ОТЧЕТ

О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

«Метод поиска и устранения уязвимостей в открытом программном обеспечении для финансовой сферы»

Исполнитель: Шмелева А.С.

Москва, 2024

СОДЕРЖАНИЕ

Введение 3

1. Открытое программное обеспечение, его особенности и риски использования 4

1.1. Открытое ПО 4

1.2. Открытое ПО в финансовой сфере, его особенности и примеры 5

1.3. Уязвимости в финансовом ПО и риски их реализации 7

2. Анализ защищенности открытого программного обеспечения для финансовой сферы 9

2.1. Цикл оценки защищенности открытого программного обеспечения для финансовой сферы 9

2.2. Реализация методики для поиска и устранения уязвимостей в открытом программном обеспечении для финансовой сферы 10

3. Практический пример 12

3.1. Реализация автоматизированного поиска уязвимостей с использованием языка Python и анализаторов кода с открытым исходным кодом 12

3.2. Библиотека Bandit 12

3.3. Программное обеспечение Safety 14

3.4. Библиотека PyLint 17

3.5. Разработанное программное обеспечение 19

Заключение 27

Список использованных источников 28

ВВЕДЕНИЕ

Открытое программное обеспечение – программное обеспечение с открытым исходным кодом. Исходный код создаваемых программ открыт, то есть доступен для просмотра и изменения. Это позволяет использовать уже созданный код для создания новых версий программ и исправления ошибок.

В данной работе будут рассмотрены:

– особенности открытого программного обеспечения, его достоинства и недостатки, риски применения, приведены примеры использования;

– открытое программное обеспечения в финансовой сфере, его особенности, примеры использования;

– уязвимости открытого финансового программного и риски их реализации;

– цикл оценки защищенности открытого программного обеспечения для финансовой сферы;

– пример методики для поиска и устранения уязвимостей в открытом программном обеспечении для финансовой сферы;

– пример автоматизированного поиска уязвимостей с использованием языка Python и анализаторов кода с открытым исходным кодом.

1. ОТКРЫТОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, ЕГО ОСОБЕННОСТИ И РИСКИ ИСПОЛЬЗОВАНИЯ

1.1. Открытое ПО

Открытое программное обеспечение – программное обеспечение, распространяемое на условиях свободного лицензионного договора, дающего пользователю право использовать программу в любых, не запрещенных законом целях, получать доступ к исходным кодам программы как в целях её изучения и адаптации, так и в целях переработки, распространять программу, вносить изменения в программу и распространять экземпляры изменённой программы с учетом возможных требований наследования лицензии.

Рассмотрим основные особенности открытого программного обеспечения:

– свобода использования: пользователи имеют право использовать программное обеспечение в любых целях без ограничений;

– доступ к исходному коду: исходный код программы доступен для просмотра, изменения и улучшения.

– распространение: пользователи могут свободно распространять копии программного обеспечения.

– совместная разработка: открытое программное обеспечение часто разрабатывается сообществом разработчиков, что способствует непрерывному улучшению и обновлению.

Важно отметить, что безопасность открытого программного обеспечения зависит от активности сообщества разработчиков, реакции на уязвимости и обновлений. Однако при правильном использовании и обновлении, открытое программное обеспечение может быть безопасным и эффективным инструментом для многих организаций.

Риски использования открытого программного обеспечения могут включать в себя:

– недостаток поддержки: в некоторых случаях может быть ограничен доступ к технической поддержке и обновлениям.

– безопасность: некоторые пользователи могут беспокоиться о безопасности открытого программного обеспечения из-за возможности наличия уязвимостей или вредоносного кода.

В примеры открытого программного обеспечения можно привести операционные системы Linux, браузер Mozilla Firefox, офисный пакет LibreOffice, системами управления базами данных MySQL и PostgreSQL, а также множество других проектов.

1.2. Открытое ПО в финансовой сфере, его особенности и примеры

Открытое программное обеспечение в финансовой сфере используется для решения финансовых задач. К таким задачам можно отнести управление активами, бухгалтерию, анализ рынка, торговлю и другие финансовые операции.

Основные особенности и примеры открытого программного обеспечения в финансовой сфере включают в себя:

– гибкость и настраиваемость: открытое программное обеспечение в финансовой сфере часто предоставляет возможность настройки под конкретные потребности организации;

– прозрачность: благодаря доступу к исходному коду пользователи могут проверить безопасность и надежность программного обеспечения;

– экономия: использование открытого программного обеспечения может помочь снизить затраты на лицензии и обслуживание.

В качестве примеров открытого финансового программного обеспечения в финансовой сфере можно привести:

– Apache OFBiz (платформа для автоматизации бизнес-процессов, включающая в себя управление заказами, инвентаризацию и учет);

– QuantLib (библиотека для квантитативного финансового моделирования, предоставляющая инструменты для анализа рисков, оценки ценных бумаг и других финансовых операций);

– OpenGamma (платформа для управления рисками и аналитики в области финансовых рынков).

