**Учебный центр**

**OOO «Скилфэктори»**

совместно с

**Федеральное государственное автономное**

**образовательное учреждение высшего образования**

**Национальный исследовательский ядерный университет «МИФИ».**

Отчёт к выпускной квалификационной работе

на тему:

**“Метод и система статического анализа для поиска уязвимостей в программном обеспечении”**

Программа дополнительного профессионального образования

“Информационная безопасность”

**Выполнили студенты MIFIIB**

**2 поток:**

Трач Александр

Миронов Никита

Зимодра Юлия

**Научный руководитель:**

Серов Сергей

Москва, 2024

Оглавление

[Введение 2](#_Toc163128056)

[Существующие подходы защиты и идентификации дефектов (уязвимостей и ошибок) ПО 5](#_Toc163128057)

[Наиболее распространенные дефекты (уязвимости и ошибки) ПО 6](#_Toc163128058)

[Технологии специализации систем автоматического поиска дефектов в исходном коде программ 7](#_Toc163128059)

[Ключевые функции, на которые следует обращать внимание в инструментах SAST 8](#_Toc163128060)

[Предложения на рынке 9](#_Toc163128061)

[Практическая часть 13](#_Toc163128062)

[Заключение 22](#_Toc163128063)

[Приложение 24](#_Toc163128064)

# Введение

В современных реалиях ПО встраивается в процессы автоматизации практически во всех сферах, количество строк кода приложений увеличивается. Как следствие, растёт число возможных уязвимостей и ошибок, что формирует потребность в эффективной проверке и тестировании исходного кода.

По данным исследования Positive Technologies в 2019 году, 82% всех зафиксированных уязвимостей вызваны ошибками, допущенными при написании исходного кода. По мнению экспертов, в среднем, в каждой второй системе есть дефект безопасности с высоким уровнем риска. Один из основных источников этой проблемы – отсутствие проверок исходного кода на наличие уязвимостей во время его написания. Также один из косвенных факторов – нежелание разработчиков уделять внимание безопасности, они чаще сосредотачиваются на функциональности приложения.

Тем не менее, рост числа уязвимостей и риск их использования злоумышленниками форсирует развитие рынка безопасности приложений. Это особенно заметно из результатов исследования, проведённого Grand View Research:

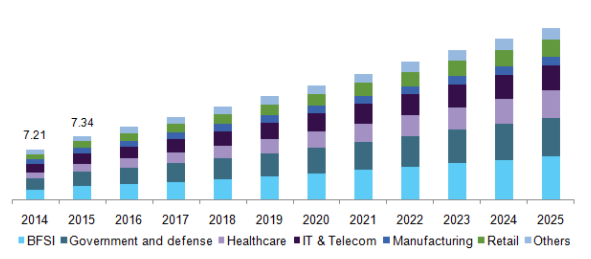


Рисунок 1. Рынок безопасности приложений США по конечному использованию (ссылка недоступна для просмотра из России), 2014–2025 гг. (млн долл. США).

Есть мнение, что не все уязвимости потенциально опасны. Некоторые из них могут быть в исходном коде годами и так ни к чему и не привести. Для выставления приоритета исправления уязвимости разработчики ориентируются на стандарт CVSS (Common Vulnerability Scoring System), но:

* она не универсальна для различных сфер;
* часть экспертов в кибербезопасности считают эту систему устаревшей (ссылка недоступна для просмотра из России) и, как следствие, опасной.

Да, можно уверенно сказать, что качество тестирования кода с годами растёт благодаря многообразию инструментов, их развитию. Старые уязвимости закрываются, но, к сожалению, появляются новые. Кроме того, развиваются инструменты и методы злоумышленников для обнаружения неправильного кода, недостатками которого можно воспользоваться.

В ответ на постоянно растущую угрозу и увеличение кодовой базы разработчики используют инструменты **AST (Application Security Testing)**. AST – это процесс повышения безопасности приложений путём выявления уязвимостей в исходном коде. Изначально такое тестирование проводилось вручную. Однако рост количества строк кода и случаев применения стороннего открытого кода, который тоже нужно проверять, привели к необходимости автоматизации процесса.

