#">Открыть</a>	<a href="#">Открыть</a>	<a href="#">Открыть</a>
<a href="#">com.appsecco.dvja.controllers.Login</a>	<a href="#">Открыть</a>	<a href="#">Открыть</a>	<a href="#">Открыть</a>

Рисунок 8.5 – Главная страница веб-интерфейса

Как уже было сказано, пользователь сам реализует модуль динамического анализа с использованием библиотеки. Типичный процесс работы на этом этапе состоит в следующем:

- создание и активация виртуального Python-окружения (рисунок 8.6),

```
F:\tmp\dvja-analysis-2>python -m venv venv  
F:\tmp\dvja-analysis-2>venv\Scripts\activate  
(venv) F:\tmp\dvja-analysis-2>
```

Рисунок 8.6 – Создание и активация виртуального окружения

- установка библиотеки (рисунок 8.7),

```
(venv) F:\tmp\dvja-analysis-2>pip install ../../tmp/test10/dynamic-analyzer/dist/  
dynamic_analyzer-1.0-py3-none-any.whl  
Processing f:\tmp\test10\dynamic-analyzer\dist\dynamic_analyzer-1.0-py3-none-any.  
whl  
Collecting gitdb==4.0.5 (from dynamic-analyzer==1.0)  
  Using cached https://files.pythonhosted.org/packages/48/11/d1800bca0a3bae820b84  
b7d813ad1eff15a48a64caea9c823fc8c1b119e8/gitdb-4.0.5-py3-none-any.whl  
Collecting smmap==3.0.4 (from dynamic-analyzer==1.0)
```

Рисунок 8.7 – Установка библиотеки

- реализация пользовательского модуля,
- запуск пользовательского модуля (рисунок 8.8).

```
(venv) F:\tmp\dvja-analysis-2>python user-dynamic.py  
Action for class com.appsecco.dvja.controllers.ApiAction and method ping in JSP-files not found  
Action for class com.appsecco.dvja.controllers.Login and method execute not found in struts.xml  
Action for class com.appsecco.dvja.controllers.ResetPassword and method execute not found in struts.xml  
Run an application. Enter an any key while it is ready up...  
Testing /userSearch...  
  Testing parameter "login"...  
ATTACK is SUCCESSFUL!!!! URI: /userSearch (ca61890b2f90fb82de91ac309c4a44c4)  
-----  
Stop an application. Enter an any key while it is ready up...  
Clearing...
```

Рисунок 8.8 – Запуск пользовательского модуля

Стоит упомянуть, что в процессе последней операции, пользователю также необходимо вручную запускать и останавливать анализируемое веб-приложение.

### 8.3 Оценка эргономичности пользовательского интерфейса

Разработанный инструмент представляет собой консольное приложение. Взаимодействие пользователя с ней происходит с помощью командой строки или терминала. Ниже приводится анализ эргономичности пользовательского интерфейса программы тестирования защищенности веб-приложений. При этом учитываются все критерии и показатели оценки.

Все оценки пользовательского интерфейса поделены на группы, отвечающие за логичность компоновки элементов, интуитивность и ассоциативность диалогового режима, полноту реализации обратной связи с пользователем и визуальное оформление пользовательского интерфейса. Ниже приведено заполнение таблицы с оценками пользовательского интерфейса разработанной системы и ближайшего аналога.

В качестве аналога для сравнения была выбрана похожий инструмент, а именно «Find Sec Bugs». Этот инструмент поставляется в различных вариациях, но наиболее популярной является вариация в виде плагина для интегрированной среды разработки. Интерфейс плагина представлен на рисунке 8.9.

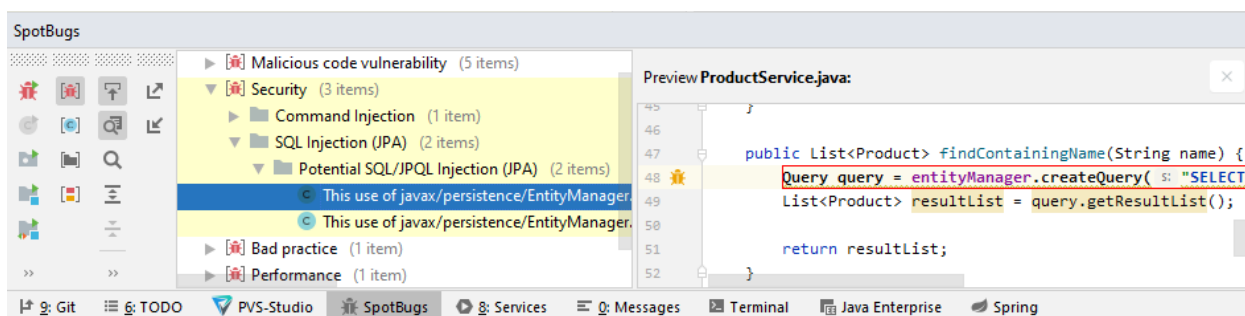


Рисунок 8.9 – Пример работы аналога

У разработанной системы и аналога есть общий функционал, который характеризует главную цель этих систем – вывод информации (краткого отчета) о найденных потенциальных уязвимостях.

Однако у аналога разрабатываемого инструмента отсутствует возможность динамически проверить возможную уязвимость, что обычно приводит к большему числу ложных срабатываний. В свою очередь это влечет за собой дополнительные проверки со стороны сотрудника службы ИБ, и как следствие к повышению напряженности от дальнейшего взаимодействия с анализируемым веб-приложением.

Результаты оценки эргономичности интерфейса приведены в таблице 8.1. Однако несколько показателей в таблице, например, «Отсутствие перекрывающихся окон на

экране», имеют шкалу оценивания 0 (не выполняется) и 1 (выполняется). В ячейках «Сумма баллов» указана сумма, накопленная оценками показателей каждой группы [22].

Таблица 8.1 — Оценка эргономичности пользовательского интерфейса

Критерии и показатели оценки	Аналог «FindSecBugs»	Разработанный инструмент
1. Логичность компоновки элементов	30	22
1.1. Наличие и сохранение во всей программе единой системы группировки полей	3	0
1.2. Порядок заполнения полей	0	0
1.3. Обоснованный порядок размещения пунктов списков	4	0
1.4. Обоснованное соотношение между «детальностью» и «обобщенностью» выводимой на экран информации	4	3
1.5. Единство в выборе способа работы с однотипными данными	4	4
1.6. Видимое разделение основных и вспомогательных блоков информации	4	4
1.7. Видимое разделение редактируемых обязательных и необязательных, а также не редактируемых полей	0	0
1.8. Разделение задач: для каждой задачи открывается свое окно, одно окно предназначено для выполнения только одной задачи	0	0
1.9. Возможность совершать несколько различных действий (решать несколько задач) одновременно	1	1

Продолжение таблицы 8.1

Критерии и показатели оценки	Аналог «FindSecBugs»	Разработанный инструмент
1.10. Отсутствие перекрывающихся окон на экране	5	5
1.11. Отсутствие рядом расположенных кнопок с противоположным действием	0	0
1.12. Отсутствие дублирующих полей ввода	5	5
2. Интуитивность и ассоциативность диалогового окна	28	16
2.1. Продуманная навигация и целевая ориентация в программе: что надо сделать в следующий момент, очевидность каждого следующего шага действий	4	2
2.2. Наличие контекстных подсказок, меню дальнейших событий или объектов, запоминание типичных путей диалога	4	1
2.3. Наличие средств, позволяющих пользователям восстановить данные после ошибочных действий	0	0
2.4. Учет предметной области и профессиональных знаний пользователя	4	5
2.5. Возможность настройки интерфейса для пользователей с разным опытом работы с компьютером	1	1
2.6. Типичность интерфейса: использование стандартных элементов взаимодействия, их традиционное или общепринятое расположение	4	2
2.7. Постоянная возможность вызова главного меню	5	1

Продолжение таблицы 8.1

Критерии и показатели оценки	Аналог «FindSecBugs»	Разработанный инструмент
2.8. Наличие механизмов поиска, средств листания и прокрутки при работе с большими фрагментами информации	2	2
2.9. Легкость и скорость обучения пользования программой, отсутствие необходимости специального обучения	4	2
3. Полнота реализации обратной связи с пользователем	9	9
3.1. Наличие сообщений о состоянии системы	4	4
3.2. Отображение режима работы системы	0	0
3.3. Настраиваемое отображение значений важных для текущей задачи показателей	0	0
3.4. Отображение действий пользователя	0	0
3.5. Ясность и информативность сообщений системы	5	5
4. Визуальное оформление пользовательского интерфейса	20	18
4.1. Ограниченное использование цвета в оформлении элементов интерфейса соответствует целевому назначению программного продукта и учитывает продолжительность работы с ним пользователя	5	4
4.2. Используемые сочетания оттенков цвета совместимы	4	3
4.3. Контрастность объектов различения с фоном комфортная и не требует перенастройки дисплея	4	3



Продолжение таблицы 8.1

Критерии и показатели оценки	Аналог «FindSecBugs»	Разработанный инструмент
4.4. Шрифт основного текста и заголовков легко читаем или может быть изменен	2	2
4.5. Размер шрифта основного текста, подписей элементов интерфейса может быть увеличен или уменьшен пользователем	0	1
4.6. Единство стиля оформления	5	5
Итоговая сумма баллов	87	65

Из таблицы видно, что аналог выигрывает в плане удобства по сравнению с разрабатываемым инструментом, так как консольный интерфейс заставляет пользователя совершать больше действий. Однако аналог обладает меньшей точностью результатов, что в конечном итоге заставляет пользователя проводить дополнительный аудит потенциально безопасных участков исходного кода анализируемого веб-приложения.

В дальнейшем можно улучшить взаимодействие с пользователем, разработав графический интерфейс, который будет предоставлять более обширные возможности по управлению анализом веб-приложения.