Открытое программное обеспечение для финансовой сферы, которое будет проанализировано в работе:

– Mastering Python for Finance[1]: книга для программирования в сфере финансов на Python с открытым исходным кодом;

– Finance[2]: набор инструментов, связанных с финансовой деятельностью. Включает в себя программное обеспечение, библиотеки, скрипты и другие компоненты, которые могут быть полезны для работы с финансовыми данными, анализа рынка или управления финансами;

– OpenBBTerminal[3]: платформа для работы с финансовыми данными и инструментами. Может предоставлять доступ к финансовым рынкам, аналитическим инструментам, торговым возможностям и другим функциям, необходимым для работы в области финансов и инвестиций.

Несмотря на риски, многие некрупные организации успешно используют открытое программное обеспечение в финансовой сфере, так как оно может предоставить гибкость, экономию и возможность контроля над развитием и поддержкой программного обеспечения.

[1] URL: https://github.com/jamesmawm/Mastering-Python-for-Finance-source-codes.
[2] URL: https://github.com/shashankvemuri/Finance.git.
[3] URL: https://github.com/OpenBB-finance/OpenBBTerminal.git.

1.3. Уязвимости в финансовом ПО и риски их реализации

Открытое финансовое программное обеспечение может быть подвержено различным уязвимостям. Некоторые из них включают в себя:

– уязвимости информационной безопасности: недостаточная защита от хакерских атак, включая взломы, фишинг и воздействие вредоносного кода.

– недостаточная конфиденциальность: некорректное управление личной информацией и конфиденциальными данными пользователей может привести к утечкам данных и нарушению приватности.

– регуляторные риски: отсутствие соответствия законодательству и регуляторным требованиям может привести к штрафам и запретам на ведение определенных видов деятельности.

– финансовые риски: возможность финансовых потерь из-за ошибок в программном обеспечении или изменений в рыночных условиях.

Несмотря на достоинства использования открытого программного обеспечения в финансовой сфере, риски его применения весьма велики:

Во-первых, недостаток поддержки программного обеспечения: некоторые проекты открытого программного обеспечения могут иметь ограниченную техническую поддержку. В таком случае возрастают временные и стоимостные затраты на поддержание программного обеспечения в оптимальном состоянии.

Во-вторых, соответствие требованиям: некоторые организации могут столкнуться с проблемами соответствия законодательству или регулятивным требованиям при использовании открытого программного обеспечения. Например, требования положений Банка России и Федеральной службы по техническому и экспортному контролю вряд ли будут соблюдаться при использовании такого программного обеспечения.

В-третьих, безопасность: пользователи могут быть обеспокоены безопасностью и надежностью открытого программного обеспечения в финансовой сфере из-за возможных уязвимостей или рисков безопасности.

При торговле, ведении бухгалтерии важно не допустить утечки информации о финансовых потоках в организации.

Для снижения рисков реализации уязвимостей в открытом финансовом программном обеспечении необходимо активно проводить аудит безопасности, использовать проверенные методы разработки программного обеспечения, обеспечивать регулярные обновления и улучшения защиты от хакерских атак. Помимо этого, важно соблюдать соответствие законодательству и регуляторным требованиям для минимизации рисков возникновения уязвимостей.

2. АНАЛИЗ ЗАЩИЩЕННОСТИ ОТКРЫТОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ФИНАНСОВОЙ СФЕРЫ

2.1. Цикл оценки защищенности открытого программного обеспечения для финансовой сферы

Оценка защищенности программного обеспечения является важным инструментом для обеспечения безопасности информационных систем, а также для соблюдения требований законодательства и стандартов безопасности. Цикл оценки защищенности открытого программного обеспечения для финансовой сферы может включать в себя следующие этапы:

1. Определение критически важных функций: выделение основных функций программного обеспечения, которые должны быть защищены от угроз безопасности.

2. Анализ уязвимостей программного обеспечения: проведение анализа уязвимостей открытого программного обеспечения с использованием специализированных инструментов (например, сканеры уязвимостей, для выявления потенциальных угроз информационной безопасности) для определения потенциальных угроз информационной безопасности.

3. Проверка соответствия стандартам информационной безопасности: оценка соответствия открытого программного обеспечения стандартам безопасности (например, ISO 27001, NIST SP 800-53, PCI DSS и другим).

4. Соблюдение законодательства и регулятивных требований: оценка соответствия открытого программного обеспечения требованиям законодательства и регулятивным нормам в финансовой сфере (например, Федеральным законам, положениям Банка России, нормативным документам Федеральной службы по техническому и экспортному контролю, Федеральной службы безопасности и др.).

5. Проведение аудита информационной безопасности программного обеспечения: проверка для выявления слабых мест и уязвимостей в программном обеспечении.

6. Проведение пентеста: проверка программного обеспечения в целях нахождения уязвимостей, которые потенциально могут быть использованы злоумышленниками.

7. Использование специализированных инструментов для проверки программного обеспечения: использование инструментов для мониторинга и обнаружения инцидентов информационной безопасности (например, системы обнаружения вторжений (IDS) и системы предотвращения вторжений (IPS)).