Использование инструментов тестирования является важной частью концепции [DevSecOps](https://pvs-studio.com/ru/blog/posts/0710/). А теперь немного подробнее. В отчёте "[Application Security Testing Market — Global Industry Analysis, Size, Share, Growth, Trends, and Forecast 2017–2025](https://www.transparencymarketresearch.com/application-security-testing-market.html)" от Transparency Market Research рынок тестирования безопасности приложений сегментирован на следующие классы продуктов:

* Static Application Security Testing (SAST),
* Dynamic Application Security Testing (DAST),
* Interactive Application Security Testing (IAST),
* Mobile Application Security Testing

# Существующие подходы защиты и идентификации дефектов (уязвимостей и ошибок) ПО

Технологии поиска дефектов (уязвимостей и ошибок) и защиты от них разрабатываются по трем основным направлениям. Во-первых, используются системы автоматического поиска дефектов с помощью статического анализа исходного кода программ, которые можно применять на самых ранних этапах разработки, что делает исправление дефектов максимально дешевым. Такие системы промышленного качества обрабатывают программные системы в миллионы строк кода, обладают приемлемым уровнем истинных срабатываний (30-70% найденных ошибок оказываются истинными) и анализируют все возможные варианты выполнения программы одновременно. Возможно применять более точные методы анализа, методы верификации программ для самых важных участков программы за счет уменьшения объема анализируемого кода. Во-вторых, применяются системы динамического анализа бинарного кода программ, позволяющие многократно запускать заданную программу на автоматически генерируемом наборе входных данных и отслеживать ситуации возникновения дефектов. Системы динамического анализа просматривают лишь часть возможных наборов входных данных, но при нахождении ошибки сразу позволяют получить данные, на которых эта ошибка проявляется (т.е. не имеют ложных срабатываний). Применимость этих систем ограничена существенными требованиями к ресурсам и ограничением на максимальный размер анализируемой программы (обычно десятки тысяч строк кода). Наконец, разрабатываются технологии защиты программы и ее окружения от эксплуатации имеющихся в ней уязвимостей во время работы программы. Это направление является актуальным из-за того, что все системы анализа не гарантируют полного нахождения имеющихся в ПО уязвимостей. Используемые методы данного подхода позволяют существенно затруднить атакующему успешную реализацию своих целей, исключить возможность эксплуатации даже известной уязвимости и сформировать универсальный метод взлома конкретной программы.

# Наиболее распространенные дефекты (уязвимости и ошибки) ПО

Согласно приведенному определению уязвимости, будем отличать понятия ошибки программы, наличие которых не приводят к взлому программы, от понятия уязвимость, которые предоставляют возможность взлома программной системы. Исходя из этого, дефекты ПО можно разделить на уязвимости (критические ошибки), приводящие к нарушению работоспособности, отказу в обслуживании, изменению защищенности информационных ресурсов, и некритические, влияющие лишь на качество программной системы (например, программа использует больше памяти для работы, чем необходимо, из-за утечек памяти при работе с ней). Наибольший интерес представляет поиск уязвимостей (критических ошибок), однако и некритические ошибки могут привести к уязвимостям, поэтому современные системы анализа обычно направлены на идентификацию обоих типов дефектов. Актуальная общепринятая классификация дефектов по типам представлена в базе Common Weakness Enumeration, список зарегистрированных уязвимостей – в базе Common Vulnerabilities and Exposures организации MITRE. Наиболее распространенными типами дефектов являются:

переполнение буфера;

ошибки при работе с динамической памятью (утечка памяти, разыменование нулевых указателей и др.);

ошибки обработки пользовательских данных;

ошибки форматных строк; ошибки синхронизации (взаимные блокировки, отсутствующие блокировки и т.п.);

утечки памяти и других ресурсов системы; некорректная работа с временными файлами и другими интерфейсами ОС;

уязвимости безопасности (слабое шифрование, хранение пароля в явном виде и т.п.), не вытекающие непосредственно из дефектов, можно выделить в отдельный класс уязвимостей.