#### 8.4 Оценка напряженности процесса эксплуатации объекта

Показатели напряженности трудового процесса у пользователей, работающих со средствами анализа защищенности приложений до и после внедрения разработанной системы приведены в таблице 8.2. При этом учитывался весь комплекс производственных факторов (стимулов, раздражителей), создающих предпосылки для возникновения неблагоприятных нервно-эмоциональных состояний (перенапряжения). Все факторы (показатели) трудового процесса имеют качественную или количественную выраженность и в соответствии с гигиеническим руководством Р 2.2.2006-05 [23] сгруппированы по видам нагрузок: интеллектуальные, сенсорные, эмоциональные, монотонные, режимные нагрузки. В зависимости от уровня либо величины анализируемого фактора, всего используются семь оценок (классов): 1.0 – значение фактора оптимально, 2.0 – допустимо,

3.1, 3.2, 3.3, 3.4 – вредно в той или иной степени, 4.0 – опасно. Напряженность трудового процесса оценивается первыми четырьмя классами (1.0 – 3.2).

Таблица 8.2 – Показатели напряженности трудового процесса до и после внедрения разработки.

Показатели напряженности трудового процесса	Класс условий труда	
	Сотрудник службы ИБ	
	до внедрения	после внедрения
1 Интеллектуальные нагрузки		
1.1 Содержание работы	2	2
1.2 Восприятие сигналов (информации) и их оценка	2	2
1.3 Распределение функций по степени сложности задания	2	2
1.4 Характер выполняемой работы	2	2
2 Сенсорные нагрузки		
2.1 Длительность сосредоточенного наблюдения (% времени смены)	2	2
2.2 Плотность сигналов (световых, звуковых) и сообщений в среднем за 1 час работы	3.1	2
2.3 Число производственных объектов одновременного наблюдения	2	1
2.4 Размер объекта различения (при расстоянии от глаз работающего до объекта различения не более 0,5 м) в мм при длительности сосредоточенного наблюдения (% времени смены)	2	2
2.5 Работа с оптическими приборами (микроскопы, лупы и т.п.) при длительности сосредоточенного наблюдения (% времени смены)	1	1
2.6 Наблюдение за экранами видеотерминалов (часов в смену): при буквенно-цифровом типе отображения информации	3.1	2

Продолжение таблицы 8.2

Показатели напряженности трудового процесса	Класс условий труда	
	Сотрудник службы ИБ	
	до внедрения	после внедрения
2.7 Нагрузка на слуховой анализатор (при производственной необходимости восприятия речи или дифференцированных сигналов)	1	1
2.8 Нагрузка на голосовой аппарат (суммарное количество часов, наговариваемое в неделю)	1	1
3 Эмоциональные нагрузки		
3.1 Степень ответственности за результат собственной деятельности. Значимость ошибки	3.1	2
3.2 Степень риска для собственной жизни	1	1
3.3 Степень ответственности за безопасность других лиц	1	1
3.4 Количество конфликтных ситуаций, обусловленных профессиональной деятельностью, за смену	1	1
4 Монотонность нагрузки		
4.1 Число элементов (приемов), необходимых для реализации простого задания или в многократно повторяющихся операциях	3.1	2
4.2 Продолжительность (в сек) выполнения простых заданий или повторяющихся операций	2	2
4.3 Время активных действий (в % к продолжительности смены). В остальное время – наблюдение за ходом производственного процесса	2	2
4.4 Монотонность производственной обстановки (время пассивного наблюдения за ходом техпроцесса в % от времени смены)	2	2
5 Режим труда и отдыха		
5.1 Фактическая продолжительность рабочего дня	1	1
5.2 Сменность работы	1	1
5.3 Наличие регламентированных перерывов и их продолжительность	1	1
Общая оценка напряженности трудового процесса	2	2

На основании анализа факторов напряженности трудового процесса в соответствии с таблицей 8.2 имеем следующие показатели.

До внедрения: 9 факторов 1 класса, 10 факторов 2 класса, 4 фактора 3.1 класса.

После внедрения: 10 факторов 1 класса, 13 факторов 2 класса.

На основе подсчета количества данных показателей можно сказать следующее.

До внедрения программного продукта условия труда являются «Допустимыми», так как выполняется следующее условие: когда от 1 до 5 показателей отнесены к 3.1 и/или 3.2 степеням вредности, а остальные показатели имеют оценку 1-го и/или 2-го классов.

После внедрения программного продукта условия труда также являются «Допустимыми», так как выполняется то же самое условие.

Можно сделать вывод, что после внедрения разработанной системы показатели напряженности трудового процесса у пользователей не изменились.

#### **8.5 Разработка мер профилактики и повышения безопасности человеко-машинного взаимодействия**

Допустимые условия труда, которые удалось сохранить после внедрения разработанного продукта, условно относят к безопасным. Это значит, что обязательным остается соблюдение пользователем профилактических защитных мероприятий, регламентированных требованиями СанПиН 2.2.2/2.4.1340-03 [24].

Так как характер работы требует постоянного взаимодействия с ЭВМ, то рекомендуется проводить комплексы упражнений для глаз и физкультурные паузы. Вредное продолжительное наблюдение за экранами видеотерминалов может приводить к переутомлению, и должно обязательно прерываться на 5-15 минут через каждые 45 минут другими видами деятельности без использования компьютера.

Кроме того, от рабочего места требуется соблюдение санитарно-гигиенических требований к помещению, освещению, микроклимату, шуму и вибрации.

## **9 Технико-экономическое обоснование**

### **9.1 Обоснование необходимости и актуальности разработки**

Средство анализа защищенности приложений – это инструмент, предназначенный для проверки свойств безопасности разрабатываемого программного продукта. Такие средства могут сканировать код, анализируя приложение методом «белого ящика». Тогда они будут называться инструментами динамического анализа кода. Либо средства анализа защищенности могут ничего не знать о внутреннем устройстве тестируемой системы, анализируя ее методом «черного ящика». В их основе лежит автоматизация и имитация взаимодействия пользователя с анализируемым приложением. В таком случае средства такого класса называются сканерами или фаззерами. Кроме того, средство анализа защищенности приложений может сочетать и статический, и динамический подходы.

Актуальность разработки средства гибридного тестирования защищенности веб-приложений заключается в том, что на данный момент не существует доступных бесплатных средств такого класса, которые могли бы сочетать и статический, и динамический подходы для тестирования защищенности веб-приложений от атак, направленных на эксплуатацию распространенных уязвимостей. Кроме того, это средство должно быть легко расширяемым для того, чтобы его можно было легко адаптировать под различные языки программирования и веб-фреймворки. Также оно должно обладать такой архитектурой и использовать такие компоненты, которые позволяли бы легко добавлять новые правила по выявлению потенциальных уязвимостей. Из вышеперечисленного становится ясно, что для решения задач, связанных с обеспечением защищенности приложений, необходим точный инструмент, обладающий большим функционалом.

В рамках выполнения выпускной квалификационной работы будут разработаны следующие модули:

- модуль построения синтаксической и семантической моделей исходного кода для языка программирования Java,
- веб-интерфейс для удобного отображения синтаксической и семантической моделей,
- модуль taint-анализа,
- модуль инструментирования исходного кода,
- модуль динамического тестирования веб-приложения.

Разрабатываемый инструмент позволит значительно улучшить процесс по обеспечению защищенности веб-приложений на стадии разработки, а именно позволит получать меньше ложных срабатываний при поиске потенциальных уязвимостей.

Также необходимо написать экономическое обоснование для разработки программного продукта «Средство гибридного тестирования защищенности веб-приложений на основе графа свойств кода».

## 9.2 Обоснование выбора аналога для сравнения

В качестве аналогов для сравнения были выбраны следующие средства статического анализа кода: «FindSecBugs» и «PVS-Studio». Они являются наиболее доступными и качественными среди всех существующих программных продуктов служащих для статического тестирования защищенности приложений.

Таблица 9.1 – Обоснование выбора аналога для сравнения

Критерии сравнения	FindSecBugs	PVS-Studio
Расширяемость функционала	+	–
Удобный интерфейс	+	+
Точность обнаружения потенциальных уязвимостей	+	+
Наличие инструментов отладки синтаксической и семантической моделей	–	–
Динамический анализ	–	+

Согласно таблице 9.1, наиболее близким и доступным аналогом является бесплатно распространяемое средство «FindSecBugs», с которым будет проводиться дальнейшее сравнение.

### 9.3 Сопоставление информационно-технических параметров аналога и разработки

Для сравнения аналога и разработки были выбраны критерии, которые представлены в таблице 9.2.

Таблица 9.2 – Критерии сравнения

Количественные параметры	Качественные параметры	Новые возможности
Количественные параметры отсутствуют (так как данный проект – это единичный ПП)	– Расширяемость функционала, – легкость использования, – точность обнаружения потенциальных уязвимостей	– Наличие инструментов отладки синтаксической и семантической моделей, – динамический анализ

Для определения показателя качества разработки и аналога необходимо определить интегральный технический показатель. Для вычисления интегрального технического показателя выберем аддитивную форму расчета. Формула для расчета интегрального технического показателя качества, показано в формуле (9.1).

$$I_T = \sum_{i=1}^n a_i b_i, \quad (9.1)$$

где  $a$  – весовой коэффициент  $i$ -го параметра;

$b$  – значение  $i$ -го параметра.

Для вычисления интегрального показателя выделим следующие технико-эксплуатационные параметры и их весовые коэффициенты:

- расширяемость функционала 0,2,
- легкость использования 0,1,
- точность обнаружения потенциальных уязвимостей 0,3,
- наличие инструментов отладки синтаксической и семантической моделей 0,1,
- динамический анализ 0,2.

Расчет интегрального показателя на основе технико-эксплуатационных параметров и коэффициентов весомости приведен в таблице 9.3.

Таблица 9.3 – Расчет интегрального технического показателя

Критерии сравнения	Весовой коэффициент $a_i$	Аналог «FindSecBugs»		Разрабатываемый проект «средство гибридного тестирования защищенности веб-приложений»	
		Показатель в баллах $b_i$	Интегральный показатель $I_t$	Показатель в баллах $b_i$	Интегральный показатель $I_t$
Расширяемость функционала	0,2	8	1,6	8	1,6
Легкость использования	0,1	8	0,8	3	0,3
Точность обнаружения потенциальных уязвимостей	0,3	5	1,5	9	2,7
Наличие инструментов отладки синтаксической и семантической моделей	0,1	0	0	9	0,9
Динамический анализ	0,3	0	0	9	2,7
Итого $I_T$	1	3,9		8,2	

Таким образом, можно высчитать коэффициент качества по формуле (9.2)

$$K_K = \sum_{i=1}^n a_i b_{ib} / \sum_{i=1}^n a_i b_{ie} = 2,1. \quad (9.2)$$



Согласно результатам расчета интегрального технического показателя, уровень качества предлагаемой разработки выше уровня качества базового аналога. Коэффициент качества больше единицы, следовательно, проект по техническим параметрам превосходит аналоги.

