**Онлайн-школа Skillfactory**

Санников Александр Игнатьевич, группа 1

**Выпускная квалификационная работа**

**по направлению ДПО «Информационная безопасность»**

по теме

**«Метод и система статического анализа для поиска уязвимостей в программном обеспечении»**

Студент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.И. САННИКОВ

подпись И.О. Фамилия

|  |  |
| --- | --- |
| Рецензент  ученая степень, звание  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  И.О. Фамилия | Руководитель  ученая степень, звание (при наличии)  С. СЕРОВ  И.О. Фамилия  Консультант  ученая степень, звание (при наличии)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  И.О. Фамилия |
| Москва 2024 г. | |

**СОДЕРАЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc158564401)

[ПОСТАНОВКА ЗАДАЧИ 5](#_Toc158564402)

[ОБЩАЯ ХАРАКТЕРИСТИКА КОМПАНИИ AWESOME SOFT LTD. 6](#_Toc158564403)

[ТЕХНИЧЕСКОЕ ЗАДАНИЕ 8](#_Toc158564404)

[МЕТОДОЛОГИЯ РАБОТЫ SAST 9](#_Toc158564405)

[SAST И DAST: ОТЛИЧИЯ 11](#_Toc158564406)

[ИНТЕГРАЦИЯ SAST В CI/CD 14](#_Toc158564407)

[ОБЗОР СОВРЕМЕННЫХ РЕШЕНИЙ СТАТИЧЕСКОГО АНАЛИЗА 16](#_Toc158564408)

[СКАНЕР BANDIT 17](#_Toc158564409)

[КОД РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ-СКАНЕРА SAST 18](#_Toc158564410)

[ПОДГОТОВКА К ТЕСТИРОВАНИЮ ПРИЛОЖЕНИЯ-СКАНЕРА SAST 20](#_Toc158564411)

[ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ-СКАНЕРА SAST 22](#_Toc158564412)

[ЗАКЛЮЧЕНИЕ 29](#_Toc158564413)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 30](#_Toc158564414)

[Приложение 1 *Отзыв руководителя на ВКР* 32](#_Toc158564415)

[Приложение 2 *Рецензии на ВКР* 33](#_Toc158564416)

ВВЕДЕНИЕ

В современном цифровом мире, где программное обеспечение играет важную роль в нашей повседневной жизни, обеспечение безопасности становится неотъемлемой необходимостью. Однако, даже самые продвинутые и надежные программные продукты могут содержать уязвимости, которые могут быть использованы злоумышленниками для несанкционированного доступа или нанесения вреда.

Статический анализ на уязвимости в программном обеспечении является мощным инструментом, который помогает выявить и исправить потенциальные слабые места в коде еще на стадии разработки. Он позволяет разработчикам исследовать и анализировать программный код, выявляя различные типы уязвимостей, такие как недостаточная проверка входных данных, отсутствие аутентификации или авторизации, и другие.

Несмотря на то, что существуют различные типы уязвимостей, некоторые из них являются особенно распространенными и имеют серьезные последствия. Например, SQL-инъекции могут позволить злоумышленникам получить несанкционированный доступ к базе данных, а межсайтовые скрипты могут использоваться для выполнения вредоносного кода на уязвимых веб-страницах. Кроме того, уязвимости в аутентификации могут привести к несанкционированному доступу к аккаунтам пользователей и утечке конфиденциальной информации.

Одним из наиболее известных и разрушительных примеров уязвимости SQL-инъекции является атака на компанию Equifax в 2017 году.

Equifax – крупнейшее агентство по кредитным отчетам в США. Злоумышленники использовали SQL-инъекции, чтобы получить несанкционированный доступ к базе данных. Уязвимость возникла из-за неправильной фильтрации и проверки входных данных, которые были переданы в SQL-запросы.

Атака позволила злоумышленникам получить доступ к конфиденциальной информации о более чем 143 миллионах человек, включая имена, социальные страховые номера, даты рождения, адреса и другие личные данные. Это привело к серьезным убыткам для компании Equifax и пострадавших пользователей, так как их личные данные были скомпрометированы.

Последствия этой атаки были ощутимы и влияли на доверие людей к системам хранения персональных данных. Компания Equifax столкнулась с многочисленными судебными исками, штрафами и утратой репутации. Так, например, пользователи, чьи данные были скомпрометированы, могут столкнуться с риском мошенничества, их финансовая безопасность может быть под угрозой.

Этот пример демонстрирует, насколько важно обращать внимание на уязвимости и принимать меры для защиты программного обеспечения от различных атак.

Цель данной дипломной работы заключается в разработке автоматизированного скрипта, который использует инструмент статического анализа для поиска уязвимостей. В работе будут рассмотрены основные виды уязвимостей, их причины, а также методы и инструменты статического анализа, которые помогут разработчикам обнаружить и устранить эти уязвимости.

ПОСТАНОВКА ЗАДАЧИ

В рамках дипломной работы я решил уделить внимание практической стороне вопроса статического анализа программного кода (SAST). Для этого было принято предложение разработать скрипт (приложение-сканер) SAST на языке Python для поиска уязвимостей в программном коде файлов приложений и мобильных игр для компании Awesome Soft Ltd.

ОБЩАЯ ХАРАКТЕРИСТИКА КОМПАНИИ AWESOME SOFT LTD.



**Рисунок 1. Логотип Awesome Soft Ltd.**

Компания Awesome Soft Ltd. - IT-компания, существующая на рынке 5 лет. Она имеет головной офис в Москве и два филиала - в Санкт-Петербурге и Екатеринбурге. Основная выручка компании поступает от продажи премиум-подписок к собственной уникальной платформе по биржевой торговле Awesome Stocks MOEX. Эта платформа предоставляет пользователям возможность торговать на Московской Фондовой Бирже (MOEX), Санкт-Петербургской Фондовое Бирже (SPBE), а также выход на Нью-Йоркскую и Гонконгскую биржи через брокеров-партнеров. Премиум-подписки предлагают дополнительные функциональные возможности, эксклюзивный контент и аналитические инструменты на основе рекомендаций искусственного интеллекта, которые помогают пользователям принимать решения при торговле на бирже с максимальной для себя выгодой.

