

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждениевысшего образования

**«МИРЭА — Российский технологический университет»**

**РТУ МИРЭА**

|  |  |  |
| --- | --- | --- |
| **Институт искусственного интеллекта**  **Базовая кафедра № 252 «Информационная безопасность»** | | |
|  |

**Практическая работа**

**«Статический анализ криптографической библиотеки»**

Студенты: Филиппов Д.В., Худобин В.С.

Группа: ККСО-01-19

**Оглавление**

[Введение 3](#_Toc155900720)

[Подготовка к работе 4](#_Toc155900721)

[Выполнение работы 5](#_Toc155900722)

[Библиотека Circl 5](#_Toc155900723)

[Анализатор Semgrep 5](#_Toc155900724)

[Анализатор Bearer 9](#_Toc155900725)

[Библиотека Cryptoswitch 12](#_Toc155900726)

[Анализатор Semgrep 12](#_Toc155900727)

[Анализатор Bearer 13](#_Toc155900728)

[Сравнительный анализ 15](#_Toc155900729)

[Заключение 17](#_Toc155900730)

[Список литературы 18](#_Toc155900731)

# **Введение**

В современном мире разработки программного обеспечения использование открытых исходных кодов стало неотъемлемой частью создания высококачественных приложений. Открытые библиотеки предоставляют разработчикам уникальную возможность использовать готовые решения, ускоряя процесс разработки и снижая вероятность ошибок. Однако внедрение стороннего кода также влечет за собой ряд потенциальных рисков, связанных с безопасностью, производительностью и поддержкой.

Цель данной практической работы заключается в проведении анализа открытой библиотеки с использованием статического анализатора кода. Статический анализ позволяет выявить потенциальные проблемы в исходном коде до его выполнения, обеспечивая раннее обнаружение ошибок и улучшение общего качества программы. В контексте открытых библиотек это особенно важно, поскольку это позволяет разработчикам удостовериться в надежности и безопасности стороннего кода перед его внедрением в свой проект.

В текущей работе будут рассматриваться библиотеки Сryptoswitch и Circl, которые реализованы на языке программирования Golang. Статические анализаторы, использующиеся в рамках данного анализа – Semgrep и Bearer.

# **Подготовка к работе**

В качестве исследуемых библиотек будет использоваться Circl и Сryptoswitch с реализацией на языке Golang. Данные библиотеки предоставляют абстракции для ряда криптографических задач, таких как шифрование, создание цифровой подписи, обмен ключами и другие.

1. Скачиваем исходный код из репозитория [Circl](https://github.com/cloudflare/circl) и [Сryptoswitch](https://github.com/elizarpif/cryptoswitch).

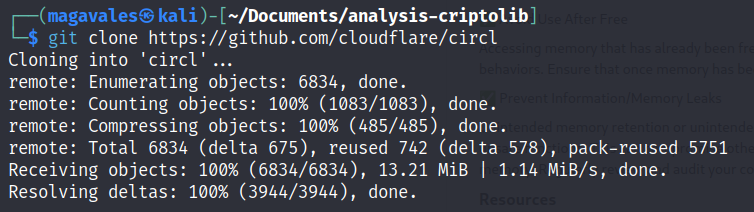


Рисунок 1. Скачивание проекта Circl

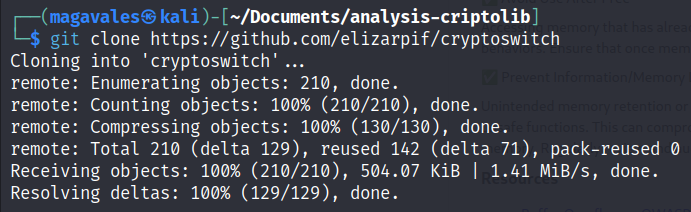


Рисунок 2. Скачивание проекта Cryptoswitch

1. Установка анализаторов Bearer и Semgrep.