#### 9.4 Экономическая оценка разработки ПП

Для экономической оценки необходимо определить цену потребления, которая сводится к оценке требуемых единовременных капитальных затрат и затрат на эксплуатацию. Таким образом, в единовременные капитальные затраты конечного потребителя  $K$  входят суммой следующие составляющие:

$$K = C_0 + Z_{отл} + Z_{др} , \quad (9.3)$$

где  $C_0$  – цена разработки, руб.;

$Z_{отл}$  – стоимость операций по отладке программы, руб.;

$Z_{др}$  – дополнительные затраты, носящие единовременный характер (определяются в зависимости от особенностей использования ПП (на месте эксплуатации), руб.;

$K$  – капитальные затраты.

В свою очередь цена ПП определяется исходя из ее себестоимости, а также прибыли, которую определяет разработчик искомого ПП:

$$C_0 = CC + Pr_0 , \quad (9.4)$$

где  $C_0$  – цена единицы продукции;

$CC$  – затраты (себестоимость) на разработку ПП, руб.;

$Pr_0$  – доход (прибыль), руб.

## 9.5 Расчет себестоимости ПП

В первую очередь, для расчета затрат определим продолжительность каждой из работ, которые определяются по нормативам, либо рассчитываются по формуле:

$$t_o = \frac{3t_{\min} + 2t_{\max}}{5}, \quad (9.5)$$

где  $t_o$  – ожидаемая длительность работы,

$t_{\min}$  – наименьшая, по мнению эксперта, длительность работы;

$t_{\max}$  – наибольшая, по мнению эксперта, длительность работы.

Таблица 9.4 – Ожидаемые длительности работ на этапе проектирования.

№	Наименование работы	Длительность работы, дн.		
		$t_{\min.}$	$t_{\max.}$	$t_o$
1	Разработка технического задания	1	4	2
2	Анализ технического задания и сбор информации	1	6	3
3	Обзор аналогов *	1	6	3
4	Поиск и изучение технологий для реализации проекта *	7	25	14
5	Создание эскизного проекта *	6	11	8
6	Технический проект *	10	20	14
7	Рабочий проект *	47	57	51
8	Оформление пояснительной записки *	2	9	5

\* – работы, производимые с использованием вычислительной техники

Для определения продолжительности этапа проектирования ( $T_n$ ) построим график организации работ во времени. График представлен на рисунке 9.1.

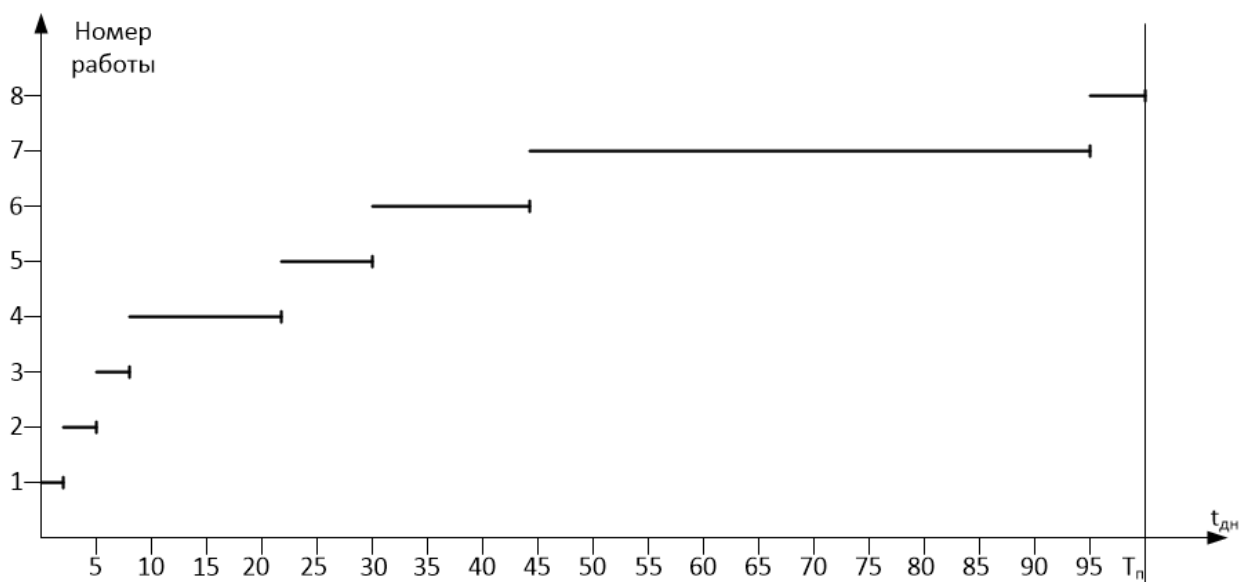


Рисунок 9.1 – График продолжительности этапа проектирования

Капитальные затраты на этапе проектирования рассчитываются по формуле (9.6).

$$Z_{\text{проект}} = Z / \text{пл}_{\text{проектир}} + M_{\text{проект}} + H_{\text{проект}}, \quad (9.6)$$

где  $Z_{\text{проект}}$  – затраты на всем этапе проектирования  $T_n$ ;

$Z / \text{пл}_{\text{проект}}$  – заработная плата проектировщика задачи на всем этапе проектирования  $T_n$ ;

$M_{\text{проект}}$  – затраты за использование ЭВМ на этапе проектирования  $T_n$ ;

$H_{\text{проект}}$  – накладные расходы на этапе проектирования  $T_n$ .

Одним из основных видов затрат на этапе проектирования является основная заработная плата проектировщика, которая рассчитывается по формуле:

$$Z / \text{пл}_{\text{проект}} (\text{осн}) = Z / \text{пл}_{\text{дн}} \cdot T_n, \quad (9.7)$$

где  $Z / \text{пл}_{\text{дн}}$  – дневная заработная плата разработчика задачи на этапе проектирования.

Затраты, связанные с использованием ЭВМ  $M_{\text{проект}}$  определяются по формуле 9.8:

$$M_{\text{проект}} = C_n \cdot t_n + C_d \cdot t_d, \quad (9.8)$$

где  $C_n$  и  $C_d$  – соответственно стоимость 1 часа процессорного и дисплейного времени;

$t_n$  и  $t_d$  –необходимое для решения задачи процессорное и дисплейное время соответственно (час).

Для разработки данной системы достаточно труда одного человека. Дневная заработная плата разработчика составляет 1500 рублей. Временные затраты на разработку и тестирование системы составляют 95 дней. Следовательно, основная заработная плата проектировщика должна быть равна 142500 рублей. Договор с исполнителем заключается на 6-часовой рабочий день.

Дополнительная заработная плата включает в себя премии, поощрительные выплаты за досрочное выполнение отдельных этапов и т.п. Дополнительная заработная плата составляет 10% от основной, что в данной разработке составит:  
 $142500 * 8/100\% = 11400$  руб.

ЭВМ используется на протяжении всего этапа проектирования. Стоимость 1 часа процессорного и дисплейного времени рассчитаем по таблице 9.5.

Таблица 9.5 – Суммы начисленной амортизации по используемому оборудованию, программному обеспечению и эксплуатационные расходы

Наименование оборудования	Стоимость оборудования (руб.)	Годовая норма амортизации (%)	Эффективный фонд времени работы оборудования (ч/год)	Время работы оборудования для разработки (ч)	Сумма (руб.)
Ноутбук Lenovo V580c	24000	20	3000	540	864
ИТОГО: 864 рубля					

Таким образом, 1 час процессорного времени равен 1 рублю. 1 час дисплейного времени равен 60 копейкам. Также за время разработки было использовано только бесплатно распространяемое ПО, а ОС Microsoft Windows 10 была уже установлена при покупке ЭВМ. Таким образом, Затраты, связанные с использованием ЭВМ,  $M_{проект}$  равны 864 рубля.

Накладные расходы определяются в процентном соотношении к фонду основной заработной платы на этапе проектирования, составляют 85% от заработной платы проектировщика и равны 121000 рублей.

Отчисления на социальные нужды в том числе страховые взносы в Пенсионный фонд РФ, Фонд социального страхования РФ и фонды обязательного медицинского страхования (федеральный и территориальный) составляют 30% от суммы основной и дополнительной заработной платы, составят:  $(142500 + 14250) * 30/100\% = 47250$  руб.

Так как в открытом доступе отсутствуют данные о стоимости разработки аналога разрабатываемого продукта, расчет затрат на разработку аналога производился в соответствии со средней заработной платой преподавателя.

Данные калькуляции себестоимости представлены в таблице 9.6.

Таблица 9.6 – Калькуляция себестоимости изготовления программного продукта

Наименование статьи калькуляции	Сумма, руб., разработка	Сумма,руб., аналог
1. Основная заработная плата разработчиков ПП	142500	158000
2. Дополнительная заработная плата разработчиков ПП (8%)	11400	12640
3. Социальные отчисления (30%)	47025	47400
4. Затраты за использование ЭВМ	864	864
5. Накладные расходы (85%)	121000	134300
Производственная себестоимость	322789	353204
6. Внепроизводственные расходы (2%)	6456	7064
Полная себестоимость	329245	360268

## 9.6 Расчет цены ПП

Определение цены продукции осуществляется на основе таблицы 9.7.

Таблица 9.7 – Определение цены продукции

Наименование статьи калькуляции	Сумма, руб., разработка	Сумма, руб., аналог
Полная себестоимость	329245	360268

Продолжение таблицы 9.7

Наименование статьи калькуляции	Сумма, руб., разработка	Сумма, руб., аналог
Закладываемая прибыль (7%)	23047	25218
Итого, продажная цена без НДС	352292	385486
НДС (18%)	63413	69387
Итого, продажная цена с НДС	415705	454873

Для получения итоговой цены используем формулу 9.9.

$$K = C_0 + Z_{отл} + Z_{др}, \quad (9.9)$$

где  $C_0$  – цена разработки, руб.;

$Z_{отл}$  – стоимость операций по отладке программы специалистом, руб.;

$Z_{др}$  – дополнительные затраты, носящие единовременный характер (определяются в зависимости от особенностей использования ПП (на месте эксплуатации), руб.