В настоящее время фирма планирует открыть для себя новую нишу - разработка собственных игр для мобильных платформ (iOS и Android). Для этого выделяются значительные материальные и интеллектуальные ресурсы.

Основной фокус компании в новой для себя сфере — это разработка качественных и увлекательных игр, которые привлекают широкую аудиторию пользователей и предлагают уникальные игровые механики, захватывающий геймплей и привлекательную графику.

Сочетание разработки игр и продуктов для финансового рынка позволяет компании Awesome Soft Ltd. диверсифицировать источники дохода и создавать мощные возможности для роста и развития. Компания стремится к удовлетворению потребностей своих клиентов, предлагая им инновационные продукты, надежные услуги и отличный пользовательский опыт.

Awesome Soft Ltd. стремится к инновациям и постоянному улучшению своих продуктов и услуг. Компания активно развивает свои команды разработчиков, дизайнеров и аналитиков, чтобы предоставить высококачественные и конкурентоспособные продукты.

Awesome Soft Ltd. активно участвует в сообществе разработчиков и индустрии игр и финансовых технологий. Компания организует мероприятия, участвует в конференциях и взаимодействует с конкурентами и экспертами в этих областях. Цель компании - не только быть успешной в своем бизнесе, но и вносить вклад в развитие индустрии и способствовать инновациям в сфере финансов и игр.

В целом, Awesome Soft Ltd. является динамичной и инновационной компанией, специализирующейся на разработке игр для мобильных платформ и предоставлении премиум-подписок к платформе по биржевой торговле. Сочетание этих двух направлений позволяет компании достигать успеха и роста, привлекая широкую аудиторию и предлагая качественные продукты и услуги.

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

На основе пожеланий заказчика, представителей компании Awesome Soft Ltd., было составлено техническое задание для разработчиков, содержащее пожелания и требования к скрипту на Python.

Описание задачи:

Разработать скрипт на языке Python, который в автоматизированном режиме будет выполнять следующие действия:

1. Получать код из указанного GitHub репозитория.

2. Запускать инструмент статического анализа (SAST) для анализа полученного кода.

3. На основе полученных результатов анализа cформировать документ с краткой информацией о проведенной проверке. Документ может быть в любом формате (например, Markdown, HTML, PDF).

Требования:

1. Скрипт должен быть написан на языке Python.

2. Для получения кода из GitHub репозитория необходимо использовать соответствующий API или библиотеку.

3. Для выполнения статического анализа кода рекомендуется использовать один из популярных инструментов, таких как Bandit или Pylint. Выбор конкретного инструмента остается на усмотрение разработчика.

4. Результаты анализа должны быть представлены в форме документа с краткой информацией о найденных уязвимостях или о том, что уязвимостей не обнаружено.

5. Скрипт должен иметь возможность принимать параметры, такие как URL репозитория и путь к выходному документу.

6. Скрипт должен быть написан с учетом принципов чистого кода и быть простым в использовании и поддержке.

Пожелания:

1. Разработчик может добавить дополнительные функции или улучшения по своему усмотрению, которые могут повысить эффективность или удобство использования скрипта.

МЕТОДОЛОГИЯ РАБОТЫ SAST

SAST (статический анализ кода в рамках безопасности приложений) — это методология тестирования, которая позволяет обнаруживать уязвимости и потенциальные проблемы безопасности в программном коде на ранних стадиях разработки. Ниже приведены основные этапы при проведении SAST:

1. Сбор и подготовка кода. Исходный код программы собирается и подготавливается для анализа. Это может включать в себя получение кода из репозитория, извлечение исходных файлов и их организацию в удобную для анализа структуру.
2. Лексический анализ. Код разбивается на лексические элементы, такие как ключевые слова, идентификаторы, операторы и т. д. Это позволяет построить дерево разбора кода и определить его структуру.
3. Синтаксический анализ. Происходит анализ синтаксической структуры кода. Синтаксический анализатор проверяет, соответствует ли код грамматическим правилам языка программирования и создает абстрактное синтаксическое дерево (AST), которое представляет структуру программы.
4. Анализ потока данных. Происходит анализ потока данных внутри программы. Анализируются пути передачи данных, переменных, их использование и обработка. Это позволяет выявить потенциальные уязвимости, связанные с неправильным использованием данных.
5. Анализ контекста. Происходит анализ контекста выполнения программы. Это включает в себя проверку правильного использования API, обработку исключений, безопасное хранение и передачу данных, и другие аспекты, связанные с безопасностью приложений.
6. Обнаружение уязвимостей. На основе проведенного анализа SAST-инструмент обнаруживает потенциальные уязвимости в коде. Это могут быть такие проблемы, как неправильная обработка пользовательского ввода, возможность инъекций, отсутствие проверок авторизации и аутентификации и другие.
7. Формирование отчета. После обнаружения уязвимостей SAST-инструмент формирует отчет, который содержит информацию о найденных проблемах. Отчет обычно включает в себя описание уязвимости, место в коде, где она обнаружена, и рекомендации по устранению проблемы.

SAST является мощным инструментом для обнаружения уязвимостей в программном коде на ранних стадиях разработки и позволяет разработчикам принять меры по обеспечению безопасности приложений. Однако важно понимать, что SAST не является идеальным и может давать ложноположительные или ложноотрицательные результаты. Поэтому важно сочетать SAST с другими методами тестирования безопасности, такими как DAST (динамическое тестирование безопасности приложений) и ручное тестирование.

SAST И DAST: ОТЛИЧИЯ

Отличие статического анализа кода от динамического анализа заключается в том, что:

1. Статический анализ происходит на этапе разработки, до фактического выполнения кода. Он основан на анализе исходного кода и его структуры, а также на применении определенных правил и эвристик для обнаружения потенциальных проблем.

2. Динамический анализ происходит во время выполнения программы. Он включает запуск программы с набором тестовых данных и мониторинг ее поведения. Динамический анализ позволяет обнаружить проблемы, которые могут возникать только во время выполнения, такие как ошибки доступа к памяти или утечки ресурсов.

Основные преимущества статического анализа кода на уязвимости включают:

- Обнаружение проблем на ранних этапах разработки, что помогает предотвратить их появление в финальной версии программы.

- Возможность обнаруживать потенциальные уязвимости в коде, которые могут быть вызваны некорректным использованием языка программирования или неправильными алгоритмами.