8. Своевременное обновление программного обеспечения: регулярное обновление открытого программного обеспечения в целях исправления обнаруженных ошибок предыдущих версий и закрытия уязвимостей.

9. Регулярное обучение персонала: обучение персонала по вопросам информационной безопасности и правильному использованию открытого программного обеспечения в целях минимизации рисков.

2.2. Реализация методики для поиска и устранения уязвимостей в открытом программном обеспечении для финансовой сферы

Для поиска и устранения уязвимостей в открытом программном обеспечении можно использовать следующие методики:

Анализ исходного кода: анализ исходного кода программного обеспечения с целью выявления потенциальных уязвимостей. Для этого можно использовать специализированные инструменты, например, статические анализаторы кода (OWASP Dependency-Check, Retire.js, Snyk и другие).

1. Проведение пентеста: контролируемая атака на программное обеспечение в целях выявления его уязвимостей. Проведение пентеста может помочь выявить уязвимости, которые не были обнаружены другими методами.

2. Использование уязвимостей известных библиотек: многие уязвимости связаны с использованием устаревших или уязвимых библиотек. Поиск и обновление таких библиотек может помочь устранить уязвимости.

3. Мониторинг обновлений безопасности: важно регулярно отслеживать обновления безопасности для всех используемых компонентов открытого программного обеспечения и оперативно устанавливать их.

4. Применение принципов безопасной разработки: включение в разработку принципов безопасной разработки, например, защита от инъекций, аутентификация и авторизация, обработка ошибок и т.д., поможет предотвратить возникновение уязвимостей.

5. Регулярное обновление и повторная проверка: после устранения уязвимостей необходимо проверять программное обеспечение на наличие новых уязвимостей и обновлять его.

6. Участие в сообществе: многие проекты открытого программного обеспечения имеют активные сообщества, в которых можно обсудить и выявить потенциальные уязвимости, а также получить рекомендации по их устранению.

Реализация этих методик в комбинации с использованием специализированных инструментов и технологий поможет обнаружить и устранить уязвимости в открытом программном обеспечении для финансовой сферы.

3. ПРАКТИЧЕСКИЙ ПРИМЕР

3.1. Реализация автоматизированного поиска уязвимостей с использованием языка Python и анализаторов кода с открытым исходным кодом

Реализация автоматизированного поиска уязвимостей с использованием языка Python и анализаторов кода с открытым исходным кодом позволяет обнаруживать потенциальные проблемы безопасности в программном обеспечении и типовые уязвимости. Для этого можно использовать библиотеки и программные продукты (например, Bandit, PyLint, Safety).

В процессе написания данной работы разработано программное средство, позволяющее анализировать репозитории Github, написанные на языке Python и используемые в финансовой сфере. С технической точки зрения, ограничение на исследование программного обеспечения финансовой сферы является чисто формальным.

В процессе работы произведен обзор библиотек и программного обеспечения, которое может быть использовано для анализа исходных текстов, произведена их оценка, и сделан вывод о целесообразности разработки программного обеспечения, которое может автоматизировать вызовы к ограниченному множеству программ для анализа исходных текстов из-за специфичности их применения. Данная разработка является актуальной не только из-за автоматизации вызовов анализаторов, но и из-за кросс-анализа выявленных уязвимостей, что позволяет увеличить глубину анализа по сравнению с применением единственного анализатора.

3.2. Библиотека Bandit

Bandit – это инструмент статического анализа кода на языке Python, который специализируется на поиске потенциальных уязвимостей

безопасности. Он обнаруживает уязвимости, такие как использование опасных функций, возможные атаки через внедрение кода, неправильное использование криптографических функций и другие потенциально опасные ситуации.

Bandit может быть применен для анализа как отдельных Python-скриптов, так и целых проектов. Он может использоваться как часть процесса непрерывной интеграции (CI) для автоматического обнаружения уязвимостей в коде перед его развертыванием.

Примеры уязвимостей, которые Bandit может обнаружить, включают:

– использование функций для выполнения произвольного кода;

– недостаточная проверка входных данных, что может привести к атакам внедрения кода;

– использование устаревших или слабых криптографических алгоритмов;

– нарушение правил безопасности при работе с сетевыми запросами.

Использование Bandit имеет ряд преимуществ:

– простота использования: Bandit легко интегрируется в процесс разработки благодаря своему простому интерфейсу;

– автоматизация: Bandit может быть запущен автоматически в рамках процесса непрерывной интеграции;

– обширные возможности поиска уязвимостей: Bandit обнаруживает широкий спектр потенциальных проблем безопасности.

При использовании Bandit возможно обнаружить и следующие недостатки:

– ложные срабатывания: как и любой инструмент статического анализа кода, Bandit может выдавать ложные срабатывания, требующие дополнительной проверки.

– ограничения по типам уязвимостей.