Цена разработки известна из таблицы 9.7, а стоимость операций по отладке и дополнительные затраты по предварительной оценке составляют 3000 руб и 2000 руб соответственно. Таким образом капитальные затраты рассчитываются по формуле 9.10.

$$K = 415705 + 3000 + 2000 = 420705 \text{ руб.} \quad (9.10)$$

## 9.7 Итоговое заключение по ТЭО

Стоимость разработки рассмотренных систем сопоставима. Разработанный продукт может коммерциализироваться следующими способами:

- продажа лицензии на использование продукта на одну или несколько ЭВМ,
- техническая поддержка на коммерческой основе.

Учитывая высокую потребность на производство программного обеспечения без уязвимостей, разработанный продукт будет иметь большой спрос на использование его

как компаниями по разработке ПО, так и компаниям, занимающимся только анализом защищенности приложений.

## ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было разработано средство гибридного тестирования защищенности веб-приложений на основе графа свойств кода. Были реализованы модули: построения синтаксической и семантической моделей исходного кода приложений на языке программирования Java, проведения taint-анализа, инструментирования исходного кода, динамического тестирования веб-приложения. Также было разработано минималистичное веб-приложение для удобного отображения синтаксической и семантической моделей.

При разработке использовалась модульная система, что позволяет удобно улучшать уже существующие модули и добавлять модули с новой функциональностью. Разработанный инструмент является больше прототипом, реализующим и демонстрирующим один из возможных подходов при анализе защищенности веб-приложений, чем законченным промышленным решением. В дальнейшем, во-первых, планируется добавить поддержку других языков программирования и фреймворков, наиболее используемых при разработке веб-приложений. Во-вторых, расширить функционал по поиску сложно-обнаруживаемых уязвимостей. И в-третьих, доработать пользовательский интерфейс для того, чтобы эксперт в большинстве случаев использовал инструмент в режиме нажатия «красной большой кнопки».



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Security Development Lifecycle – Википедия [Электронный ресурс]. – URL: [https://ru.wikipedia.org/wiki/Security\\_Development\\_Lifecycle](https://ru.wikipedia.org/wiki/Security_Development_Lifecycle) (дата обращения 20.04.2020).
2. Ayewah, Nathaniel; Hovemeyer, David; Morgenthaler, J. David; Penix, John; Pugh, William, Using Static Analysis to Find Bugs. IEEE Software, 2008, P. 22–29.
3. Какая разница между DevOps и DevSecOps? [Электронный ресурс]. – URL: <https://www.viva64.com/ru/b/0710/> (дата обращения 25.04.2020).
4. Марк Лутц, Изучаем Python, 4-е изд. – Пер. с англ. – СПб.: Символ-Плюс, 2011.
5. Прекращение поддержки Python 2 [Электронный ресурс]. – URL: <https://www.opennet.ru/opennews/art.shtml?num=52772> (дата обращения 27.04.2020).
6. Terence Parr, The Definitive ANTLR 4 Reference, 2012, P. 12-13.
7. Документация flask [Электронный ресурс]. – URL: <https://ru.wikibooks.org/wiki/Flask> (дата обращения 27.04.2020).
8. Сравнение Django и Flask [Электронный ресурс]. – URL: <https://techrocks.ru/2017/09/29/django-vs-flask-what-is-the-best-for-your-web-application/> (дата обращения 27.04.2020).
9. Первый релиз NoSQL БД OrientDB [Электронный ресурс]. – URL: <http://www.opennet.ru/opennews/art.shtml?num=33847> (дата обращения 01.05.2020).
10. Apache TinkerPop – Национальная библиотека им. Н.Э.Баумана [Электронный ресурс]. URL: [https://ru.bmstu.wiki/Apache\\_TinkerPop](https://ru.bmstu.wiki/Apache_TinkerPop) (дата обращения 02.05.2020).
11. Introduction to Redis [Электронный ресурс]. – URL: <https://redis.io/topics/introduction> (дата обращения 06.05.2020).
12. Абстрактное синтаксическое дерево [Электронный ресурс]. – URL: <https://www.viva64.com/ru/t/0004/> (дата обращения 25.04.2020).
13. S. Sparks, S. Embleton, R. Cunningham, and C. C. Zou, Automated vulnerability analysis: Leveraging control flow for evolutionary input crafting, In Proc. of Annual Computer Security Applications Conference (ACSAC), 2007, P. 477–486.
14. H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck. Structural detection of android malware using embedded callgraphs. In Proc. of ACM CCS Workshop on Artificial Intelligence and Security (AISEC), 2013.
15. Fabian Yamaguchi, Nico Golde, Daniel Arp and Konrad Rieck Modeling and Discovering Vulnerabilities with Code Property Graphs, 2014, P. 15.

16. Уязвимости и угрозы веб-приложений в 2019 году [Электронный ресурс]. – URL: <https://www.ptsecurity.com/ru-ru/research/analytics/web-vulnerabilities-2020/> (дата обращения 20.05.2020).
17. CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') [Электронный ресурс]. – URL: <https://cwe.mitre.org/data/definitions/79.html> (дата обращения 20.05.2020).
18. Excess XSS: комплексный учебник по межсайтовому скриптингу [Электронный ресурс]. – URL: <https://defcon.ru/web-security/3428/> (дата обращения 20.05.2020).
19. Sqlmap: SQL-инъекции — это просто [Электронный ресурс]. – URL: <https://xakep.ru/2011/12/06/57950/#toc01> (дата обращения 20.05.2020).
20. Внедрение команд ОС: понятие, эксплуатация, автоматизированный поиск уязвимости [Электронный ресурс]. URL: <https://hackware.ru/?p=1133> (дата обращения 20.05.2020).
21. Инструментация — эволюция анализа [Электронный ресурс]. – URL: <https://xakep.ru/2013/09/11/61232/> (дата обращения 05.05.2020).
22. Компаниец В.С. Учебно-методическое пособие по выполнению раздела «Безопасность человеко-машинного взаимодействия» в выпускных квалификационных работах студентов ИКТИБ – Таганрог: ИКТИБ ЮФУ, 2015. – 47 с.
23. СанПиН 2.2.2/2.4.1340–03. «Гигиенические требования к персональным электронно-вычислительным машинам и организации работы с дополнениями». – КонсультантПлюс [Электронный ресурс]. – URL: [http://www.consultant.ru/document/cons\\_doc\\_LAW\\_42836/](http://www.consultant.ru/document/cons_doc_LAW_42836/) (дата обращения 29.05.2020).
24. Р 2.2.2006-05 Гигиена труда. Руководство по гигиенической оценке факторов рабочей среды и трудового процесса. Критерии и классификация условий труда" (утв. Главным государственным санитарным врачом РФ 29.07.2005) – КонсультантПлюс [Электронный ресурс]. – URL: [http://www.consultant.ru/document/cons\\_doc\\_LAW\\_85537/](http://www.consultant.ru/document/cons_doc_LAW_85537/) (дата обращения 29.05.2020).

## ПРИЛОЖЕНИЕ А

### А.1 Главный программный модуль статического анализа

```
def runJClassesExtracting(projectConfig):
    db = Database(projectConfig)
    db.clear(DBCollections.JavaClasses)
    print("Obtaining Java classes info...")
    for dirname, dirnames, filenames in os.walk(projectConfig["target-dir"]):
        for filename in filenames:
            if not filename.endswith(".java"):
                continue
            filePath = os.path.join(dirname, filename)
            print("Handling: " + filePath)
            javaClassExtractor = JavaClassExtractor(projectConfig)
            javaClassExtractor.extractInfo(filePath)
            javaClassExtractor.dump()

    print("Done")
    print("Dumping database...")
    db.commit()

def runASTBuilding(projectConfig):
    db = Database(projectConfig)
    db.clear(DBCollections.ASTs)
    print("Building graphs...")
    for dirname, dirnames, filenames in os.walk(projectConfig["target-dir"]):
        for filename in filenames:
            if not filename.endswith(".java"):
                continue
            filePath = os.path.join(dirname, filename)
            print("Handling: " + filePath)
            print("Building AST...")
            astBuilder = ASTBuilder(projectConfig)
            astBuilder.build(filePath)
            astBuilder.dump()
            ast = astBuilder.getAST()
            packageName = ast.getProperty("package")
            baseName = os.path.splitext(filename)[0]
            ast.exportNew(f"{packageName}.{baseName}")
    print("Done")
    print("Dumping database...")
    db.commit()
    OrientDB(projectConfig).populateASTs()

def runCFGBuilding(projectConfig):
    db = Database(projectConfig)
    db.clear(DBCollections.CFGs)
    print("Building graphs...")
    for dirname, dirnames, filenames in os.walk(projectConfig["target-dir"]):
        for filename in filenames:
            if not filename.endswith(".java"):
                continue
            filePath = os.path.join(dirname, filename)
            print("Handling: " + filePath)
            print("Building CFGs...")
            cfgBuilder = CFGBuilder(projectConfig)
```

```

        cfgBuilder.build(filePath)
        cfgBuilder.dump()
        cfgs = cfgBuilder.getCFGs()
        for qn, CFG in cfgs.items():
            CFG.exportNew(filename=qn)
    print("Done")
    print("Dumping database...")
    db.commit()
    OrientDB(projectConfig).populateCFGs()

def runDFGBuilding(projectConfig):
    db = Database(projectConfig)
    db.clear(DBCollections.DFGs)
    print("Building graphs...")
    for dirname, dirnames, filenames in os.walk(projectConfig["target-dir"]):
        for filename in filenames:
            if not filename.endswith(".java"):
                continue
            filePath = os.path.join(dirname, filename)
            print("Handling: " + filePath)
            print("Building DFGs...")
            cfgs = db.getCFGsByFilePath(filePath)

            # Если нет функций и их CFG, то следовательно нет и DFG
            if len(cfgs) == 0:
                continue

            ast = db.getASTByFilePath(filePath)
            dfgBuilder = DFGBuilder(projectConfig)
            dfgBuilder.build(filePath, ast)
            dfgBuilder.dump()
            dfgs = dfgBuilder.getDFGs()
            for qn, DFG in dfgs.items():
                DFG.exportNew(filename=qn)
    dfgs = db.getAllDFGs()
    DFGBuilder.addIPDataFlows(dfgs, projectConfig)
    print("Done")
    print("Dumping database...")
    db.commit()
    OrientDB(projectConfig).populateDFGs()

def runTaintFlowAnalysis(projectConfig):
    astSources = SourcesManager(projectConfig).getSources()
    astSinks = SinksManager(projectConfig).getSinks()

    print(f"Found {len(astSources)} sources")
    if len(astSources) == 0:
        return

    print(f"Found {len(astSinks)} sinks")
    if len(astSinks) == 0:
        return

    gremlin = Gremlin(projectConfig)
    db = Database(projectConfig)
    db.clear(DBCollections.TaintFlows)

    taintFlows = []
    for astSink in astSinks:
        print(astSink.getOptionalProperty("sinkText") + " in file " +