- Увеличение общего уровня безопасности программного кода и защита от известных типов уязвимостей.

Однако, статический анализ кода не способен обнаружить все возможные проблемы и уязвимости, и может давать ложные срабатывания. Поэтому рекомендуется комбинировать статический анализ с динамическим тестированием для достижения наилучших результатов в обеспечении безопасности кода.

Вот таблица, в которой сравниваются технические аспекты статического анализа и динамического анализа при тестировании программного обеспечения:

**Таблица 1. Сравнение статического и динамического анализа.**

| **Аспект** | **Статический анализ** | **Динамический анализ** |
| --- | --- | --- |
| *Определение* | Проверяет код, не выполняя его | Включает в себя тестирование программного обеспечения путем выполнения кода. |
| *Методология* | Проверка синтаксиса, обзоры кода, сопоставление с образцом | Модульные тесты, интеграционные тесты, системные тесты |
| *Типы* | Нерабочий | Время выполнения |
| *Инструменты* | Линтеры, статические анализаторы (например, SonarQube, ESLint) | Отладчики, профилировщики производительности (например, Valgrind) |
| *Фокус* | Качество кода, уязвимости безопасности | Функциональность, производительность, ошибки выполнения |
| *Обнаружены ошибки* | Ошибки времени компиляции, нарушения безопасности, запах кода | Логические ошибки, исключения во время выполнения, утечки ресурсов |
| *Время обратной связи* | Немедленно (до начала выполнения) | Во время или после исполнения |
| *Требования к ресурсам* | Обычно требует меньше ресурсов | Может потребоваться больше ресурсов (например, мощности сервера) |
| *Случаи использования* | Ранние стадии разработки, до выполнения кода | Пост-разработка, контроль качества перед выпуском |
| *Зернистость* | Может быть более детальным, анализируя конкретные блоки кода. | Смотрит на поведение программы в целом |
| *Автоматизация* | Высокая автоматизация | Может быть автоматизировано, но может также потребоваться ручное тестирование. |
| *Анализ зависимостей* | Ограничено зависимостями кода без фактического потока данных. | Включает тестирование потока данных и взаимодействия. |
| *Анализ пути выполнения* | Анализирует потенциальные пути выполнения | Анализирует фактический путь выполнения, пройденный во время теста. |
| *Имитация пользовательской среды* | Не имитирует | В некоторой степени имитирует среду пользователя |
| *Время бежать* | В целом быстро | Может занять больше времени в зависимости от сложности |
| *Сложность настройки* | Менее сложный, нет необходимости в тестовой среде | Более сложный, требует настройки тестовой среды. |
| *Исправление ошибок* | Помогает выявить проблемы на ранней стадии, их легче исправить | Может выявлять ошибки, которые возникают только при выполнении |

ИНТЕГРАЦИЯ SAST В CI/CD

Статический анализ кода на уязвимости может быть встроен в процесс непрерывной интеграции и доставки (CI/CD) для автоматического обнаружения потенциальных проблем в коде на ранних этапах разработки. Существует несколько вариантов такой интеграции:

1. Добавить шаг SAST в систему CI/CD, чтобы он выполнялся автоматически при каждой сборке или развертывании. Например, например можно использовать такие инструменты, как Bandit, Pylint, SonarQube и др.

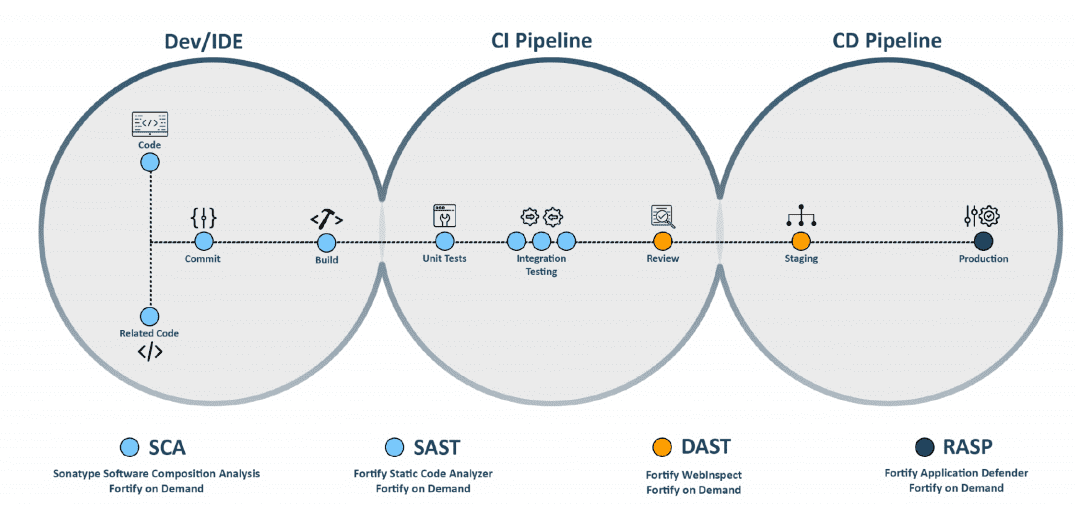
2. При интеграции SAST в систему CI/CD, можно настроить набор правил анализа, которые определяют, какие уязвимости и проблемы будут обнаруживаться. Так можно выбрать только те правила, которые наиболее релевантны для конкретного проекта и требований безопасности.

3. Можно настроить уровень серьезности угроз для обнаруженных уязвимостей, чтобы определить, насколько критичными они считаются. Например, установить высокий уровень серьезности для критических уязвимостей, чтобы они требовали немедленного внимания и исправления.

4. Интеграция с уведомлениями и системами отслеживания ошибок: при обнаружении уязвимостей статическим анализатором кода, можно настроить систему CI/CD таким образом, чтобы отправлять уведомления о проблемах разработчикам или команде безопасности. Также возможно интегрировать инструменты SAST с системой отслеживания ошибок, чтобы проблемы были автоматически зарегистрированы и отслеживались.

5. Включение статического анализа в пайплайн проверки кода, который выполняется перед отправкой изменений в основную ветку или перед созданием pull-запроса. Это позволит обнаруживать проблемы на ранней стадии, еще до того, как код попадет в основную ветку разработки или репозиторий.