Данный пример демонстрирует использование анализатора кода Bandit для поиска уязвимостей в Python-скрипте. После запуска скрипта анализатор выдаст список найденных уязвимостей, которые могут быть потенциально опасными для безопасности программы.

```
Files in scope (1):
    C:\Users\shado\PycharmProjects\Sasha\Finance.git\tickers.py (score: {SEVERITY: 25, CONFIDENCE: 15})
Files excluded (0):

Test results:
>> Issue: [B113:request_without_timeout] Requests call without timeout
   Severity: Medium   Confidence: Low
   CWE: CWE-400 (https://cwe.mitre.org/data/definitions/400.html)
   More Info: https://bandit.readthedocs.io/en/1.7.7/plugins/b113_request_without_timeout.html
   Location: C:\Users\shado\PycharmProjects\Sasha\Finance.git\tickers.py:6:11
5       url = 'https://en.wikipedia.org/wiki/List_of_S%26P_500_companies'
6       html = requests.get(url).text
7       df = pd.read_html(html, header=0)[0]

--------------------------------------------------
>> Issue: [B113:request_without_timeout] Requests call without timeout
   Severity: Medium   Confidence: Low
   CWE: CWE-400 (https://cwe.mitre.org/data/definitions/400.html)
   More Info: https://bandit.readthedocs.io/en/1.7.7/plugins/b113_request_without_timeout.html
   Location: C:\Users\shado\PycharmProjects\Sasha\Finance.git\tickers.py:13:11
12      url = 'http://www.nasdaqtrader.com/dynamic/symdir/nasdaqlisted.txt'
13      data = requests.get(url).text
14      # The data is '|' separated and last two lines are not needed

--------------------------------------------------
>> Issue: [B113:request_without_timeout] Requests call without timeout
   Severity: Medium   Confidence: Low
   CWE: CWE-400 (https://cwe.mitre.org/data/definitions/400.html)
   More Info: https://bandit.readthedocs.io/en/1.7.7/plugins/b113_request_without_timeout.html
```

Рис. 1. Демонстрация использования анализатора кода Bandit

3.3. Программное обеспечение Safety

Safety – это инструмент, который предназначен для проверки зависимостей в проектах на языке Python на наличие известных уязвимостей безопасности. Он сканирует файлы зависимостей (например, requirements.txt) и сравнивает используемые версии библиотек с базой данных известных уязвимостей. Если обнаруживаются уязвимости, Safety предупреждает

об этом путем формирования отчета, позволяя разработчикам принять меры по обновлению зависимостей.

Примеры уязвимостей, которые Safety может обнаружить, включают:

– использование устаревших версий библиотек, в которых уже известны уязвимости;

– недостаточно обновленные зависимости, которые могут содержать исправленные уязвимости в новых версиях;

– зависимости с известными уязвимостями безопасности, которые могут быть использованы злоумышленниками для атак.

К плюсам использования Safety можно отнести:

– простоту интеграции: Safety легко интегрируется в процесс разработки благодаря своему простому интерфейсу и возможности автоматического запуска;

– обширную базу данных уязвимостей: Safety использует обширную базу данных известных уязвимостей, что позволяет обнаруживать широкий спектр потенциальных проблем безопасности;

– поддержку различных форматов файлов зависимостей: Safety может сканировать файлы зависимостей в различных форматах (например, requirements.txt, Pipfile.lock и другие).

Минусы использования Safety:

– ложные срабатывания: как и любой инструмент статического анализа кода, Safety может иногда выдавать ложные срабатывания, требующие дополнительной проверки;

– ограничения по типам уязвимостей: Safety специализируется на обнаружении известных уязвимостей и не всегда может обнаруживать новые или неизвестные угрозы безопасности.

В целом, Safety является полезным инструментом для обнаружения известных уязвимостей безопасности в зависимостях Python-проектов,

но его результаты всегда следует проверять и анализировать в контексте конкретного проекта.

Рассмотрим пример запуска утилиты:

```
safety check -r requirements.txt
```

Эта команда сканирует файл requirements.txt и проверяет его зависимости на наличие известных уязвимостей безопасности. Если уязвимости найдены, Safety выдаст соответствующие предупреждения.

Пример вывода Safety:

```
=== Running check for package vulnerabilities ===
nose (1.3.7) : [Low] Cross-Site Scripting in debug page
django (1.11.29) : [Critical] SQL Injection in ORM
requests (2.25.1) : [Medium] SSL/TLS Insecure Connection
```

В этом примере Safety обнаружил уязвимости в версиях библиотек nose, django и requests и указал их уровень критичности. Возможен анализ непосредственно requirement.txt или других файлов описывающих зависимости.

Рис. 2. Демонстрация использования анализатора кода Safety

Не всегда подобное утверждение является настоящей уязвимостью, ложные срабатывания должны обрабатываться разработчиком в ручном режиме.

3.4. Библиотека PyLint

PyLint – это инструмент статического анализа кода для языка Python, который предназначен для поиска потенциальных ошибок, стилистических несоответствий и других проблем в исходном коде.