```

```

astSink.getFile() + " at line " + str(
    astSink.getLineOfCode()) + " (sharedId: " + astSink.getSharedId() + ")")
dftp, targetDFGName = gremlin.findASTNodeInDFG(astSink.sharedId)
print(f" DFG-node sharedId: {dftp.getSharedId()}")

    if astSink.getOptionalProperty("args"):
        dftp.setOptionalProperty("checkpoint",
astSink.getOptionalProperty("args")[0])
    elif astSink.getOptionalProperty("assignmentExpression"):
        dftp.setOptionalProperty("checkpoint",
astSink.getOptionalProperty("assignmentExpression"))

    for astSource in astSources:
        print("\t" + astSource.getOptionalProperty(
            "sourceText") + " in file " + astSource.getFile() + " at line " +
str(
            astSource.getLineOfCode()) + " (sharedId: " + astSource.getSharedId()
+ ")")
        dfsp, sourceDFGName = gremlin.findASTNodeInDFG(astSource.sharedId)
        print(f"\t\t\tDFG-node sharedId: {dfsp.getSharedId()}")
        if checkDFReachability(gremlin, dfsp.getSharedId(), dftp.getSharedId()):
            taintFlows.append(dict(
                source=dfsp, sink=dftp,
vulnerability=astSink.getOptionalProperty("vulnerability"))
            )

    taintFlows = deleteDuplicateTaintFlows(taintFlows)
    db.setAllTaintFlows(taintFlows)
    print("Dumping to database...")
    db.commit()

def runCallgraphAnalysis(projectConfig):
    db = Database(projectConfig)
    db.clear(DBCollections.CallGraph)
    gremlin = Gremlin(projectConfig)
    javaClasses = db.getAllJavaClasses()
    for jc in javaClasses.values():
        packageName = jc.package
        className = jc.name
        for method in jc.methods:
            methodQN = f"{packageName}.{className}.{method.name}"
            print(f"Searching for callees for {methodQN} ...")
            gResp = gremlin.g.V().hasLabel("ASTNode").has("kind", "CLASS").where(
                __.out().has("kind", "NAME").has("code", className)
            ) \
                .out().has("kind", "METHOD").where(
                __.out().has("kind", "NAME").has("code", method.name)
            ).repeat(__.out()).emit().has("kind", "CALL").out().has("kind",
"NAME").values("code").toList()
            if len(gResp) > 0:
                db.putInCallGraph(methodQN, gResp)
    db.commit()

def main():
    if len(sys.argv) < 2:
        print("Usage: python %s <command> [options]" % sys.argv[0])
        return

    command = sys.argv[1]
    if command == "init":
        if len(sys.argv) < 3:

```

```

        print("Usage: python %s init <project-name>" % sys.argv[0])
        return

    projectName = sys.argv[2]
    print(f"Creating a project with name '{projectName}'...")

    if os.path.exists(projectName):
        print(f"Directory with name '{projectName}' is existed")
        return

    os.mkdir(projectName)
    os.chdir(projectName)
    with open(Config.TEMPLATE_PROJECT_CONFIG) as f:
        projectConfigJson = json.load(f)
        projectConfigJson["name"] = projectName
        projectConfigJson["DB"] = f"{projectName}.db"
        projectConfigJson["orientdb-name"] = projectName
    with open(Config.PROJECT_CONFIG_FILENAME, "w") as f:
        json.dump(projectConfigJson, f, indent=4)
    elif command == "run-static":
        with open(Config.PROJECT_CONFIG_FILENAME) as f:
            projectConfig = json.load(f)

        subcommand = sys.argv[2]
        if subcommand == "all":
            runJClassesExtracting(projectConfig)
            runASTBuilding(projectConfig)
            runCFGBuilding(projectConfig)
            runDFGBuilding(projectConfig)
            runTaintFlowAnalysis(projectConfig)
            runCallgraphAnalysis(projectConfig)
        elif subcommand == "classes":
            runJClassesExtracting(projectConfig)
        elif subcommand == "ast":
            runASTBuilding(projectConfig)
        elif subcommand == "cfg":
            runCFGBuilding(projectConfig)
        elif subcommand == "dfg":
            runDFGBuilding(projectConfig)
        elif subcommand == "taint":
            runTaintFlowAnalysis(projectConfig)
        elif subcommand == "callgraph":
            runCallgraphAnalysis(projectConfig)

    elif command == "web":
        with open(Config.PROJECT_CONFIG_FILENAME) as f:
            projectConfig = json.load(f)

        runWebApp(projectConfig)

if __name__ == "__main__":
    main()

```

## ПРИЛОЖЕНИЕ Б

### Б.1 Запускающая часть модуля построения AST

```
class ASTBuilder:
    def __init__(self, projectConfig):
        self.projectConfig = projectConfig
        self.ast = None

    def getAST(self):
        return self.ast

    def build(self, filePath) -> AbstractSyntaxTree:
        inputStream = FileStream(filePath)
        lexer = JavaLexer(inputStream)
        tokens = CommonTokenStream(lexer)
        parser = JavaParser(tokens)
        parseTree = parser.compilationUnit()
        ast = AbstractSyntaxTree()
        ast.setProperty("filePath", filePath)
        visitor = ASTVisitor(ast)
        visitor.visit(parseTree)

        packageName = ast.getProperty("package")
        baseName = os.path.basename(os.path.splitext(filePath)[0])
        qualifiedName = f"{packageName}.{baseName}"

        for v in ast.nodes:
            v.setFile(qualifiedName)

        self.ast = ast

    def dump(self):
        db = Database(self.projectConfig)
        packageName = self.ast.getProperty("package")
        filePath = self.ast.getProperty("filePath")
        baseName = os.path.basename(os.path.splitext(filePath)[0])
        qualifiedName = f"{packageName}.{baseName}"
        db.putAST(os.path.basename(qualifiedName), self.ast)
```

### Б.2 Фрагмент Посетителя для построения AST

```
class ASTVisitor(JavaParserVisitor):
    def __init__(self, ast: AbstractSyntaxTree):
        self.ast = ast
        self.parentStack = Stack()
        self.ast.getRoot().setCode("Filename")
        self.parentStack.push(self.ast.getRoot())
        self.typeModifier = ""
        self.memberModifier = ""
        self.vars = dict()
        self.varsCounter = 0

# *****
# ***                                DECLARATIONS                                ***
```

```

# *****

def visitPackageDeclaration(self, ctx: JavaParser.PackageDeclarationContext):
    # packageDeclaration: annotation* PACKAGE qualifiedName ';'
    self.ast.setProperty("package", ctx.qualifiedName().getText())
    packageNode = ASNode(ASNodeKind.PACKAGE)
    packageNode.setCode(ctx.qualifiedName().getText())
    packageNode.setLineOfCode(ctx.start.line)
    packageNode.setSharedId(ctx)
    self.ast.addVertex(packageNode)
    self.ast.addEdge(Edge(self.parentStack.peek(), None, packageNode))

def visitImportDeclaration(self, ctx: JavaParser.ImportDeclarationContext):
    # importDeclaration: IMPORT STATIC? qualifiedName ('.' '*' )? ';'
    qualifiedName = ctx.qualifiedName().getText()
    last = ctx.getChildCount() - 1
    if ctx.getText()[last - 1] == "*" and ctx.getText()[last - 2] == ".":
        qualifiedName += ".*"
    importNode = ASNode(ASNodeKind.IMPORT)
    importNode.setCode(qualifiedName)
    importNode.setLineOfCode(ctx.start.line)
    importNode.setSharedId(ctx)
    self.ast.addVertex(importNode)
    self.ast.addEdge(Edge(self.parentStack.peek(), None, importNode))

```



## ПРИЛОЖЕНИЕ В

### В.1 Запускающая часть модуля построения CFG

```
class CFGBuilder:
    def __init__(self, projectConfig):
        self.projectConfig = projectConfig
        self.CFGs = dict()

    def getCFGs(self):
        return self.CFGs

    def build(self, filePath: str) -> Dict[str, ControlFlowGraph]:
        inputStream = FileStream(filePath)
        lexer = JavaLexer(inputStream)
        tokens = CommonTokenStream(lexer)
        parser = JavaParser(tokens)
        parseTree = parser.compilationUnit()
        cfgs = dict()
        visitor = CFGVisitor(cfgs, filePath=filePath)
        visitor.visit(parseTree)

        for qn, CFG in cfgs.items():
            for v in CFG.nodes:
                v.setMethod(qn)

            CFG.setProperty("filePath", filePath)

        self.CFGs = cfgs

    def dump(self):
        db = Database(self.projectConfig)
        for qn, CFG in self.CFGs.items():
            db.putCFG(qn, CFG)
```