Интеграция статического анализа кода на уязвимости в процесс CI/CD позволяет автоматически обнаруживать и предотвращать потенциальные проблемы безопасности на ранних этапах разработки и обеспечивать высокий уровень безопасности вашего кода.



**Рисунок 2. Интеграция SAST в CI/CD.**

ОБЗОР СОВРЕМЕННЫХ РЕШЕНИЙ СТАТИЧЕСКОГО АНАЛИЗА

1. Pylint: инструмент статического анализа кода для языка Python. Pylint может обнаруживать различные проблемы в коде, включая стилевые нарушения и потенциальные ошибки.

2. SonarQube: платформа для непрерывного контроля качества кода. SonarQube поддерживает множество языков программирования и может выполнять статический анализ кода, искать уязвимости и другие проблемы.

3. FindBugs: инструмент предназначен для статического анализа кода на языке Java. Он может обнаруживать различные проблемы, такие как потенциальные ошибки, утечки памяти и проблемы с производительностью.

4. ESLint: инструмент статического анализа кода для JavaScript. ESLint может проверять код на соответствие различным стандартам кодирования, обнаруживать потенциальные ошибки и другие проблемы.

5. Cppcheck: инструмент статического анализа кода на языке C++. Cppcheck может обнаруживать различные проблемы, такие как некорректное использование указателей, утечки памяти и другие ошибки.

6. Bandit: сканер безопасности для Python-кода, который можно использовать для обнаружения распространенных проблем безопасности. Он анализирует Python-код и сообщает о потенциальных проблемах безопасности, таких как уязвимости, небезопасные криптографические методы и др.

СКАНЕР BANDIT

Сканер Bandit — это инструмент для статического анализа кода, разработанный для обнаружения потенциальных уязвимостей в программном коде на языке Python. Он используется для поиска уязвимостей без необходимости запуска кода.

К его преимуществам можно отнести:

1. Простота использования: Bandit предоставляет простой и интуитивно понятный интерфейс командной строки, что делает его легким в освоении и использовании даже для начинающих разработчиков.

2. Широкий спектр проверок: Bandit предлагает множество встроенных правил анализа кода, которые позволяют обнаруживать различные типы уязвимостей, такие как потенциальные уязвимости безопасности, неправильное использование API, возможности для атаки внедрения кода и т. д.

3. Гибкость настройки: Bandit позволяет настраивать набор правил анализа кода, чтобы отобразить специфические требования для конкретного проекта. Можно выбрать только нужные правила или добавить собственные правила, чтобы адаптировать инструмент под специфические потребности.

4. Интеграция с CI/CD: Bandit может быть легко интегрирован в процессы непрерывной интеграции и развертывания (CI/CD), позволяя автоматически проверять код на наличие уязвимостей перед его публикацией или развертыванием.

5. Поддержка сообщества: Bandit разрабатывается и поддерживается сообществом, что означает, что вы можете рассчитывать на обновления и улучшения инструмента со временем.

КОД РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ-СКАНЕРА SAST

По условиям задачи был написан следующий скрипт на языке Python:

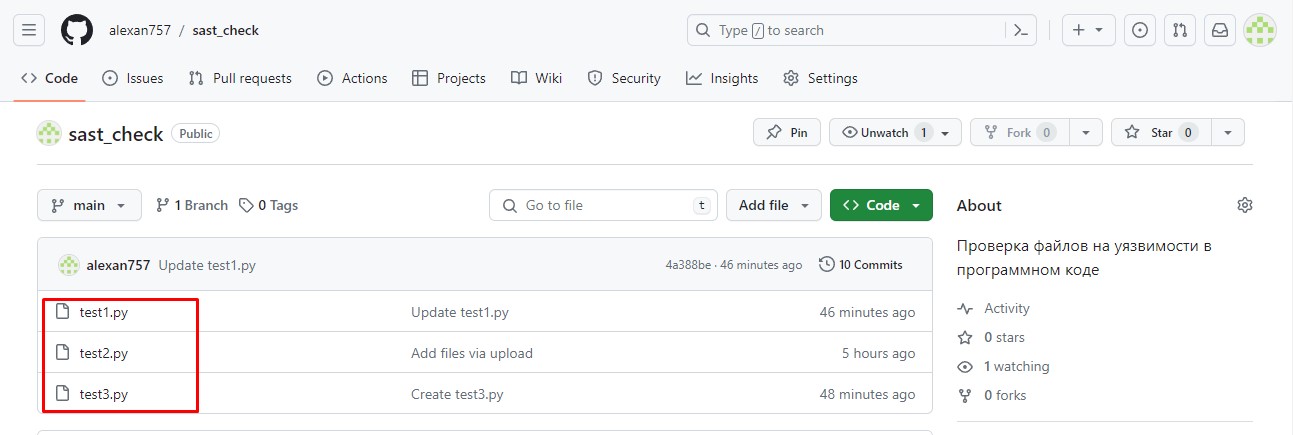
import subprocess  
import argparse  
import logging  
import requests  
import os  
import pathlib  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 *# Разбиваем URL репозитория на части  
 # parts = args.repository.split('/')* start\_url='https://github.com/alexan757/sast\_check'  
  
 parts = start\_url.split('/')  
 username = parts[-2]  
 repo\_name = parts[-1].replace('.git', '')  
  
 *# Получаем файлы репозитория через GitHub API* files\_url = f'https://api.github.com/repos/{username}/{repo\_name}/contents'  
 print(files\_url)  
 response = requests.get(files\_url).json()  
 *# Создаем директорию для сохранения файлов* current\_dir = pathlib.Path(\_\_file\_\_).parent.resolve()  
target\_dir = os.path.join(current\_dir, f'{repo\_name}\_files')  
 os.makedirs(target\_dir, exist\_ok=True)  
  
 *# Скачиваем и сохраняем файлы для анализа* for file\_info in response:  
 if file\_info['type'] == 'file':  
 file\_url = file\_info['download\_url']  
 file\_name = file\_info['name']  
 file\_path = os.path.join(target\_dir, file\_name)  
  
 file\_content = requests.get(file\_url).text  
with open(file\_path, 'w', encoding='utf-8') as f:  
 f.write(file\_content)  
  
 *# Определяем путь к файлам для анализа* response = subprocess.run(['bandit', file\_path], stdout=subprocess.PIPE).stdout.decode('utf-8')  
 print(response)  
 print("I've done!")  
  