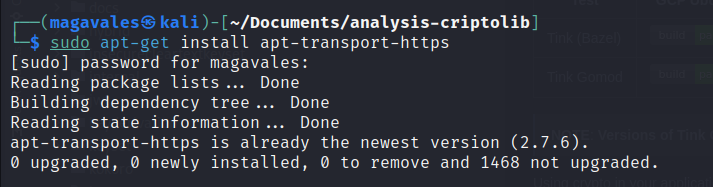


Рисунок 3. Установка анализатора Bearer ч.1



Рисунок 4. Установка анализатора Bearer ч.2

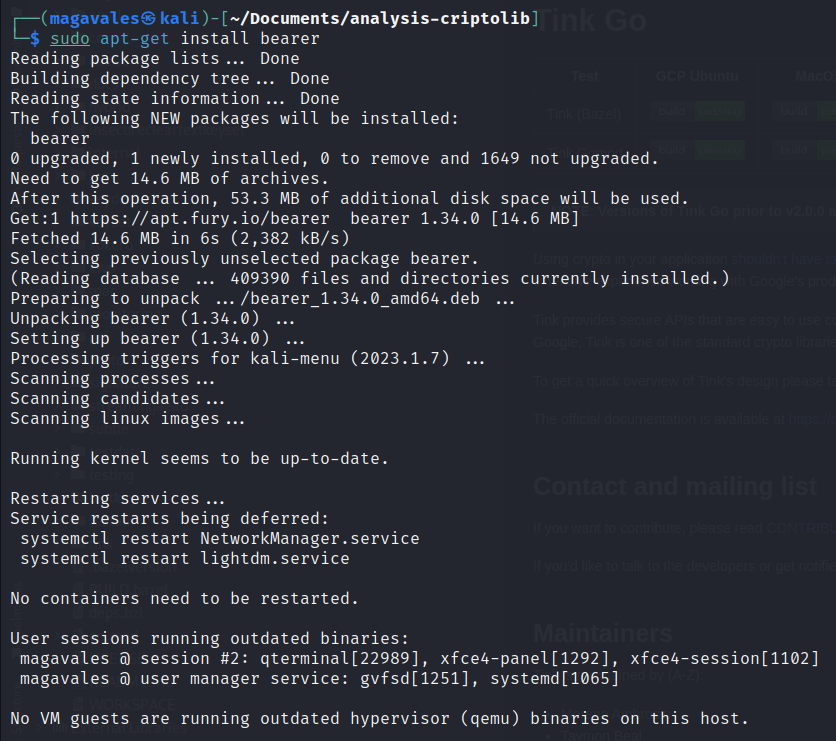


Рисунок 5. Установка анализатора Bearer ч.3

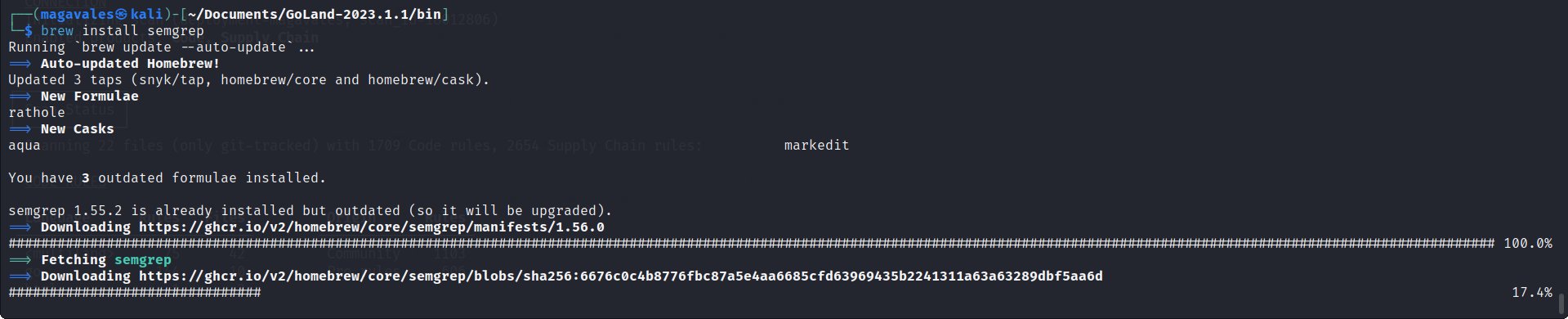


Рисунок 6. Установка анализатора Semgrep

# **Выполнение работы**

## **Библиотека Circl**

### Анализатор Semgrep

1. Запускаем статический анализатор:

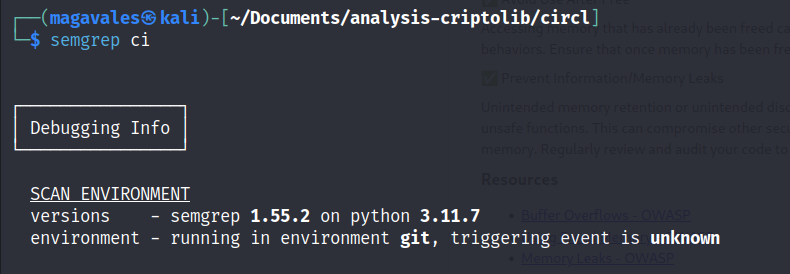


Рисунок 7. Работа анализатора Semgrep

1. Рассмотрим полученные результаты:

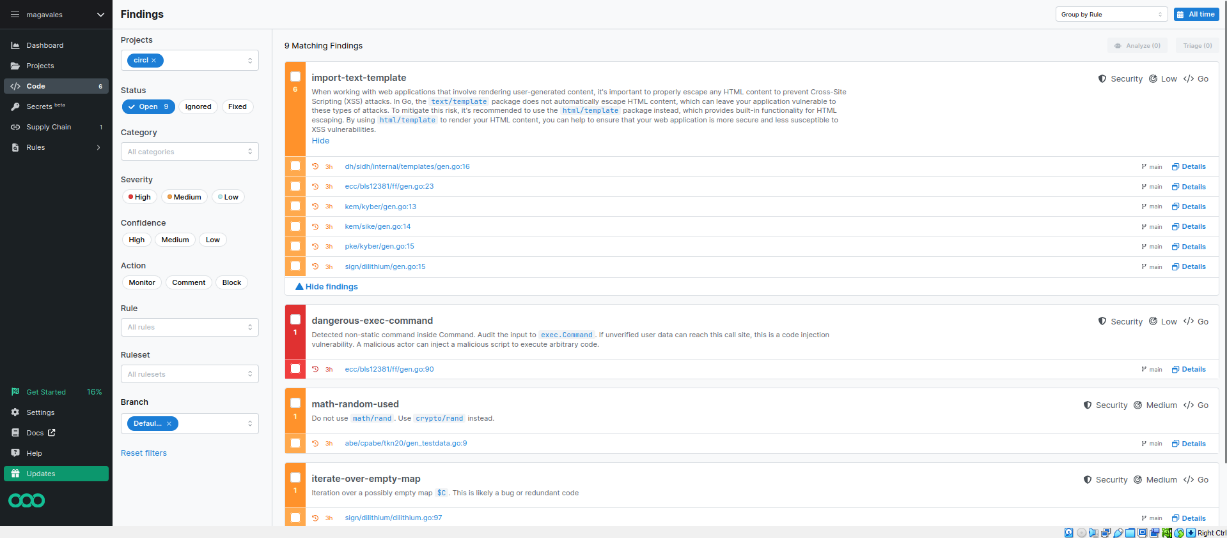


Рисунок 8. Полученные результаты от Semgrep

Первая ошибка, полученная в ходе работы анализатора Semgrep, связана с импортом стандартной библиотеки «text/template» и ее использованием.