# Технологии специализации систем автоматического поиска дефектов в исходном коде программ

Необходимо отметить, что эффективность систем автоматического поиска дефектов ПО зависит от того, на каком наборе тестовых программ нарабатывались эвристики этих систем. По умолчанию эвристики поиска ситуаций в исходном коде, ранжирования собранных данных по важности, параметры точности применяемых алгоритмов настроены на некоторое «среднее» значение поведения анализируемых программ. Так, если программист перед использованием некоторого указателя проверяет его значение на корректность во всех точках программы, кроме одной-двух, то велика вероятность, что и в этих точках программы значение указателя может быть некорректным, и требуется выдать предупреждение о возможном разыменовании нулевого указателя. Конечно, такая эвристика применяется лишь в том случае, если основные алгоритмы статического анализа не смогли с достаточной точностью установить значение указателя. Следовательно, для специальных классов программ является возможной такая доработка системы автоматического поиска, что точность выполняемого системой анализа для этих классов программ повысится. При этом основные виды анализов, выполняемые системой, останутся неизменными. Перечислим основные способы такой доработки систем автоматического поиска:

определение специфичных для данного класса программ ситуаций, сигнализирующих о дефектах в исходном коде, доработка эвристик системы для выдачи предупреждений о таких ситуациях;

настройка параметров анализа системы или применение дополнительных высокоточных методов анализа (например, чувствительных к путям выполнения) для ключевых, ограниченных по объему участков программы;

ранжирование вычисленной в ходе анализа информации по важности для конкретного класса программы (например, более точное вычисление информации о параметрах вызовов интерфейсов межпроцессного взаимодействия ОС для параллельных программ);

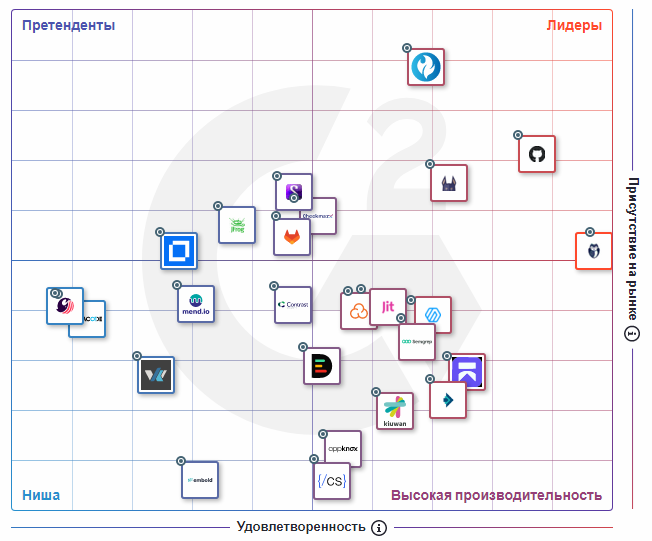
создание спецификаций для используемых программой библиотек, исходный код которых недоступен (как правило, система анализа знает о поведении лишь стандартных библиотек POSIX, C, C++).

Важным специальным классом программы являются системные программы, в частности, ядро и драйверы операционной системы. Как показано выше, объектами воздействия при эксплуатации уязвимостей являются чаще всего именно системные ресурсы, ошибки в привилегированных программах ОС сразу при эксплуатации позволяют добиться необходимой эскалации прав, а также дальнейшего нарушения защищенности информации. Например, при анализе ядра Linux можно создать спецификации для функций выделения и освобождения памяти, используемых в ядре, а также учесть соглашения ядра о возврате кодов ошибок из интерфейсных функций. Для встраиваемых систем, ОС реального времени можно воспользоваться относительно небольшим объемом анализируемого исходного кода и повысить точность алгоритмов анализа. Разработка и реализация таких методов специализации системы автоматического поиска дефектов в настоящий момент является предметом исследований.

# Ключевые функции, на которые следует обращать внимание в инструментах SAST

1. **Поддержка языков программирования.** Убедитесь, что ваш инструмент SAST поддерживает все языки программирования, с которыми работает ваша команда.
2. **Интеграция с конвейерами CI/CD и IDE.** Инструменты SAST должны легко интегрироваться в ваши конвейеры CI/CD, чтобы обеспечить автоматизацию сканирования на разных этапах разработки. Они также должны интегрироваться с IDE для получения обратной связи в режиме реального времени во время создания кода.
3. **Наборы правил и классификация уязвимостей.** Инструмент должен иметь обширную библиотеку правил анализа кода, охватывающую общие проблемы безопасности и лучшие практики, а также иметь возможность интеграции с общедоступными базами данных уязвимостей, такими как OSWAP и CVE.
4. **Руководство по устранению:** комплексный инструмент SAST должен предлагать рекомендации по устранению в режиме реального времени, чтобы вы могли принять меры, как только будет обнаружена любая новая уязвимость.
5. **Процент ложных срабатываний.** Ложные срабатывания истощают ресурсы и, проще говоря, невероятно раздражают. Выберите инструмент, который может свести к минимуму ложные срабатывания и уменьшить шум.