 *# Сохраняем результаты анализа в файл* with open(f'{repo\_name}\_{file\_name}\_report\_.md.txt', 'w', encoding='utf-8') as f:  
 f.write(response)

Скрипт состоит из пяти ключевых этапов:

1. Разделение URL репозитория на части: домен-имя пользователя-название репозитория.
2. Получение списка файлов репозитория через GitHub API.
3. Создание директории на локальной машине и загрузка файлов из репозитория для анализа.
4. Подключение модуля Bandit для сканирования файлов на уязвимости.
5. Сохранение результатов сканирования на уязвимости в файл отчета.

ПОДГОТОВКА К ТЕСТИРОВАНИЮ ПРИЛОЖЕНИЯ-СКАНЕРА SAST

Перед началом тестирования нашего приложения в репозиторий <https://github.com/alexan757/sast_check> были загружены 3 файла: test1.py, test2.py, test3.py, которые содержат уязвимости в программном коде.



**Рисунок 3. Файлы для анализа в репозитории.**

Ниже приведен исходный код каждого файла.

Test1.py

import requests  
  
api\_key = "YOUR\_API\_KEY"  
  
def get\_data():  
 url = "https://api.example.com/data"  
 headers = {  
 "Authorization": f"Bearer {api\_key}"  
 }  
 response = requests.get(url, headers=headers)  
 return response.json()  
  
data = get\_data()  
  
import xml.etree.ElementTree as ET  
  
xml\_data = """  
<!DOCTYPE foo [  
 <!ENTITY xxe SYSTEM "file:///etc/passwd">  
]>  
<root>  
 <data>&xxe;</data>  
</root>  
"""  
  
def process\_xml():  
 root = ET.fromstring(xml\_data)  
 data = root.find("data").text  
 return data  
  
result = process\_xml()  
print(result)

Test2.py

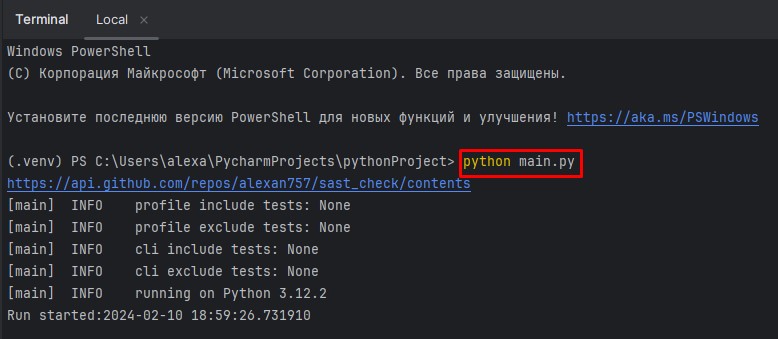
from django.db import connection  
  
def find\_user(username):  
  
 with connection.cursor() as cur:  
  
 cur.execute(f"""select username from USERS where name = '%s'""" % username)  
  
 output = cur.fetchone()  
  
 return output

Test3.py

import requests  
  
internal\_url = "http://43.56.10.200/api/data"  
  
def get\_internal\_data():  
 response = requests.get(internal\_url)  
 return response.json()  
  
data = get\_internal\_data()  
print(data)

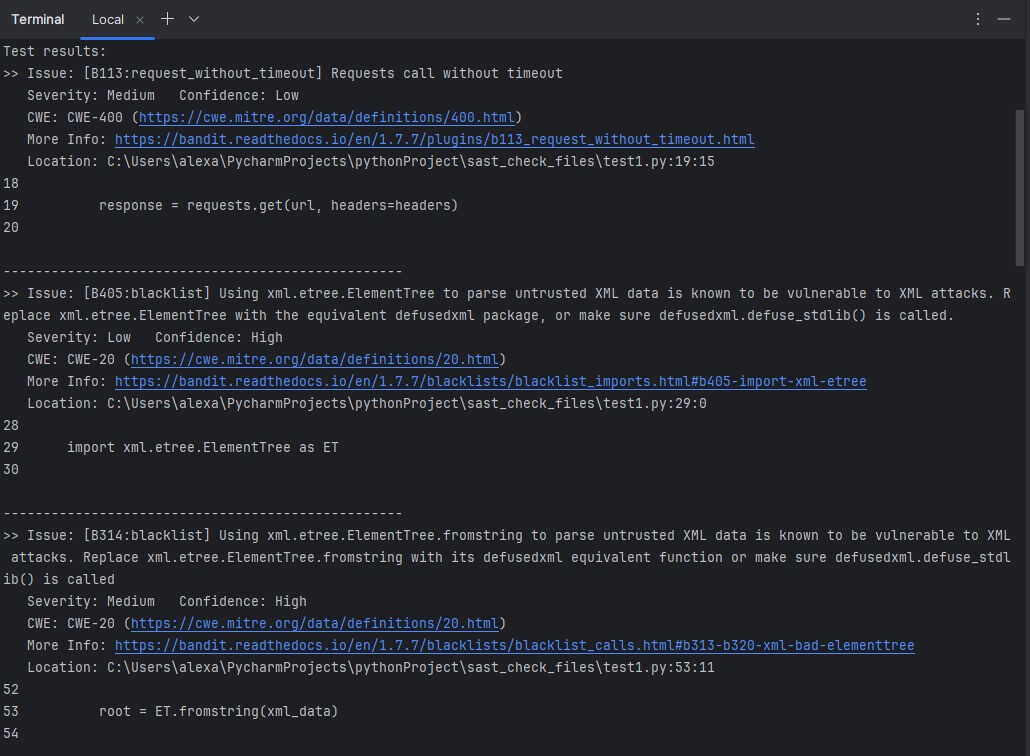
ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ-СКАНЕРА SAST

Для детального изучения каждого этапа тестирования проведем запуск разработанного приложения в терминале. Запускаем файл main.py.