PyLint может применяться в следующих целях:

– поиск потенциальных ошибок: PyLint анализирует код на предмет возможных ошибок, таких как использование неопределенных переменных, неправильное использование операторов и т.д.;

– проверка соответствия стандартам кодирования: PyLint проверяет код на соответствие стандартам оформления кода, определенным в PEP 8, что помогает поддерживать единообразие в стиле написания кода;

– обнаружение потенциальных уязвимостей безопасности: PyLint может помочь выявить некоторые типичные уязвимости безопасности (например, использование функций eval() или открытые соединения к базе данных).

К достоинствам применения PyLint можно отнести:

– помощь в выявлении потенциальных ошибок и уязвимостей в коде до его запуска, что позволяет предотвратить многие проблемы на ранних этапах разработки;

– помощь в соблюдении стандартов оформления кода и улучшении читаемости и поддерживаемости кода.

Среди минусов применения PyLint можно выделить:

– сложность настройки и запуска для больших проектов из-за большого количества предупреждений, которые PyLint может генерировать.

– Возможно потребуется некоторое время для изучения и понимания вывода результатов проверки, особенно для новых пользователей.

Пример вызова PyLint:

```
pylint [file][path]
```

PyLint проанализирует файл [file] и выдаст результаты своей проверки, включая предупреждения о потенциальных ошибках, стилистических рекомендациях и т.д.

При передаче параметра path PyLint проведет проверку для всех файлов в определенной директории.

Рис. 3. Демонстрация использования анализатора кода PyLint

При вызове PyLint можно использовать различные опции и флаги для настройки поведения анализатора. Например, можно указать определенные стандарты оформления кода для проверки или отключить определенные типы предупреждений. За счет указания оценки качества кода, разработчику легче производить его рефакторинг и другие улучшения в процессе CI/CD пайплайна.

3.5. Разработанное программное обеспечение

В связи с необходимостью проверки различными анализаторами исходного кода и неоднородностью их вывода формируются отчетные файлы для каждого из них.

Алгоритм работы программы:

Код выполняет ряд действий, связанных с загрузкой репозитория из Github, анализом его содержимого с помощью инструментов безопасности и статического анализа для файлов Python. Рассмотрим и подробно разберем каждую часть кода:

1. Импорт необходимых модулей:

– Repo: позволяет работать с репозиториями Git;

– os: для работы с операционной системой, файлами в частности;

– subprocess: для запуска внешних процессов;

– glob: b для работы с шаблонами файлов и путей.

2. Установка пути к папке для отчетов (report_path) и URL репозитория Github (repo_url).

3. Клонирование репозитория Github.

4. Проверка наличия уязвимостей с помощью инструмента Safety:

– выполняется сканирование безопасности с помощью инструмента Safety;

– результат сканирования записывается в файл safety<repo_path>.txt.

5. Проверка наличия уязвимостей с помощью инструмента Bandit:

– для каждого файла Python в репозитории выполняется проверка;

– результаты проверки записываются в файл bandit<repo_path>.txt.

6. Статический анализ с помощью инструмента Pylint:

– для каждого файла Python в репозитории выполняется статический анализ с использованием Pylint;

– результаты анализа записываются в файл pylint<repo_path>.txt.

8. Выводятся сообщения о завершении работы каждого этапа.

Непосредственно сама программа представлена на скриншотах и в электронном приложении к отчету:

```python
from git import Repo
import os
import subprocess
import glob

if __name__ == '__main__':
    report_path = "C:/Users/Public"
    ## = "./FIN2"
    # repo_url = "https://github.com/jamesmawm/Mastering-Python-for-Finance-source-codes"
    repo_url = "https://github.com/shashankvemuri/Finance.git"
    # repo_url = "https://github.com/OpenBB-finance/OpenBBTerminal.git"
    print("downloading repo: " + repo_url)
    repo_path = repo_url.split("/")[-1]

    if not os.path.exists(repo_path):
        Repo.clone_from(repo_url, repo_path)
        print("download finished")
    else:
        print("Repo exist")

    ##safety checks
    print("safety started to work...")
    os.chdir(repo_path)
    with open(report_path + '/safety' + repo_path + '.txt', 'w+') as outfile:
        subprocess.run( args: ['safety', 'scan', '--target', './'], stdout=outfile)
    print("Safety check file provided: " + report_path + '/safety.txt')

    # bandit checks
    print("bandit started to work...")
    with open(report_path + '/bandit' + repo_path + ".txt", 'w+') as outfile:
        for cfile in glob.glob(os.getcwd() + '/**/*.py', recursive=True):
            print("bandit check file:" + cfile)
            outfile.write("-----" + cfile + "-----" + "\n")
            proc = subprocess.run( args: ['bandit', cfile, '-ll', '-v'], stdout=outfile)
    print("bandit end his scan:" + report_path + '/bandit' + repo_path + ".txt")
```

Рис. 4. Код разработанной программы на языке Python. Часть 1.

```python
    ## Pylint
    print("Pylint started to work...")
    # with open(report_path + '/pylint' + repo_path + ".txt", "w") as outfile:
    #     subprocess.run(['pylint', repo_path], stdout=outfile)
    #     print("Pylint ended work." + report_path + "/pylint.txt")

    with open(report_path + '/pylint' + repo_path + ".txt", 'w+') as outfile:
        for cfile in glob.glob(os.getcwd() + '/**/*.py', recursive=True):
            print("Pylint check file:" + cfile)
            with open(cfile, 'r') as file:
                outfile.write("-----" + cfile + "-----" + "\n")
                subprocess.run( args: ['pylint', cfile], stdout=outfile)
    print("Pylint ended work." + report_path + '/pylint' + repo_path + ".txt")
```

Рис. 5. Код разработанной программы на языке Python. Часть 2.