# Предложения на рынке



CodeQL

Инструмент презентовали в компании Semmle в 2018 году. Сейчас он входит в линейку продуктов Microsoft. Одно из преимуществ – постоянно пополняемый список уязвимостей, которые CodeQL обнаруживает в коде. Еще плюс – популярность продукта, благодаря чему в Сети можно найти множество инструкций и обсуждений.

Weggli

Исследует код на C/C++. Чтобы провести статический анализ с его помощью, собирать проект необязательно. Из преимуществ – высокая скорость поиска. Но есть и минус – информации и кейсов использования Weggli для задач ИБ в интернете пока мало.

Pylint

Инструмент для статического анализа кода на Python. Его актуальность связана с популярностью языка программирования. Pylint проверяет код на соответствие стандартам PEP 8 (Python Enhancement Proposal 8) и другим рекомендациям, которые касаются разработки на Python.

ESLint

Анализирует код на JavaScript. Обнаруживает и даже помогает исправлять синтаксические ошибки, неиспользуемые переменные, неправильное использование функций и другие нарушения стиля и безопасности разработки.

SonarLint

Поддерживает статический анализ кода на различных языках программирования, что делает его универсальным решением для проектов с различными технологиями. Обеспечивает всестороннее исследование, которое включает обнаружение ошибок, дублированного кода, нарушений кодового стиля, уязвимостей безопасности и других проблем.

DeepCode

Находит ошибки, уязвимости безопасности, проблемы с производительностью и API на основе ИИ. Поставляется бесплатно для открытых исходных кодов и частных репозиториев, а также для небольших команд – до 30 разработчиков.

LGTM

Предназначен для статического анализа безопасности кода на языках Java, Python, JavaScript, TypeScript, C #, C и C ++. Находит потенциальные уязвимости, такие как использование небезопасных функций, некорректная обработка ввода пользователя и др.

[Checkmarx](https://translated.turbopages.org/proxy_u/en-ru.ru.1880ccea-660db9b4-a2069a26-74722d776562/https/checkmarx.com/?__ya_mt_enable_static_translations=1)

Поставляется с предварительной настройкой для поддержки широкого спектра языков. Он выявляет недостатки безопасности и предлагает исправления. С помощью Checkmarx SAST можно выполнять быстрое прогрессивное или полное сканирование. Хотя интерфейс немного устарел по сравнению с более свежими опциями, он заслуживает доверия и работает именно так, как указано на упаковке.

[Klocwork](https://translated.turbopages.org/proxy_u/en-ru.ru.1880ccea-660db9b4-a2069a26-74722d776562/https/www.perforce.com/products/klocwork?__ya_mt_enable_static_translations=1)

Предназначен для адаптации к организации любого размера и работает с несколькими кодовыми базами. Возможности статического анализа Klocwork работают в тандеме с вашими code twines и другими средствами обнаружения проблем IDE на ходу. Он особенно хорош для обнаружения ошибок нулевого указателя, выхода массива за пределы и подобных проблем без необходимости выполнения кода.

[SpectralOps](https://translated.turbopages.org/proxy_u/en-ru.ru.1880ccea-660db9b4-a2069a26-74722d776562/https/spectralops.io/?__ya_mt_enable_static_translations=1)

Выделяется на рынке тем, что он непрерывно анализирует весь SDLC на предмет предопределенных кодов, ключей и несоответствующего кода. Основная цель Spectral - сохранить секреты (такие как пароли, API и ключи шифрования) в безопасности.Секреты часто жестко кодируются на ранних этапах разработки функций, а затем скрываются в коде, что делает их уязвимыми для потенциальных злоумышленников. Это не ограничивается кодированием; другие форматы файлов также могут быть источником утечек.