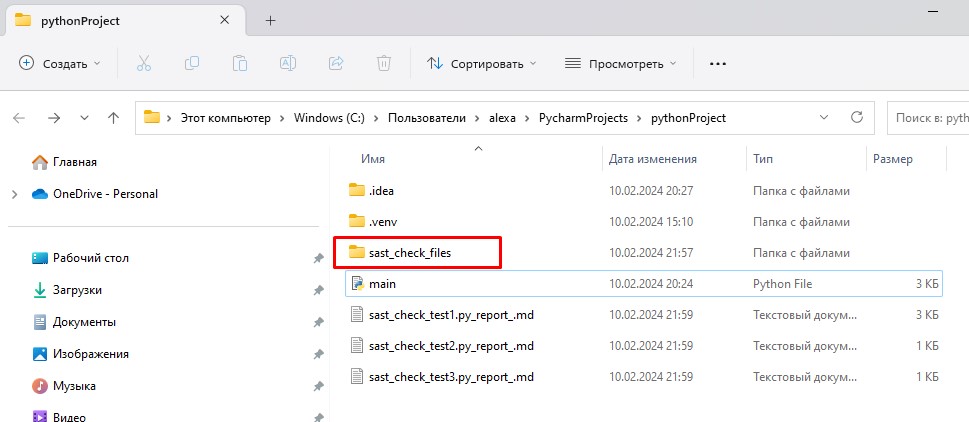
**Рисунок 4. Запуск файла main.py.**

После запуска отображаются параметры и профили сканирования. Ниже в терминале появляются первые результаты анализа.



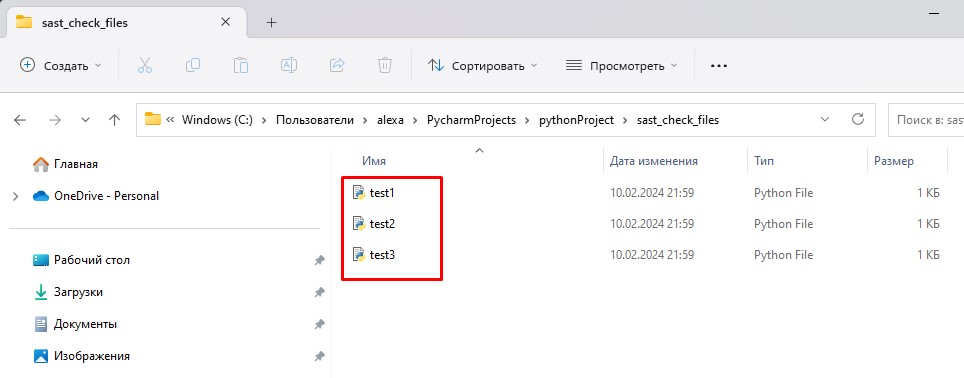
**Рисунок 5. Результаты тестирования в консоли.**

В папке проекта программы была создана директория sast\_check\_files.



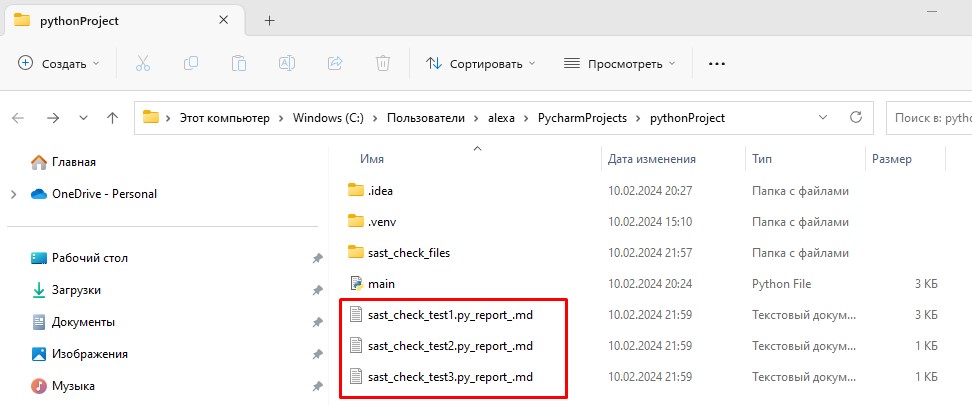
**Рисунок 6. Директория sast\_check\_files.**

В директорию sast\_check\_files были скопированы файлы для анализа из нашего репозитория.



**Рисунок 7. Файлы для анализа скопированы на локальную машину.**

После сканирования в папке проекта были созданы 3 файла с отчетами о проведенном анализе на уязвимости.



**Рисунок 8. Отчеты о проведенном анализе на уязвимости.**

По итогу в каждом из трех анализируемых файлов: test1.py, test2.py, test3.py, наше приложение-сканер SAST нашло уязвимости в программном коде.

Рассмотрим отчет по каждому файлу отдельно.

Отчет по test1.py. В программном коде test1.py найдено сразу две уязвимости:

**1. B113: Requests call without timeout**

Классификация: CWE-400: Неконтролируемое потребление ресурсов

Уровень угрозы: Средний

Описание: данная уязвимость возникает, когда при использовании библиотеки Requests для сетевых запросов не устанавливается таймаут. Таймаут — это время ожидания ответа от сервера. Если таймаут не установлен, запрос может замедлиться или даже зависнуть, если сервер не отвечает или отвечает очень медленно. Это может привести к тому, что приложение будет заблокировано и не сможет продолжать свою работу до получения ответа или истечения таймаута.Установка таймаута при использовании Requests позволяет контролировать время ожидания ответа от сервера. Вы можете указать, сколько секунд вы желаете ожидать ответа, и если ответ не поступает в течение указанного времени, будет сгенерировано исключение.

Сегмент кода, содержащий уязвимость:

*response = requests.get(url, headers=headers)*

Решение: необходимо установить таймаут, например на 5 секунд. Если ответ от сервера не поступает в течение этого времени, будет сгенерировано исключение *requests.Timeout*, которое можно обработать соответствующим образом.