### В.2 Фрагмент Посетителя для построения CFG

```
class CFGVisitor(JavaParserVisitor):
    def __init__(self, cfgs: List[ControlFlowGraph], filePath: str):
        self.cfgs = cfgs
        self.filePath = filePath
        self.preNodes = Stack()
        self.preEdgeKinds = Stack()
        self.loopBlocks = Stack()
        self.labeledBlocks = []
        self.tryBlocks = Queue()
        self.classNames = Stack()
        self.dontPop = False
        self.casesQueue = Queue()

        self.currentCFG = None
        self.currentMethodName = None
        self.packageName = None
```

```

def visitMethodDeclaration(self, ctx:JavaParser.MethodDeclarationContext):
    # methodDeclaration
    #      : typeTypeOrVoid IDENTIFIER formalParameters ('[' ' '])*
    #      (THROWS qualifiedNameList)?
    #      methodBody

    self.init()
    entry = CFNode(CFNodeKind.ENTRY)
    entry.setLineOfCode(ctx.start.line)
    entry.setFile(self.filePath)
    retType = ctx.typeTypeOrVoid().getText()
    args = getOriginalCodeText(ctx.formalParameters())
    entry.setCode(retType + " " + ctx.IDENTIFIER().getText() + args)
    entry.setSharedId(ctx)
    self.currentCFG = ControlFlowGraph()
    self.currentMethodName = ctx.IDENTIFIER().getText()
    self.currentCFG.setProperty("className", self.classNames.peek())
    self.currentCFG.setProperty("methodName", self.currentMethodName)
    self.currentCFG.addVertex(entry)
    entry.setOptionalProperty("name", ctx.IDENTIFIER().getText())
    entry.setOptionalProperty("class", self.classNames.peek())
    entry.setOptionalProperty("type", retType)

    self.preNodes.push(entry)
    self.preEdgeKinds.push(CFEdgeKind.EPS)
    self.visitChildren(ctx)
    qualifiedName =
f"{self.packageName}.{self.classNames.peek()}.{self.currentMethodName}"
    logger.info(f"Building CFG for {qualifiedName} method...")
    self.cfgs[qualifiedName] = self.currentCFG
    self.currentCFG = None
    self.currentMethodName = None

def visitLocalVariableDeclaration(self,
ctx:JavaParser.LocalVariableDeclarationContext):
    # LocalVariableDeclaration: variableModifier* typeType variableDeclarators
    # variableDeclarators: variableDeclarator (',' variableDeclarator)*
    # variableDeclarator: variableDeclaratorId ('=' variableInitializer)?

    for varCtx in ctx.variableDeclarators().variableDeclarator():
        declr = CFNode(CFNodeKind.ASSIGN)
        declr.setLineOfCode(varCtx.start.line)
        declr.setFile(self.filePath)
        declr.setCode(ctx.typeType().getText() + " " +
getOriginalCodeText(varCtx) + ";")
        declr.setSharedId(varCtx)
        self.addNodeAndPreEdge(declr)
        self.preEdgeKinds.push(CFEdgeKind.EPS)
        self.preNodes.push(declr)
    return None

```

## ПРИЛОЖЕНИЕ Г

### Г.1 Запускающая часть модуля построения DFG

```
class DFGBuilder:
    def __init__(self, projectConfig):
        self.projectConfig = projectConfig
        self.DFGs = dict()

    def getDFGs(self):
        return self.DFGs

    def build(self, filePath: str, ast: AbstractSyntaxTree) -> Dict[str,
DataFlowGraph]:
        inputStream = FileStream(filePath)
        lexer = JavaLexer(inputStream)
        tokens = CommonTokenStream(lexer)
        parser = JavaParser(tokens)
        parseTree = parser.compilationUnit()
        logger.info("Iterative DEF-USE analysis ... ")
        dfgs = dict()
        iteration = 0
        while True:
            iteration += 1
            changed = False
            visitor = DFGVisitor(iteration, dfgs, ast, filePath,
Database(self.projectConfig), self.projectConfig)
            visitor.visit(parseTree)
            changed |= visitor.changed
            logger.debug("Iteration #" + str(iteration) + ": " + ("CHANGED" if
changed else "NO-CHANGE"))
            logging.debug("=====")
            if not changed:
                break
        logger.info("Done.")

        db = Database(self.projectConfig)
        for qn, dfg in dfgs.items():
            cfg = db.getCFG(qn)
            dfg.attachCFG(cfg)

        logger.info("Adding data-flows...")
        DFGBuilder.addDataFlowEdges(dfgs)

        for qn, dfg in dfgs.items():
            for v in dfg.nodes:
                v.setMethod(qn)

        dfg.setProperty("filePath", filePath)

        self.DFGs = dfgs

    @staticmethod
    def addDataFlowEdges(ddgs: Dict[str, DataFlowGraph]):
        visitedDefs = set()
        for qn, ddg in ddgs.items():
            cfg = ddg.getCFG()
            logger.info(f"Handling {qn} CFG...")
            visitedDefs.clear()
```

```

entry = cfg.getEntry()
defTraversal = CFPATHTraversal(cfg, entry)
while defTraversal.hasNext():
    defCFNode = defTraversal.next()

    if defCFNode.Id in visitedDefs:
        defTraversal.continueNextPath()
        continue
    visitedDefs.add(defCFNode.Id)

    defDDNode = ddg.getNodeByID(defCFNode.getSharedId())
    if defDDNode is None:
        continue

    if len(defDDNode.getAllDEFs()) == 0 and not
defDDNode.containsIPDEFs():
        continue

    # first add any self-flows of this node
    for flow in defDDNode.getAllSelfFlows():
        ddg.addEdge(Edge(defDDNode, flow, defDDNode))

    # now traverse the CFG for any USEs till a DEF
    visitedUses = set()
    for DEF in defDDNode.getAllDEFs():
        useTraversal = CFPATHTraversal(cfg, defCFNode)
        visitedUses.clear()
        useCFNode = useTraversal.next()
        visitedUses.add(useCFNode.Id)
        while useTraversal.hasNext():
            useCFNode = useTraversal.next()
            useDDNode = ddg.getNodeByID(useCFNode.getSharedId())
            if useDDNode is None:
                continue
            if useDDNode.hasDEF(DEF):
                useTraversal.continueNextPath() # no need to
continue this path
            if useCFNode.Id in visitedUses:
                useTraversal.continueNextPath() # no need to
continue this path
            else:
                visitedUses.add(useCFNode.Id)
                if useDDNode.hasUSE(DEF):
                    ddg.addEdge(DFEdge(defDDNode, DEF, useDDNode,
DFEdgeKind.INTRA))

    @staticmethod
    def addIPDataFlows(ddgs: Dict[str, DataFlowGraph], projectConfig):
        db = Database(projectConfig)
        for qn, ddg in ddgs.items():
            for node in ddg.nodes:
                if node.IP_DEFS is not None:
                    IPDFTarget =
db.getDFGNodeBySharedId(node.IP_DEFS["entrySharedId"])
                    ddg.addEdge(DFEdge(
                        node, "inter-procedural", IPDFTarget, DFEdgeKind.INTER)
                    )
            db.putDFG(qn, ddg)

    def dump(self):
        db = Database(self.projectConfig)

```

```

for qn, dfg in self.DFGs.items():
    db.putDFG(qn, dfg)

```

## Г.2 Фрагмент Посетителя для построения AST

```

class DFGVisitor(JavaParserVisitor):
    def __init__(self, iteration, ddgs: List[DataFlowGraph],
                 ast: AbstractSyntaxTree, filePath: str, db: Database,
projectConfig):
        self.analysisVisit = False
        self.iteration = iteration
        self.ddgs = ddgs
        self.localVars = []
        self.useList = set()
        self.defList = set()
        self.selfFlowList = set()
        self.changed = False
        self.methodParams = []
        self.activeClasses = Stack()
        self.ast = ast
        self.filePath = filePath
        self.db = db
        self.projectConfig = projectConfig

        self.currentDFG = None
        self.currentMethod = None
        self.packageName = None

    def analyseDefUse(self, node, expression):
        logger.debug("--- ANALYSIS ---")
        logger.debug(node)
        self.analysisVisit = True
        expr = self.visit(expression)
        logger.debug(expr)

        localVarStr = ""
        localVarStr += "LOCAL VARS = ["
        for lv in self.localVars:
            localVarStr += lv.type + " " + lv.name + ", "
        localVarStr += "]"
        logger.debug(localVarStr)

        if isUsableExpression(expr):
            self.useList.add(expr)
            logger.debug("USABLE")

        self.analysisVisit = False
        logger.debug("Changed = " + str(self.changed))
        logger.debug("DEFS = " + ' '.join(node.getAllDEFS()))
        logger.debug("USES = " + ' '.join(node.getAllUSES()))

        for DEF in self.defList:
            status = self.isDefined(DEF)
            if status > -1:
                self.changed |= node.addDEF(DEF)
            else:
                logger.debug(f"{DEF} is not defined!")

```

```

logger.debug("Changed = " + str(self.changed))
logger.debug("DEFS = " + ' '.join(node.getAllDEFS()))

for USE in self.useList:
    status = self.isDefined(USE)
    if status > -1:
        self.changed |= node.addUSE(USE)
    else:
        logger.debug(f"{USE} is not defined!")
logger.debug("Changed = " + str(self.changed))
logger.debug("USES = " + ' '.join(node.getAllUSES()))

for selfFlow in self.selfFlowList:
    status = self.isDefined(selfFlow)
    if status > -1:
        self.changed |= node.addSelfFlow(selfFlow)
    else:
        logger.debug(f"{selfFlow} is not defined!")
logger.debug("Changed = " + str(self.changed))
logger.debug("SelfFlows = " + ' '.join(node.getAllSelfFlows()))

self.defList.clear()
self.useList.clear()
self.selfFlowList.clear()
logger.debug("-----")

def visitLocalVariableDeclaration(self, ctx:
JavaParser.LocalVariableDeclarationContext):
    # LocalVariableDeclaration : variableModifier* typeType variableDeclarators
    # variableDeclarators : variableDeclarator (',' variableDeclarator)*
    # variableDeclarator : variableDeclaratorId ('=' variableInitializer)?

    # if self.analysisVisit:
    #     return self.visit(ctx.variableDeclarators())

    for var in ctx.variableDeclarators().variableDeclarator():
        self.localVars.append(JavaField(
            None,
            False,
            self.visit(ctx.typeType()),
            var.variableDeclaratorId().IDENTIFIER().getText()
        ))

    if self.analysisVisit:
        return self.visit(ctx.variableDeclarators())

    if self.iteration == 1:
        declr = DFNode()
        declr.setLineOfCode(var.start.line)
        declr.setFile(self.filePath)
        declr.setCode(getOriginalCodeText(var))
        declr.setSharedId(var)
        self.currentDFG.addVertex(declr)
    else:
        declr = self.currentDFG.getNodeByCtx(var)

    self.analyseDefUse(declr, var)
return None