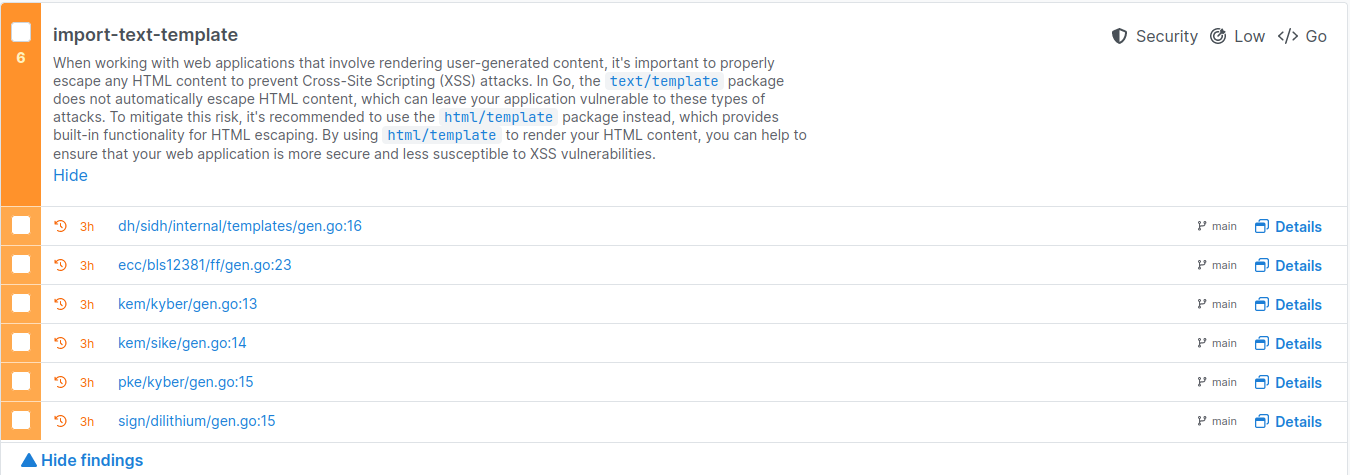


Рисунок 9. Ошибка, связанная с импортом и использование библиотеки

Анализатор посчитал, что в рамках данной библиотеки было бы разумно использовать другую библиотеку – «html/template», так как она является более безопасной для представления выходных html-файлов, так как предоставляет защиту. В рамках рассматриваемой библиотеки не было обнаружено использование html-файлов в том или ином виде, поэтому использование предложенной библиотеки может быть избыточным. Следовательно, данную ошибку стоит отнести к ложноположительному типу.

Вторая ошибка – ошибка высокого уровня, а следовательно, представляет большую угрозу для пользователя рассматриваемой библиотеки.

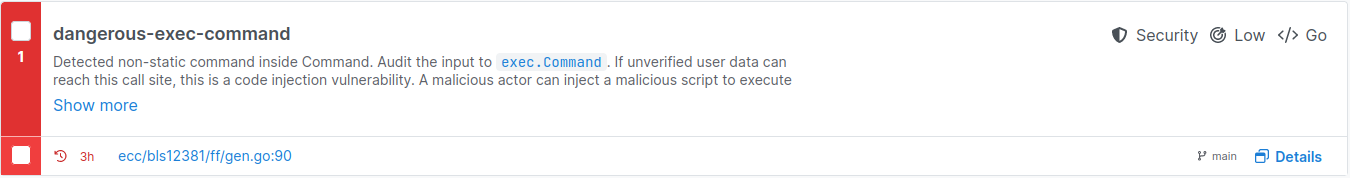


Рисунок 10. Уязвимость класса инъекция

Данная уязвимость опасна тем, что в программе может быть выполнен вредоносный код злоумышленника. Описанная возможность имеет шансы на исполнение, так как некоторые данные пользователь должен вводить и эти данные поступают в функции, которые производят их выполнение в консоли. На следующем рисунки представлена детальная ситуация по использованию введенных пользователем данных.

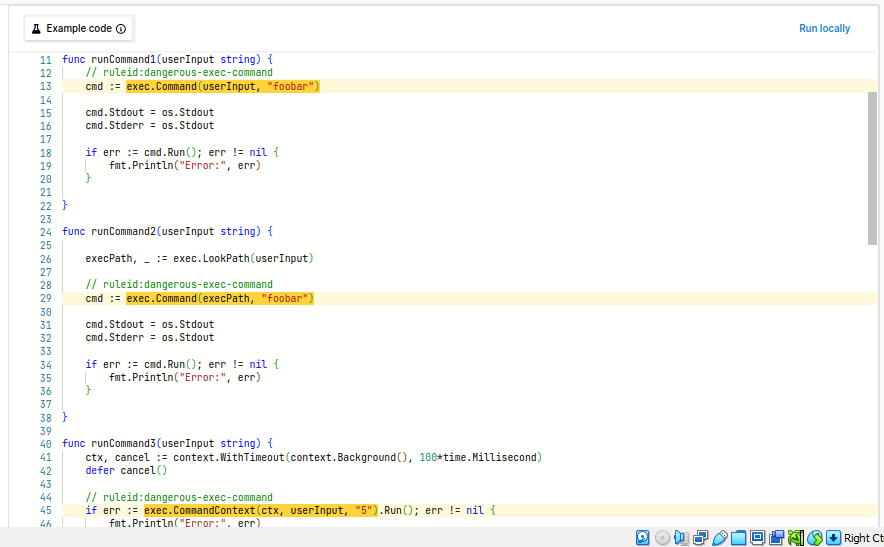


Рисунок 11. Детали отчета по уязвимости класса инъекция

На представленном выше рисунке можно увидеть, что данные функции могут позволить выполнить вредоносный код, который будет внедрен по средствам инъекции. Более подробно можно ознакомиться на сайте [OWASP 10](https://owasp.org/Top10/A03_2021-Injection). Получается, что данная ошибка является потенциальным направлением атаки, следовательно, необходимо либо делать данные статичными для того, чтобы использовать консоль, либо пытаться делать их более безопасными.