[HCL Appscan](https://translated.turbopages.org/proxy_u/en-ru.ru.1880ccea-660db9b4-a2069a26-74722d776562/https/www.hcltechsw.com/appscan?__ya_mt_enable_static_translations=1)

Это SAST для онлайн-приложений, которые когда-то принадлежали IBM. Он отличается от других более экономичных вариантов меньшим количеством ложноположительных результатов, поддерживаемых компьютерным обучением. AppScan создает надежные сценарии тестирования для ваших веб-приложений, чтобы обеспечить плавный переход к рабочей среде, а также устранить известные недостатки безопасности.

Semgrep

Это инструмент статического анализа с открытым исходным кодом, который используется для поиска ошибок, уязвимостей безопасности и других проблем в исходном коде. Semgrep анализирует код локально на вашем компьютере или в вашей среде сборки: код никуда не загружается.

Анализ может быть выполнен без необходимости пользовательской настройки и с использованием наборов правил, созданных Semgrep Inc. и участниками с открытым исходным кодом. Инструмент также позволяет пользователям писать свои собственные шаблоны и правила через интерфейс командной строки, используя язык шаблонов, уникальный для semgrep. Также доступны [бесплатный онлайн-редактор правил](https://semgrep.dev/playground/new) и [учебник](https://semgrep.dev/learn).

Существующие правила Semgrep: это правила по умолчанию, включенные в [реестр правил Semgrep](https://semgrep.dev/explore). Команда разработчиков Semgrep создает и поддерживает эти правила, охватывающие различные потенциальные уязвимости безопасности и ошибки кодирования. Разработчики могут использовать эти правила как есть или настроить их так, чтобы они лучше соответствовали их конкретной кодовой базе.

Что можно обнаружить:

* Уязвимости безопасности - уязвимости, такие как инъекции SQL, XSS, небезопасные вызовы функций, использование устаревших алгоритмов шифрования и т.д.
* Ошибки в коде - например, можно обнаружить неправильное использование операторов, дублирование кода, проблемы с памятью. А также ошибки стиля: неправильное форматирование кода, использование лишних пробелов и табуляции, неправильное использование комментариев.
* Проблемы производительности, такие как медленные запросы к базе данных, ненужные вычисления, длинные циклы.

Semgrep может сканировать код на различных языках программирования, включая Python, Java, JavaScript, Go, Ruby, PHP, C++, C#.

# Практическая часть

Навыки статического анализа кода в арсенале исследователя безопасности приложений фактически являются must-have скиллом. Искать ручками уязвимости в коде, не прибегая к автоматизации, для небольших проектов вполне быть может и приемлемый сценарий. Но для больших задач с миллионами строк кода — это непозволительная роскошь с точки зрения временных затрат. Решения по автоматизации всего этого процесса не заставили себя долго ждать. Именно по этой причине мы имеем такой широкий выбор инструментов статического анализа, в разнообразии которого можно утонуть.

К сожалению, о SAST далеко не все отзываются лестно. Так уж вышло, что общее отношение к SAST у ряда специалистов скорее скептично-негативное ввиду большого количества ложных срабатываний. Это отбивает желание пользоваться подобными инструментами, тратя на них свое драгоценное время.

Нами был выбран в качестве инструмента статического анализа Semgrep.

[Semgrep](https://github.com/returntocorp/semgrep) — легковесный инструмент статического анализа для поиска ошибок в коде. Был разработан командой [r2c](https://r2c.dev/#semgrep), написан в большинстве своем на OCaml. Первый релиз состоялся в начале 2020 года. Пожалуй, это один из немногих инструментов, в котором трансформация "From Zero To Hero" для пользователя является самой скоростной. Всё это благодаря тому, что язык запросов, или в данном случае паттернов, максимально похож на язык программирования исследуемого проекта.

Когда Semgrep выполняет анализ кода, он создает абстрактное синтаксическое дерево (AST), которое затем транслируется в промежуточный язык, с которым уже и проводится анализ. Это крайне удобно и позволяет за короткое время научиться создавать эффективные правила. На этот счет инструмент имеет отличную [документацию](https://semgrep.dev/docs/), а также большое количество хвалебных отзывов от AppSec-комьюнити, в том числе в виде отдельных статей. Мы тоже не обойдем стороной его **достоинства** — вот они:

Не требует сборки проекта, а значит можно предоставлять несобираемые исходники, результат декомпиляции