Результатом работы программы являются три файла, по одному на каждый используемый инструмент анализа. Файлы представлены в текстовом виде, могут быть открыты любым сопутствующим инструментом. Для демонстрации работы программы проводился анализ репозиториев из пункта 1.2. отчета, в частности:

1. http://github.com/jamesmawm/Mastering-Python-for-Finance-source-codes.

2. https://github.com/shashankvemuri/Finance.git.

3. https://github.com/OpenBB-finance/OpenBBTerminal.git

В результате были получено 9 файлов с отчетными материалами, представленные на рисунке 6.

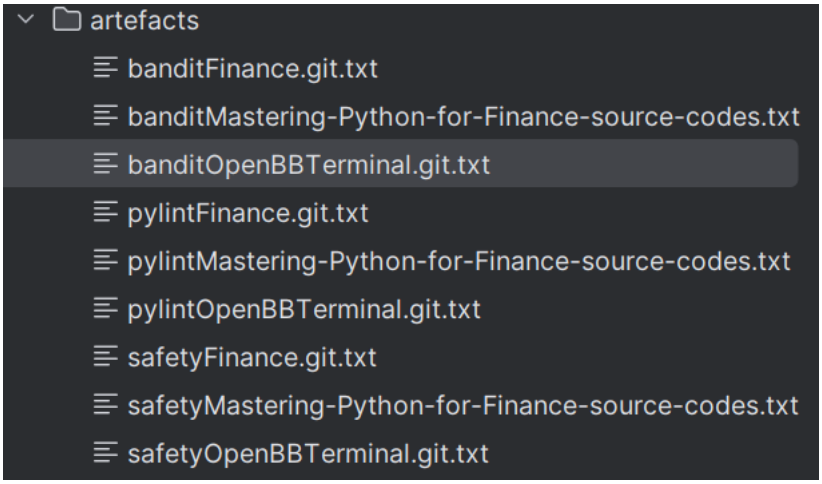

Рис. 6. Отчетные материалы программы.

Каждый из файлов имеет собственную структуру, соответствующую выводу используемого анализатора. На рисунках 7 - 9 представлен сформированный вывод.

Рис. 7. Отчетные материалы программы, анализатор Bandit.