**2. B405: blacklist**

Классификация: CWE-20: неправильная проверка ввода

Уровень угрозы: Низкая

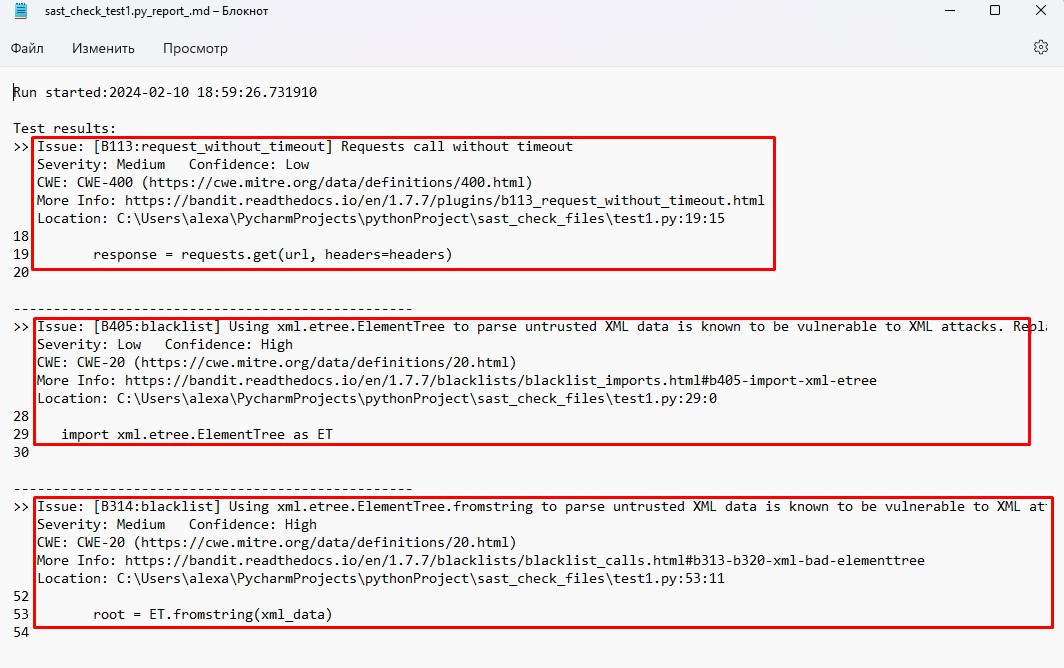
Описание: уязвимость XML Entity Expansion (XXE) возникает, когда входные данные XML обрабатываются небезопасным способом, что может привести к атакам, таким как чтение файлов с сервера или выполнение произвольного кода.

Сегмент кода, содержащий уязвимость:

*import xml.etree.ElementTree as ET*

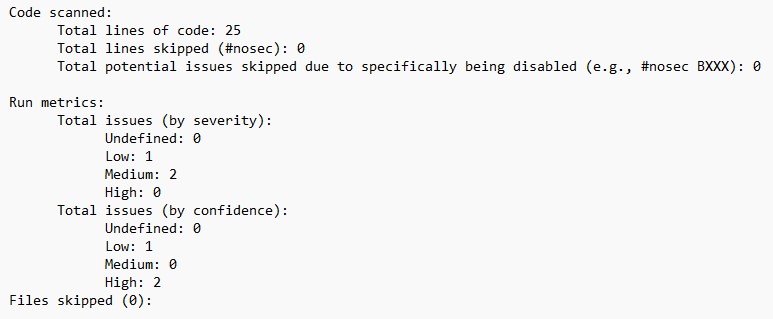
*root = ET.fromstring(xml\_data)*

Решение: чтобы исправить эту уязвимость, рекомендуется использовать безопасный парсер XML, который отключает обработку внешних сущностей. Например, *defusedxml*.



**Рисунок 9. Отчет по test1.py (1).**

В отчете под описанными уязвимостями отображается итоговые результаты сканирования:



**Рисунок 10. Отчет по test1.py (2).**

Отчет по test2.py. В программном коде test2.py найдена одна уязвимость:

**B608: hardcoded\_sql\_expressions**

Классификация: CWE-89: Неправильная нейтрализация специальных элементов, используемых в команде SQL («SQL-инъекция»)

Уровень угрозы: Средний

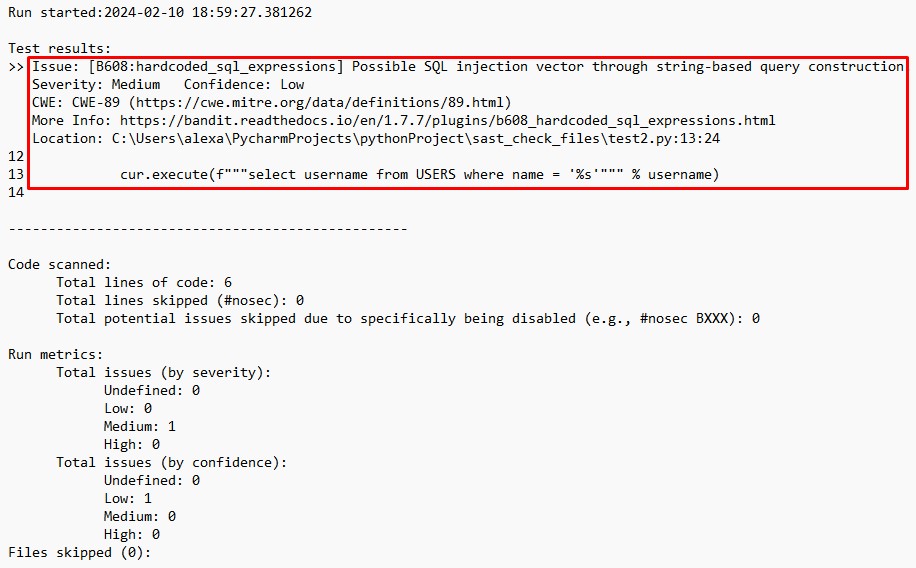
Описание: данная уязвимость возникает, когда в программном коде не выполняется должная проверка и очистка ввода пользователя, который затем используется в SQL-запросах без должного экранирования. Это позволяет злоумышленникам внедрять вредоносный SQL-код в запросы и получать несанкционированный доступ к базе данных или изменять ее содержимое.

SQL-инъекции могут возникать в различных типах SQL-запросов, таких как SELECT, INSERT, UPDATE или DELETE. Злоумышленник может использовать различные техники, чтобы внедрить вредоносный код, включая комментарии, UNION операторы, использование специальных символов и другие методы.

Сегмент кода, содержащий уязвимость:

*cur.execute(f"""select username from USERS where name = '%s'""" % username)*

Решение: для защиты от SQL-инъекций необходимо использовать параметризованные запросы или подготавливать запросы с использованием специальных методов в соответствующей библиотеке для работы с базой данных. Это позволяет правильно экранировать ввод пользователя и предотвращает возможность внедрения вредоносного SQL-кода.