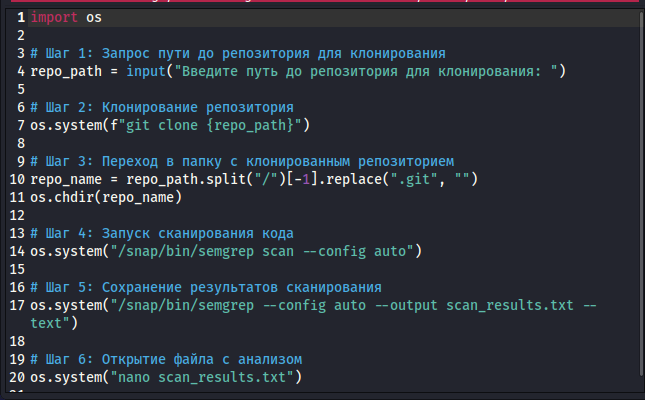
* Поддерживаемые языки: C#, Go, Java, JavaScript, JSX, JSON, PHP, Python, Ruby, Scala, TypeScript, TSX. C/C++ имеют статус экспериментальных языков, как и [многие другие](https://semgrep.dev/docs/supported-languages/)
* Синтаксис паттернов совпадает с синтаксисом языка проекта
* Быстрая обработка запросов благодаря многопоточности
* Имеет крутую [playground-площадку](https://semgrep.dev/playground/new) для обучения синтаксису
* В [реестре](https://semgrep.dev/r) semgrep-правил можно подсмотреть security rules для множества языков, создавать свои и делиться ими
* Есть собственный [TrophyСase](https://semgrep.dev/docs/trophy-case/) — найденные CVE с помощью Semgrep

И несколько **недоработок**:

* Не знает про межпроцедурное взаимодействие, анализирует все методы подряд без разбора что, где и когда вызывается
* Не так много публичных правил для C/C++

Далее пишем скрипт для создания дополнительного пайплайна для проверки кода и тестирования своих правил. Целью данной работы является показать универсальность инструментов SAST и внесение вкалада в развитие безопасности. Так же Semgrep позволяет сохранять результаты сканирования в различных форматах, что позволит их использовать в дальнейшем и если понадобиться загружать в другие пайплайны.

Создаем скрипт, который будет загружать репозитории из Github (Gitlab) и запускать их сканирование с последующим выведением и сохранением результатов.

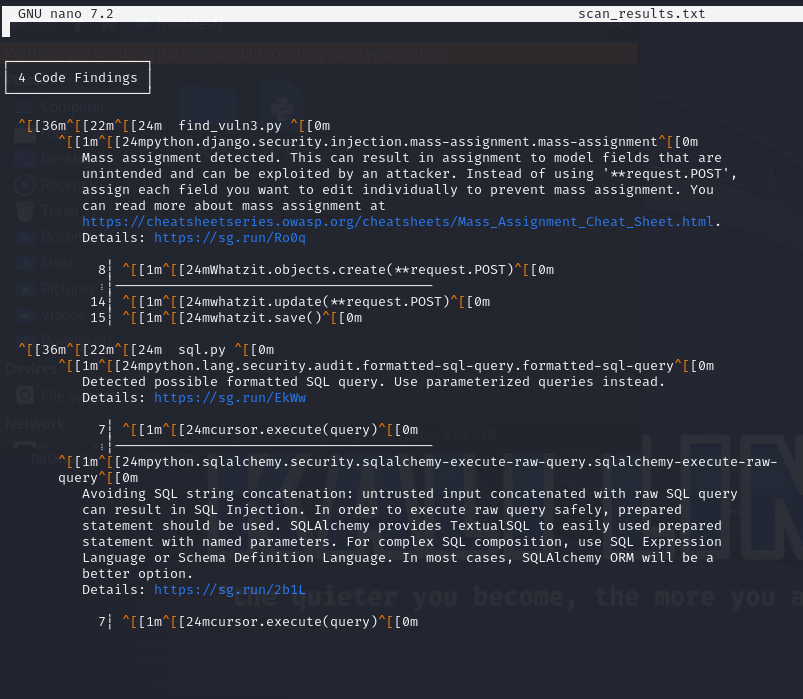


Запускаем скрипт «vkr.py» и указываем репозиторий для клонирования. После завершения клонирования автоматически запускается сканирование