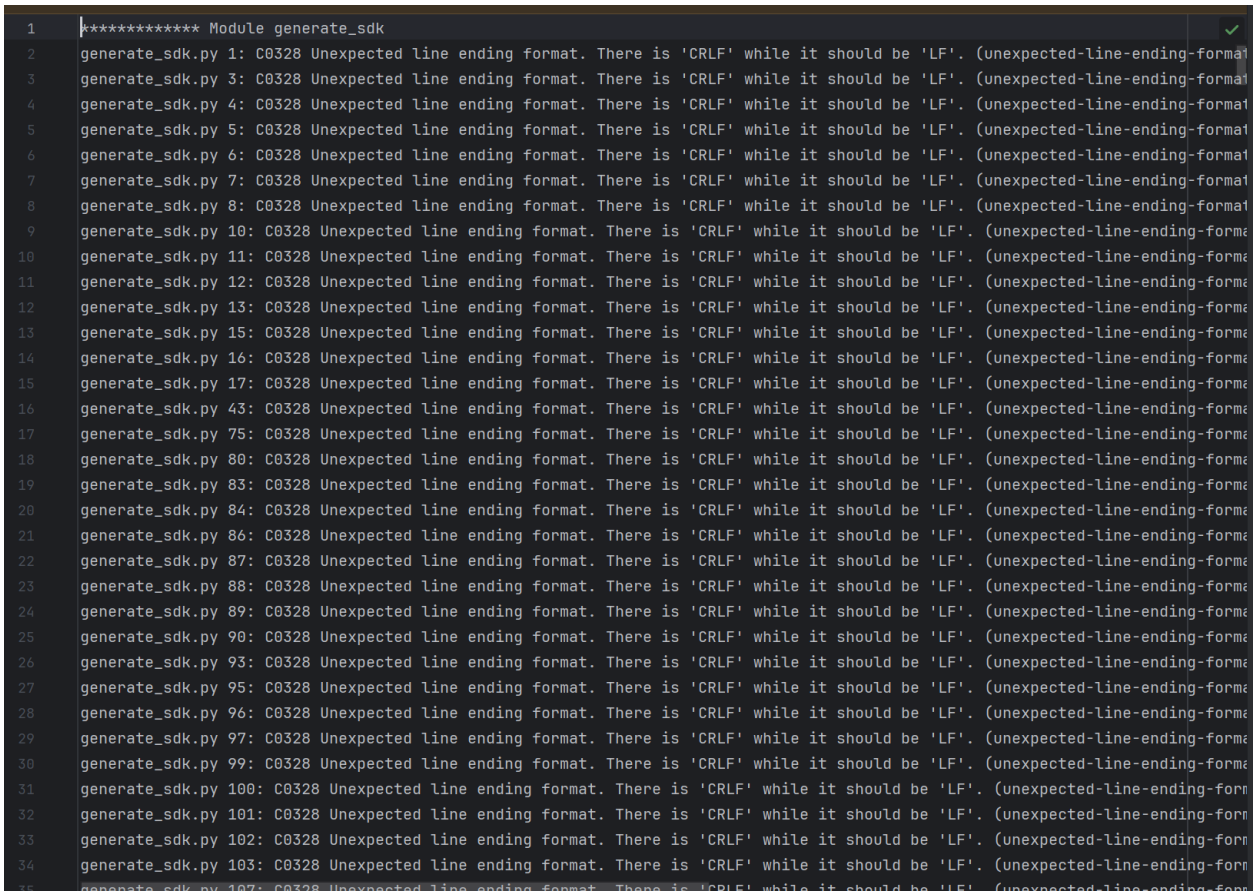

Рис. 8. Отчетные материалы программы, анализатор Pylint

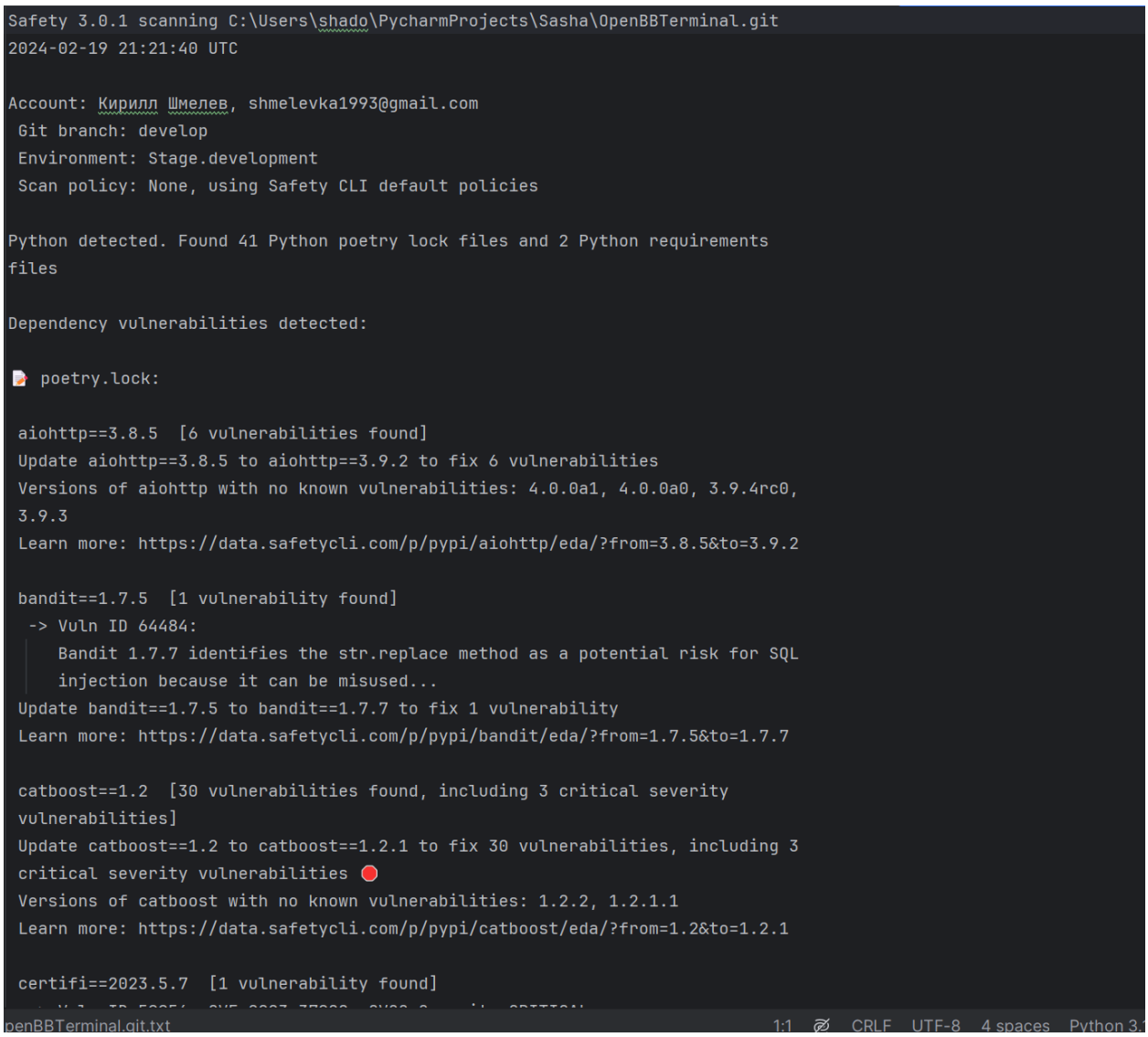

Рис. 9. Отчетные материалы программы, анализатор Pylint

Проанализировав вывод программы, можно утверждать, что используемые анализаторы дополняют друг друга и позволяют обеспечивать лучшее качество программного продукта.