Третья ошибка связана с использование псевдослучайного генератора значений из стандартной библиотеки языка.

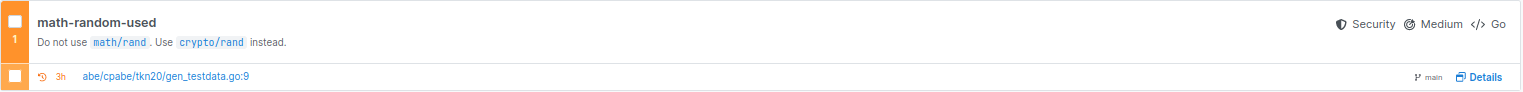


Рисунок 12. Использование небезопасной функции генерации случайных значений

Генерируемые значения нельзя использовать для криптографических потребностей, поскольку результат является предсказуемым. В рамках рассматриваемой библиотеки полученная ошибка связана с файлом, который отвечает за тестовые данные. Получаем, что данную ошибку стоит отнести к ложному срабатыванию.

Четвертая ошибка – возможное итерирование по пустой *map*.

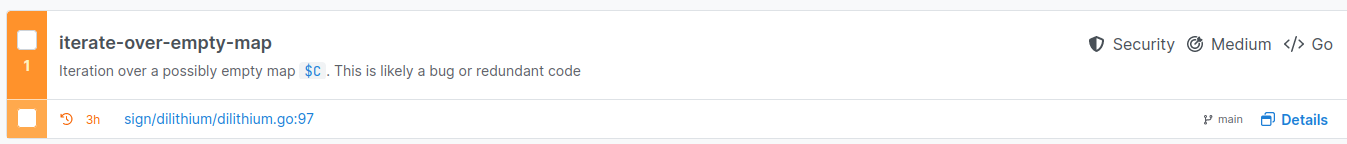


Рисунок 13. Возможное итерирование по пустой map

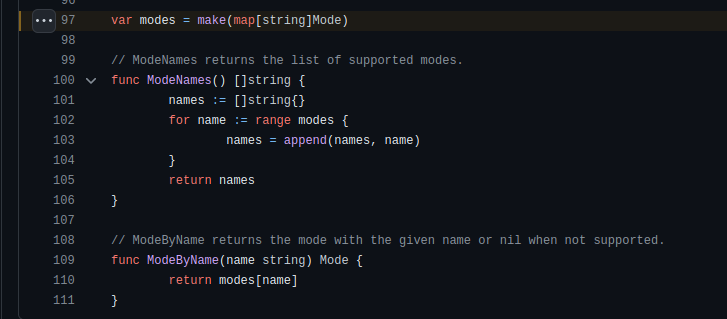


Рисунок 14. Фрагмент кода из библиотеки Circl

Полученная ошибка представляет опасность для проектов, использующих данную библиотеку тем, что возможна некорректная работа программы из-за итерации по пустой *map*.

### Анализатор Bearer

1. Запускаем статический анализатор:

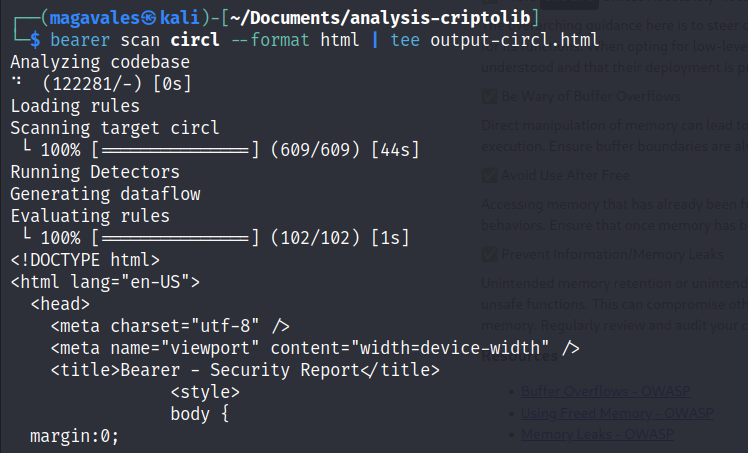


Рисунок 15. Работа анализатора Bearer

1. Рассмотрим полученные результаты в ходе работы анализатора Bearer:

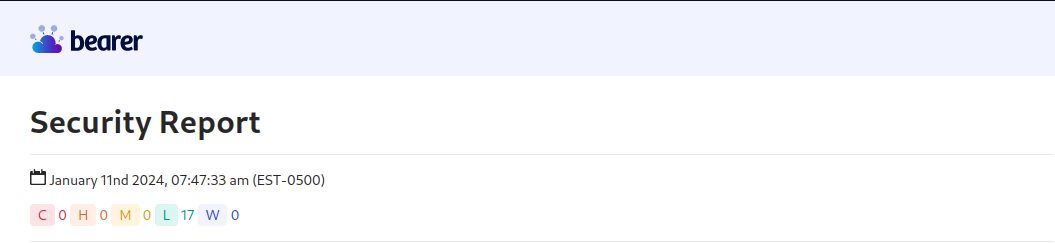


Рисунок 16. Результаты работы анализатора Bearer

Все полученные 17 ошибок являются однотипными и касаются использования пакета «unsafe».

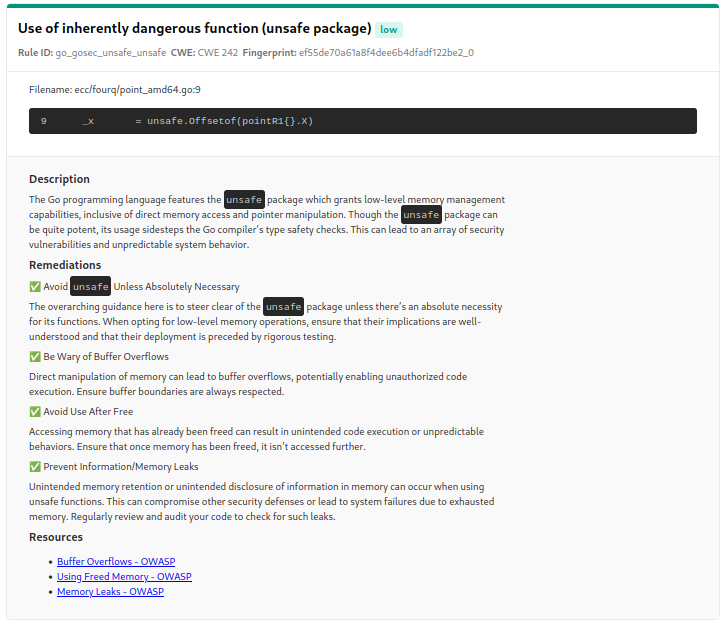


Рисунок 17. Использование пакета "unsafe" ч.1

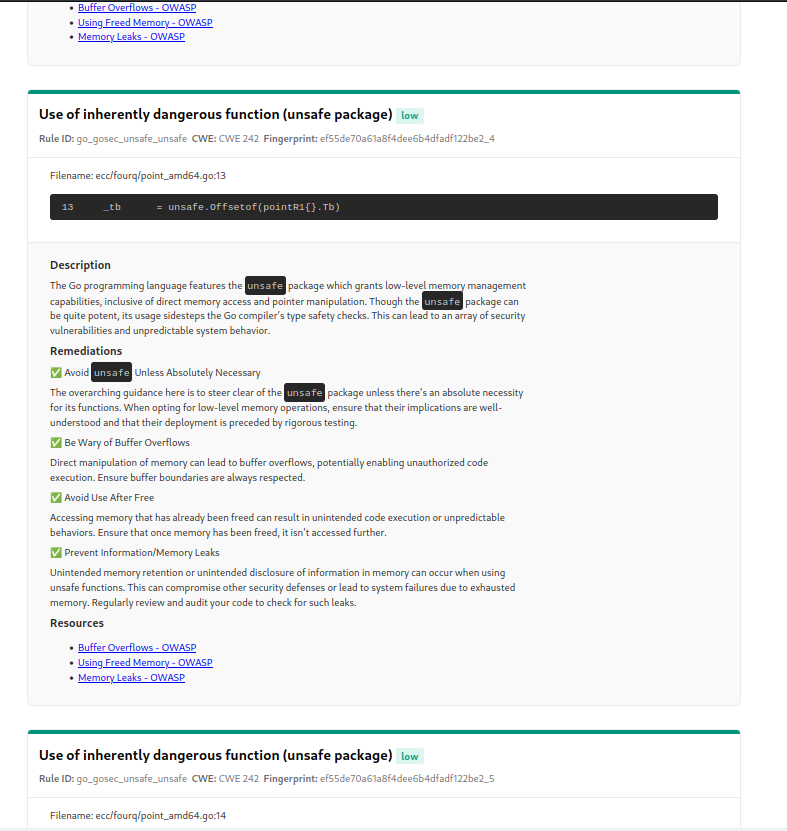


Рисунок 18. Использование пакета "unsafe" ч.2

Данный пакет присутствует в языке программирования Go и предоставляет возможность к низкоуровневому управлению памяти, а также прямой доступ к самой памяти и позволяется манипулировать указателями. Использование пакета «unsafe» позволяет обходить проверки безопасности типов, которые по умолчанию используются в Go. Такое использование может привести к различным уязвимостям безопасности и непредсказуемому поведению системы. В