**Рисунок 11. Отчет по test2.py.**

Отчет по test3.py. В программном коде test3.py найдена одна уязвимость:

**B113: Requests call without timeout**

Классификация: CWE-400: Неконтролируемое потребление ресурсов

Уровень угрозы: Средний

Описание: данная уязвимость возникает, когда при использовании библиотеки Requests для сетевых запросов не устанавливается таймаут.

Таймаут — это время ожидания ответа от сервера. Если таймаут не установлен, запрос может замедлиться или даже зависнуть, если сервер не отвечает или отвечает очень медленно. Это может привести к тому, что приложение будет заблокировано и не сможет продолжать свою работу до получения ответа или истечения таймаута.

Установка таймаута при использовании Requests позволяет контролировать время ожидания ответа от сервера. Вы можете указать, сколько секунд вы желаете ожидать ответа, и если ответ не поступает в течение указанного времени, будет сгенерировано исключение.

Сегмент кода, содержащий уязвимость:

*def get\_internal\_data():*

*response = requests.get(internal\_url)*

*return response.json()*

Решение: необходимо установить таймаут, например на 5 секунд. Если ответ от сервера не поступает в течение этого времени, будет сгенерировано исключение *requests.Timeout*, которое можно обработать соответствующим образом.

**

**Рисунок 12. Отчет по test3.py.**

ЗАКЛЮЧЕНИЕ

В данной дипломной работе было разработано приложение-сканер SAST на языке Python с использованием модуля Bandit для статического анализа уязвимостей в программном коде файлов с репозитория на GitHub. Целью работы было обнаружение потенциальных уязвимостей и повышение уровня безопасности программного кода.

Приложение прошло успешное тестирование и продемонстрировало эффективность в обнаружении уязвимостей в переданных ему файлах. С помощью алгоритмов и правил, встроенных в модуль Bandit, были выявлены различные типы уязвимостей, такие как возможные SQL-инъекции, использование небезопасных функций и др.

Статический анализ кода с использованием модуля Bandit значительно повысил безопасность разрабатываемого программного кода. Обнаружение уязвимостей на ранней стадии разработки позволяет предотвратить потенциальные атаки и эксплойты, а также улучшить общую безопасность создаваемого ПО.

Разработанное приложение-сканер SAST предоставляет разработчикам и команде безопасности возможность автоматического анализа программного кода файлов с репозитория на GitHub и обнаружения потенциальных уязвимостей. Интеграция данного приложения в процесс CI/CD позволяет автоматизировать анализ кода и обеспечить непрерывную безопасность при разработке и развертывании программного продукта.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Златопольский Д.М. Основы программирования на языке Python. – М.: ДМК Пресс, 2017. – 284 с.

2. Лутц М. Программирование на Python, том I, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.

3. Лутц М. Программирование на Python, том II, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.

4. Свейгарт, Эл. Автоматизация рутиных задач с помощью Python: практическое руководство для начинающих. Пер. с англ. — М.: Вильямc, 2016. – 592 с.

5. Федоров, Д. Ю. Программирование на языке высокого уровня Python : учебное пособие для прикладного бакалавриата / Д. Ю. Федоров. – 2-е изд., перераб. и доп. – Москва : Издательство Юрайт, 2019. – 161 с.

6. Лебедев А.В., Декалин А.В. "Статический анализ кода программного обеспечения: методы, алгоритмы, программные средства". Москва, 2015.

7. Федоров Ю.В., Шамшин А.В., "Статический анализ кода на основе метрик программной инженерии". Информационные технологии и вычислительные системы, 2018.

8. Chess B., West J. Secure Programming with Static Analysis – 2007. – 618 с.

9. Обнаружение уязвимостей в теории и на практике, или почему не существует идеального статического анализатора [Электронный ресурс]. - 2018. - Режим доступа: https://habr.com/company/solarsecurity/blog/ 420337/- Дата доступа: 06.12.2018

10. Exploit (computer security) [Электронный ресурс]. - 2019. - Режим доступа: https://en.wikipedia.org/wiki/Exploit\_(computer\_security) - Дата доступа: 16.03.2019

11. How Does Static Analysis Prevent Defects and Accelerate Delivery? [Электронный ресурс]. - 2018. - Режим доступа: https://dzone.com/articles/ how-does-static-analysis-prevent-defects-and-accel- Дата доступа: 19.03.2019

12. MISRA C:2012 Amendment 1 [Электронный ресурс]. - 2016. - Режим доступа: https://www.misra.org.uk/LinkClick.aspx? fileticket=V2wsZxtVGkE%3D&tabid=57- Дата доступа: 10.04.2019

### Приложение 1 *Отзыва руководителя на ВКР*

**ОТЗЫВ РУКОВОДИТЕЛЯ**

на выпускную квалификационную работу бакалавра

Студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Курса \_\_\_\_\_\_\_\_\_\_\_\_\_\_Группы\_\_\_\_\_

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О., ученая степень и (или) ученое звание)

Тема \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*текст отзыва с указанием критериев оценивания*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Выпускная квалификационная работа студента заслуживает оценки \_\_\_ баллов по 10‑балльной шкале.

Необходимо привести рекомендации о допуске выпускной работы к защите, отзыв о работе выпускника в период выполнения ВКР, оценку за работу по десятибалльной шкале и заключение о возможности присвоения студенту квалификации бакалавр по направлению подготовки «Информационная безопасность».

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О. руководителя)

Дата:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Приложение 2 *Рецензии на ВКР*

**РЕЦЕНЗИЯ**

на выпускную квалификационную работу бакалавра

Студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Курса \_\_\_\_\_\_\_\_\_\_ Группы\_\_\_\_\_

Рецензент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О., ученая степень и (или) ученое звание (при наличии), должность)

Тема \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Пожалуйста, охарактеризуйте:

Соответствие работы выбранной теме

Самостоятельность работы

Достигнутые результаты, практическое значение, недостатки работы.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Выпускная квалификационная работа студента заслуживает оценки \_\_\_ баллов по 10‑балльной шкале.

Необходимо привести оценку за работу по десятибалльной шкале и заключение о возможности присвоения студенту квалификации бакалавр по направлению подготовки «Информационная безопасность».

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О. рецензента)

Дата:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_