Разработанная программа является актуальной и полезной для разработчиков и сотрудников подразделений информационной безопасности по нескольким причинам:

1. Автоматизация процесса безопасности: программа автоматизирует процесс загрузки репозитория, сканирования его содержимого на наличие уязвимостей и проведения статического анализа кода. Это позволяет быстро

выявлять потенциальные проблемы без необходимости выполнять каждую задачу вручную.

2. Улучшение безопасности кода: проведение сканирования на уязвимости и статического анализа помогает выявлять потенциальные уязвимости и ошибки в коде, что способствует повышению безопасности приложений.

3. Масштабируемость и повторное использование: программа может быть легко адаптирована и расширена для работы с различными репозиториями и инструментами безопасности и выстраивания непрерывного процесса, что делает ее универсальным инструментом для обеспечения безопасности кода.

Рассмотрим возможность расширения функционала разработанного программного средства. В случае необходимости возможно добавление следующих возможностей:

– интеграция с другими инструментами безопасности, такими как сборка сформированных отчетов для дальнейшей централизованной обработки, проверка конфиденциальной информации и расширение перечня используемых инструментов;

– отправка уведомлений о результатах анализа: добавление функционала для отправки уведомлений о результатах анализа по электронной почте или мессенджерам. Для отдельных инструментов возможно оповещение в реальном времени при ухудшении показателей (Pylint/Bandit).

– создание отчетов с графиками/диаграммами или другой человеко-читаемый формат для наглядного представления результатов анализа.

Данная программа не решает проблемы вовлеченности программиста в анализ найденных уязвимостей и недоработок исходного кода, но позволяет ориентироваться на несколько инструментов с различными базами знаний, что улучшает качество конечного продукта и позволяет вести контроль

за закрытием уязвимостей административными методами (из-за читаемого формата вывода).

ЗАКЛЮЧЕНИЕ

Открытое программное обеспечение широко используется в финансовой сфере. Несмотря на возможные риски, многие некрупные организации успешно используют открытое программное обеспечение в финансовой сфере, так как оно может предоставить гибкость, экономию и возможность контроля над развитием и поддержкой программного обеспечения.

Приведенная в работе методика для поиска и устранения уязвимостей в открытом программном обеспечении в комбинации с использованием специализированных инструментов и технологий поможет обнаружить и устранить уязвимости в открытом программном обеспечении для финансовой сферы.

Рассмотренные в работе анализаторы исходного кода с открытым исходным кодом PyLint, Safety и Bandit, вызванные с помощью написанного программного кода позволят улучшить качество конечного продукта вести контроль за закрытием уязвимостей административными методами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Смирнов А.В. Открытое программное обеспечение: Возможности и перспективы.

2. Лазарев А.А. Открытое программное обеспечение в России: состояние и перспективы развития.

3. Поддубный А.В. Открытое программное обеспечение: Опыт использования в России.

4. Олексин И.В. Применение открытого программного обеспечения как одна из мер импортозамещения // 2023. Электронный ресурс: https://novainfo.ru/article/19758.

5. Доклад The Linux Foundation о применении открытого программного обеспечения в финансовой сфере. URL: https://www.linuxfoundation.org/publications/open-source-in-the-financial-services-industry.

6. Официальный сайт Open Source Initiative. URL: https://opensource.org.

7. Документация Pylint. URL: https://pylint.readthedocs.io/en/latest/index.html.

8. How to use GitPython to pull remote repository. URL: https://www.askpython.com/python/examples/gitpython-to-pull-remote-repository.

9. Примеры работ с процессами. URL: https://python-scripts.com/subprocess.

10. Функция glob() модуля glob в Python. URL: https://docs-python.ru/standart-library/modul-glob-python/funktsija-glob-modulja-glob.

11. Документация Safety. URL: https://pypi.org/project/safety.

12. Документация Bandit. URL: https://bandit.readthedocs.io/en/latest.

13. Ищем уязвимости в Python-коде с помощью open source инструмента Bandit. URL: https://habr.com/ru/companies/macloud/articles/563246.

14. Репозиторий Github. URL: https://github.com/jamesmawm/Mastering-Python-for-Finance-source-codes.

15. Репозиторий Github. URL: https://github.com/shashankvemuri/Finance.git.

16. Репозиторий Github. URL: https://github.com/OpenBB-finance/OpenBBTerminal.git.