```

## ПРИЛОЖЕНИЕ Д

### Д.1 Модуль Taint-анализа для поиска источников

```
class SourcesManager:
    def __init__(self, projectConfig):
        self.projectConfig = projectConfig
        self.gremlin = Gremlin(projectConfig)

    def getSources(self):
        astSources = []
        if self.projectConfig["web-framework"] == "Struts2":
            db = Database(self.projectConfig)
            for classQN, jc in db.getAllJavaClasses().items():
                if hasSuperClass(classQN, "ActionSupport", db):
                    for field in jc.fieldList:
                        getter = "get" + field.name[0].upper() + field.name[1:]
                        getterQN = f"{classQN}.{getter}"
                        astSources.extend(findSourceByMethodQNCall(getterQN,
self.gremlin))
                        astSources.extend(findSourceInParams(field.name,
self.gremlin))

        elif self.projectConfig["web-framework"] == "SpringMVC":
            endpointExtractor = SpringMVCEndpointExtractor(self.projectConfig)
            endpointExtractor.extractRouteData()
            controllersData = endpointExtractor.getRouteData()

            for item in controllersData:
                astSources.extend(findSourceForSpringMVC(
                    classQN=item["class"],
                    methodName=item["method"],
                    params=item["params"],
                    gremlin=self.gremlin)
                )

        else:
            print("For web framework '%s' does not implement source search" %
self.projectConfig["web-framework"])

            seen_sharedIds = set()
            new_list = []
            for obj in astSources:
                if obj.sharedId not in seen_sharedIds:
                    new_list.append(obj)
                    seen_sharedIds.add(obj.sharedId)

            return new_list
```

### Д.2 Методы-паттерны для поиска источников при Taint-анализе

```
def findSourceByMethodQNCall(methodQN, gremlin: Gremlin):
    className, methodName = methodQN.split(".")[1:]
    packageName = ".".join(methodQN.split(".")[0:-2])
    g = gremlin.g
```

```

callNameNodes = g.V().hasLabel("ASTNode").has("kind", "ROOT").where(
    __.out().has("kind", "PACKAGE").has("code", packageName)
).out().has("kind", "CLASS").where(
    __.out().has("kind", "NAME").has("code", className)
).repeat(__.out()).emit(__.has("kind", "CALL").where(
    __.out().has("kind", "NAME").has("code", methodName)
)).valueMap().toList()

results = []
for cnn in callNameNodes:
    astNode = gremlin.deserializeASTNode(cnn)
    astNode.setOptionalProperty("sourceText", f'Call of method "{methodName}"')
    results.append(astNode)

return results

def findSourceInParams(name, gremlin):
    g = gremlin.g
    params = g.V().hasLabel("ASTNode").has("code", name).where(
        __.in_().has("kind", "PARAMS")
    ).valueMap().toList()

    results = []
    for p in params:
        astNode = gremlin.deserializeASTNode(p)
        astNode.setOptionalProperty("sourceText", f'Parameter "{name}"')
        results.append(astNode)

    return results

def findSourceForSpringMVC(classQN, methodName, params, gremlin):
    className = classQN.split(".")[1]
    g = gremlin.g
    results = []
    for param in params:
        gResp = g.V().hasLabel("ASTNode").has("kind", "CLASS").where(
            __.out().has("kind", "NAME").has("code", className)
        ).out().has("kind", "METHOD").where(
            __.out().has("kind", "NAME").has("code", methodName)
        ).out().has("kind", "PARAMS").out().has("kind", "VARIABLE").where(
            __.out().has("kind", "NAME").has("code", param)
        ).valueMap().toList()

        if len(gResp) == 0:
            continue

        gResp = gResp[0]
        astNode = gremlin.deserializeASTNode(gResp)
        astNode.setOptionalProperty("sourceText", f'Parameter "{param}" of method "{methodName}"')
        results.append(astNode)

    return results

```

### Д.3 Модуль Taint-анализа для поиска чувствительных инструкций (sinks)



```

class SinksManager:
    def __init__(self, projectConfig):
        self.projectConfig = projectConfig
        self.gremlin = Gremlin(projectConfig)

    def getSinks(self):
        astSinks = []
        if self.projectConfig["web-framework"] == "Struts2":
            astSinks = findCreateQuery(self.gremlin)

            db = Database(self.projectConfig)
            for classQN, jc in db.getAllJavaClasses().items():
                if hasSuperClass(classQN, "ActionSupport", db):
                    for field in jc.fieldList:
                        setter = "set" + field.name[0].upper() + field.name[1:]
                        setterQN = f"{classQN}.{setter}"
                        astSinks.extend(findSinkByMethodQNCall(setterQN,
self.gremlin))
                        astSinks.extend(findSinkInAssingments(field.name,
self.gremlin))

            astSinks.extend(findSaveMethodCalls(self.gremlin))
            astSinks.extend(findExec(self.gremlin))

        elif self.projectConfig["web-framework"] == "SpringMVC":
            astSinks = findExecuteQuery(self.gremlin)

        seen_sharedIds = set()
        new_list = []
        for obj in astSinks:
            if obj.sharedId not in seen_sharedIds:
                new_list.append(obj)
                seen_sharedIds.add(obj.sharedId)

        return new_list

```

#### Д.4 Методы-паттерны для поиска чувствительных инструкций при Taint-анализе

```

# -----
# ---          SQL-injections          ---
# -----
def findCreateQuery(gremlin: Gremlin):
    g = gremlin.g
    found = g.V().hasLabel("ASTNode").has("kind", "CALL").where(
        __.out().has("kind", "NAME").has("code", "createQuery")
    ).valueMap().toList()

    results = []
    for f in found:
        astNode = gremlin.deserializeASTNode(f)
        astNode.setOptionalProperty("sinkText", f'Call of method "createQuery"')
        astNode.setOptionalProperty("vulnerability", "SQL")
        results.append(astNode)
    return results

```

```

def findExecuteQuery(gremlin: Gremlin):
    g = gremlin.g
    found = g.V().hasLabel("ASTNode").has("kind", "CALL").where(
        __.and_(
            __.out().has("kind", "NAME").has("code", "executeQuery"),
            __.out().has("kind", "PARAMS")
        )
    ).valueMap().toList()

    results = []
    for f in found:
        astNode = gremlin.deserializeASTNode(f)
        astNode.setOptionalProperty("sinkText", f'Call of method "executeQuery"')
        astNode.setOptionalProperty("vulnerability", "SQL")
        results.append(astNode)
    return results

# -----
# ---                                XSS                                ---
# -----
def findSinkByMethodQNCall(methodQN, gremlin: Gremlin):
    # Пока не будет замечать, что классы могут иметь методы с одинаковыми именами
    className, methodName = methodQN.split(".")[1:-2:]
    packageName = ".".join(methodQN.split(".")[1:-2])
    g = gremlin.g

    callNameNodes = g.V().hasLabel("ASTNode").has("kind", "ROOT").where(
        __.out().has("kind", "PACKAGE").has("code", packageName)
    ).out().has("kind", "CLASS").where(
        __.out().has("kind", "NAME").has("code", className)
    ).repeat(__.out()).emit(__.has("kind", "CALL").where(
        __.out().has("kind", "NAME").has("code", methodName)
    )).valueMap().toList()

    results = []
    for cnn in callNameNodes:
        astNode = gremlin.deserializeASTNode(cnn)
        astNode.setOptionalProperty("sinkText", f'Call of method "{methodName}"')
        astNode.setOptionalProperty("vulnerability", "XSS")
        results.append(astNode)

    return results

def findSinkInAssignments(name, gremlin):
    assignments = gremlin.g.V().hasLabel("ASTNode").has("code", "products").where(
        __.repeat(__.in_()).until(__.has("kind", "ASSIGN_LEFT"))
    ).as_("left_part").repeat(__.in_()).until(__.has("kind",
"ASSIGN")).values("optionalProperties")\
        .as_("right_part").select("left_part",
"right_part").by(valueMap()).by().toList()

    results = []
    for assignment in assignments:
        astNode = gremlin.deserializeASTNode(assignment["left_part"])
        astNode.setOptionalProperty("sinkText", f'Variable "{name}" assignment')
        astNode.setOptionalProperty("vulnerability", "XSS")
        astNode.setOptionalProperty("assignmentExpression",
json.loads(assignment["right_part"])["assignmentExpression"])
        results.append(astNode)

    return results

```

```

def findSaveMethodCalls(gremlin: Gremlin):
    g = gremlin.g
    calls = g.V().hasLabel("ASTNode").has("kind", "CALL").where(__.out().has("kind",
"NAME").has("code", "save"))\
        .valueMap().toList()

    results = []
    for call in calls:
        astNode = gremlin.deserializeASTNode(call)
        astNode.setOptionalProperty("sinkText", f'Call of method "save"')
        astNode.setOptionalProperty("vulnerability", "XSS")
        results.append(astNode)

    return results

# -----
# ---                  command injections          ---
# -----
def findExec(gremlin: Gremlin):
    g = gremlin.g
    calls = g.V().hasLabel("ASTNode").has("kind", "CALL").where(
        __.out().has("kind", "NAME").has("code", "exec")
    ).valueMap().toList()

    results = []
    for call in calls:
        astNode = gremlin.deserializeASTNode(call)
        astNode.setOptionalProperty("sinkText", f'Call of method "exec"')
        astNode.setOptionalProperty("vulnerability", "CI")
        results.append(astNode)

    return results