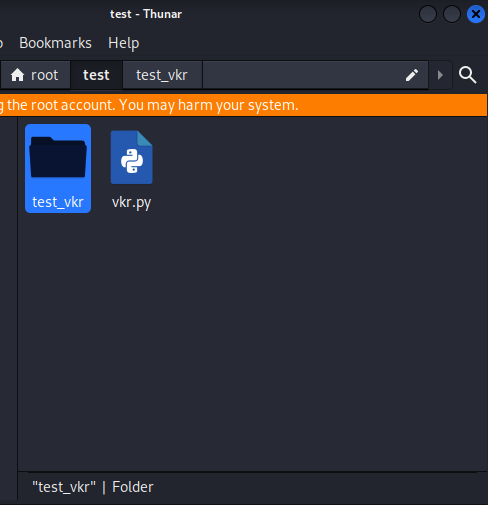


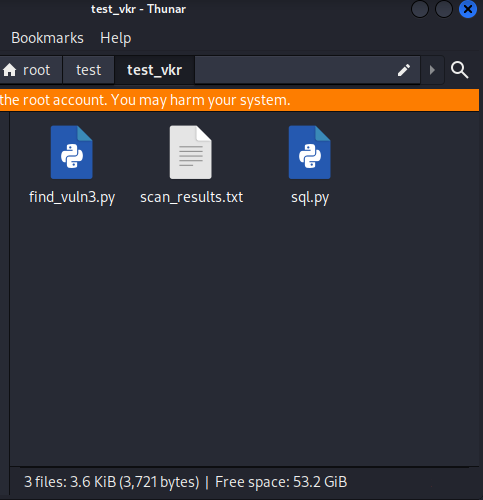


Далее скрипт выводит нам результаты скнирования



Клонированный репозиторий и сохраненные результаты сканирования





# Заключение

Нами была проделана работа на основе анализатора Semgrep. Он полностью open-source, легковесный и может анализировать код на различных языках программирования. Было обнаружено 3 уязвимости:

1. Mass assignment (массовое присвоение), которое может быть проэксплуатировано атакующим.
2. Possible formatted SQL query (возможный форматированный SQL запрос)
3. Avoiding SQL string concatenation (избегание конкатенации строк) (Конкатенация строк в Python — это операция соединения двух или более строк в одну)

К каждому пункту прилагаются советы по исправлению и ссылки для более глубокого изучения вопроса. Также указаны номера строк с уязвимостями.

Выбранный нами SAST отвечает большинству требований: Semgrep имеет широкое покрытие, он гибкий, имеет возможность автоматизации, корректно обрабатывает изменения и дополнения, довольно точный, хорошо интегрируется со средой IDE и, главное, абсолютно доступный по цене.

С теоретической точки зрения исследования вносят вклад в растущий объем знаний в области кибербезопасности, предоставляя эмпирические данные, которые подкрепляют существующие теории о цифровой уязвимости и защитных механизмах. На практике полученные данные предлагают план повышения безопасности, призывая к внедрению комплексных методов безопасности и внедрению лучших практик в управлении кибербезопасностью.

Подчеркивается важность проведения оценок уязвимости в рамках правовых и этических стандартов. С методологической точки зрения исследования подчеркивают эффективность автоматизированных инструментов в выявлении уязвимостей безопасности.

Будущие исследования должны изучить развитие автоматизированных систем обнаружения уязвимостей, интегрирующих алгоритмы искусственного интеллекта и машинного обучения для повышения прогностических возможностей и эффективности оценок безопасности. Развитие этих технологий открывает новые горизонты в области кибербезопасности.

Существует убедительный аргумент в пользу принятия междисциплинарных подходов в изучении кибербезопасности. Объединение идей психологии, социологии и права может обеспечить более целостное понимание проблем кибербезопасности, затрагивая не только технические аспекты, но и человеческий фактор, который играет решающую роль в экосистеме безопасности.

Динамичный характер киберугроз требует продольных исследований, которые отслеживают эволюцию методов обеспечения безопасности и уязвимостей с течением времени. Такие исследования могут дать ценную информацию об эффективности мер безопасности, адаптивности злоумышленников.

Учитывая глобальный характер киберугроз, дальнейшие исследования также должны быть сосредоточены на сравнительном анализе тенденций кибербезопасности в различных регионах и секторах. Этот подход может раскрыть уникальные проблемы и лучшие практики, способствуя более полному пониманию глобальной динамики кибербезопасности.

# Приложение

Исходный код сканируемых файлов