```

## ПРИЛОЖЕНИЕ Е

### Е.1 Главный модуль динамического анализа

```
class DynamicAnalyzer:
    def __init__(self, projectConfig):
        self.projectConfig = projectConfig
        self.endpointExtractor = None
        self.attackEndpointExtractor = None
        self.webDriver = None

    def setEndpointExtractor(self, endpointExtractor):
        self.endpointExtractor = endpointExtractor

    def setAttackEndpointExtractor(self, attackEndpointExtractor):
        self.attackEndpointExtractor = attackEndpointExtractor

    def setWebDriver(self, webDriver):
        self.webDriver = webDriver

    def run(self):
        self.checkReadiness()

        # First
        self.endpointExtractor.extractEndpoints()
        self.endpointExtractor.dump()

        self.attackEndpointExtractor.extractEndpoints()
        self.attackEndpointExtractor.dump()

        # Second
        checkpointManager = CheckpointManager(projectConfig=self.projectConfig)
        checkpointManager.prepare()
        instrumentTool = InstrumentTool(self.projectConfig, checkpointManager)
        instrumentTool.instrument()

        input("Run an application. Enter an any key while it is ready up...")

        # Third
        self.webDriver.authenticate()
        self.webDriver.prepare()
        attacker = Attacker(self.webDriver, checkpointManager, self.projectConfig)
        attacker.attack()

        input("Stop an application. Enter an any key while it is ready up...")

        # Fourth
        instrumentTool.revert()

    def checkReadiness(self):
        if self.endpointExtractor is None:
            print("Not defined an EndpointExtractor. Use setEndpointExtractor method")
            exit(1)

        if self.attackEndpointExtractor is None:
            print("Not defined an AttackEndpointExtractor. Use setAttackEndpointExtractor method")
            exit(1)
```

```

if self.webDriver is None:
    print("Not defined an WebDriver. Use setWebDriver method")
    exit(1)

```

## E.2 Модуль динамического анализа для инструментирования

```

class GitManager:
    def __init__(self, projectConfig):
        self.projectConfig = projectConfig
        self.repo = Repo(projectConfig["REPO"])

    def prepare(self):
        git = self.repo.git
        self.saveCurrentBranch()
        git.checkout("HEAD", b="instrument")
        git.add(update=True)
        self.repo.index.commit("init state")

    def clear(self):
        print("Clearing...")
        git = self.repo.git
        self.repo.git.reset('--hard')
        self.repo.index.reset(commit="HEAD~1")
        git.checkout(self.loadSavedBranch())
        git.branch("-D", "instrument")

    def saveCurrentBranch(self):
        branch = self.repo.active_branch.name
        with open(self.projectConfig["SAVED_BRANCH_FILE"], "w") as f:
            f.write(branch)

    def loadSavedBranch(self):
        with open(self.projectConfig["SAVED_BRANCH_FILE"]) as f:
            branch = f.read()
            return branch

class InstrumentTool:
    def __init__(self, projectConfig, checkpointManager=None):
        self.projectConfig = projectConfig
        self.checkpointManager = checkpointManager

    def instrument(self):
        gitManager = GitManager(self.projectConfig)
        gitManager.prepare()

        with open(self.projectConfig["DB"]) as f:
            taintFlows = json.load(f)["taintFlows"]

        sinks = [tf["sink"] for tf in taintFlows]
        uniqueSinks = []
        for sink in sinks:
            if sink not in uniqueSinks:
                uniqueSinks.append(sink)

        infos = []
        for sink in uniqueSinks:

```

```

        info = {
            "id": self.checkpointManager.sinkToCheckpoint[sink["sharedId"]],
            "line": sink["line"],
            "file": sink["file"],
            "argument": sink["optionalProperties"]["checkpoint"]
        }
        infos.append(info)

    groups = dict()
    for info in infos:
        if info["file"] in groups:
            groups[info["file"]].append(info)
        else:
            groups[info["file"]] = [info]

    for file, infoGroup in groups.items():
        with open(file) as f:
            content = f.readlines()

        lineWithPackage = 0
        for idx, line in enumerate(content):
            if line.startswith("package"):
                lineWithPackage = idx

        content.insert(lineWithPackage + 1, "import
org.khbr.checkpointstorage.CheckPointStorage;\n")

        infoGroup = sorted(infoGroup, key=lambda x: x["line"])

        checkpointLineOffset = 0 # 1 потому что уже была вставка import
        for info in infoGroup:
            Id = info["id"]
            argument = info["argument"]
            checkpoint = f"new CheckPointStorage().setValue(\"{Id}\",
{argument});\n"
            content.insert(info["line"] + checkpointLineOffset, checkpoint)
            checkpointLineOffset += 1

        with open(file, "w") as f:
            f.write("\n".join(content))

    def revert(self):
        gitManager = GitManager(self.projectConfig)
        gitManager.clear()

```

### Е.3 Модуль динамического анализа для управления контрольными точками

```

class CheckpointManager:
    def __init__(self, projectConfig):
        self.projectConfig = projectConfig
        self.sinkToCheckpoint = dict()

    def prepare(self):
        with open(self.projectConfig["DB"]) as f:
            taintFlows = json.load(f)["taintFlows"]

        sinks = [tf["sink"] for tf in taintFlows]
        uniqueSinks = []

```

```

for sink in sinks:
    if sink not in uniqueSinks:
        uniqueSinks.append(sink)

for idx, us in enumerate(uniqueSinks):
    self.sinkToCheckpoint[us["sharedId"]] = idx + 1
    r = redis.Redis()
    r.set(f"ch_{idx + 1}", "")

def getCheckpointValue(self, sharedId: str) -> str:
    r = redis.Redis()
    checkpointId = self.sinkToCheckpoint[sharedId]
    checkpointValue = r.get(f"ch_{checkpointId}").decode()
    return checkpointValue

```

## E.4 Модуль динамического анализа для проведения атаки

```

class Attacker:
    def __init__(self, webDriver: WebDriver, checkpointManager: CheckpointManager,
projectConfig):
        self.checkpointManager = checkpointManager
        self.attackEndpoints = []
        self.loadAttackEndpoints()
        self.webDriver = webDriver
        self.projectConfig = projectConfig
        self.report = Report()

    def attack(self):
        for attackEndpoint in self.attackEndpoints:
            vulnerability = attackEndpoint['vulnerability']
            endpointUri = attackEndpoint["uri"]
            fullUri = f"{self.projectConfig['base_url']}{endpointUri}"
            print(f"Testing {endpointUri}...")
            # Safe-запрос
            payload = dict()
            for testParam in attackEndpoint["params"]:
                print(f'\tTesting parameter "{testParam}"...')
                for param in attackEndpoint["params"]:
                    if param == testParam:
                        payload[param] = "test"
                    else:
                        payload[param] = "COMMON_PARAM_VALUE"

            self.webDriver.post(fullUri, data=payload)

            if vulnerability == "SQL":
                checker = InjectionChecker()
            elif vulnerability == "XSS":
                checker = XSSChecker()
            else:
                checker = CIChecker()

            checkpointValue =
self.checkpointManager.getCheckpointValue(attackEndpoint["sinkSharedId"])
            checker.setSafeQuery(checkpointValue)

            # Unsafe-запрос
            if vulnerability == "SQL":

```

```

        payload[testParam] = "' or 1=1 -- sample"
    elif vulnerability == "XSS":
        payload[testParam] = "some<script>alert(4);</script>"
    else:
        payload[testParam] = "localhost; ls"

    self.webDriver.post(fullUri, data=payload)

    checkpointValue =
self.checkpointManager.getCheckpointValue(attackEndpoint["sinkSharedId"])
    if not checker.checkPotentialUnsafeQuery(checkpointValue):
        self.report.addRecord(attackEndpoint, testParam)
        line = attackEndpoint['sinkLine']
        file = attackEndpoint['sinkFile']
        code = attackEndpoint['sinkCode']
        sharedId = attackEndpoint['sinkSharedId']
        print(f"Potential vulnerability ({vulnerability}) has been
found!!!! URI: {endpointUri}\n"
              f"\tin file {file} at line {line} ({sharedId})\n"
              f"\t{code}")
        print("-----")
    self.report.dump()

def loadAttackEndpoints(self):
    with open(Config.ATTACK_ENDPOINTS_FILE) as f:
        self.attackEndpoints = json.load(f)

```

## E.5 Модуль для проверки на XSS-уязвимость

```

class XSSChecker:
    def __init__(self):
        self.rightNumberOfTokens = None

    def setSafeQuery(self, query):
        self.rightNumberOfTokens = self.getNumberOfTokens(query)

    def checkPotentialUnsafeQuery(self, query):
        numberOfTokens2 = self.getNumberOfTokens(query)
        return self.rightNumberOfTokens == numberOfTokens2

    def getNumberOfTokens(self, _query):
        inputStream = InputStream(_query.upper())
        lexer = HTMLLexer(inputStream)
        tokens = CommonTokenStream(lexer)
        parser = HTMLParser(tokens)
        parseTree = parser.htmlContent()
        return len(tokens.tokens)

```

## E.6 Модуль для проверки на уязвимость типа SQL-инъекция

```

class InjectionChecker:
    def __init__(self):
        self.rightNumberOfTokens = None

```



```

def setSafeQuery(self, query):
    self.rightNumberOfTokens = self.getNumberOfTokens(query)

def checkPotentialUnsafeQuery(self, query):
    numberOfTokens2 = self.getNumberOfTokens(query)
    return self.rightNumberOfTokens == numberOfTokens2

def getNumberOfTokens(self, _query):
    inputStream = InputStream(_query.upper())
    lexer = MySqlLexer(inputStream)
    tokens = CommonTokenStream(lexer)
    parser = MySqlParser(tokens)
    parseTree = parser.root()
    return len(tokens.tokens)

```

## E.7 Модуль для проверки на уязвимость класса внедрение команд OS

```
METACHARS = [";", "&&", "||"]
```

```

class CIChecker:
    def __init__(self):
        self.rightNumberOfMetaChars = None

    def setSafeQuery(self, query):
        self.rightNumberOfTokens = self.getNumberOfMetaChars(query)

    def checkPotentialUnsafeQuery(self, query):
        numberOfTokens2 = self.getNumberOfMetaChars(query)
        return self.rightNumberOfTokens == numberOfTokens2

    def getNumberOfMetaChars(self, _query):
        numberOfMetaChars = 0
        for metaChar in METACHARS:
            numberOfMetaChars += _query.count(metaChar)
        return numberOfMetaChars

```