рассматриваемой библиотеке применение пакета «unsafe» обусловлено необходимостью оптимизации работы под определенную аппаратную составляющую системы. Однако, стоит все же подумать над другими методами реализации и избегать использование данного пакета.

## **Библиотека** **Cryptoswitch**

### Анализатор Semgrep

1. Запускаем статический анализатор:

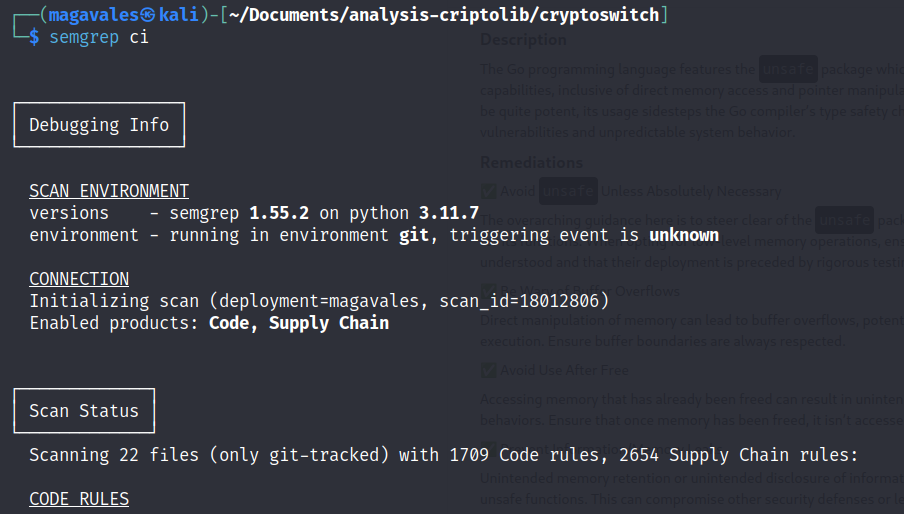


Рисунок 19. Работа анализатора Semgrep

1. Рассмотрим полученные результаты:

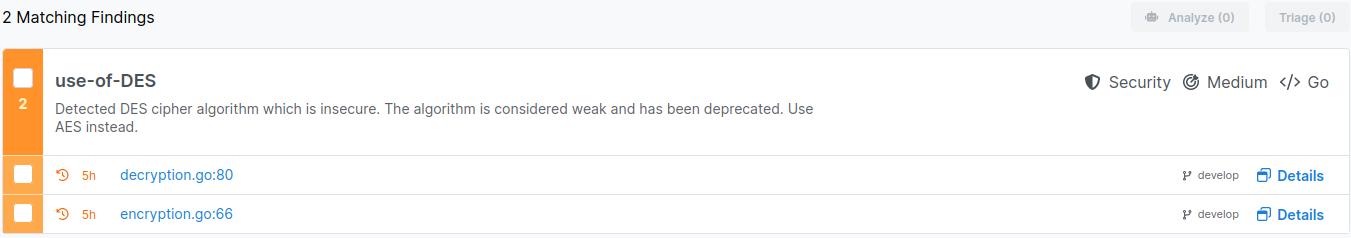


Рисунок 20. Полученные результаты от Semgrep

При анализе библиотеки Cryptoswitch статистический анализатор Semgrep обнаружил единственную ошибку – использование алгоритма DES. Рассматриваемая библиотека использует несколько алгоритмов: AES, DES, Camelia, Twofish.

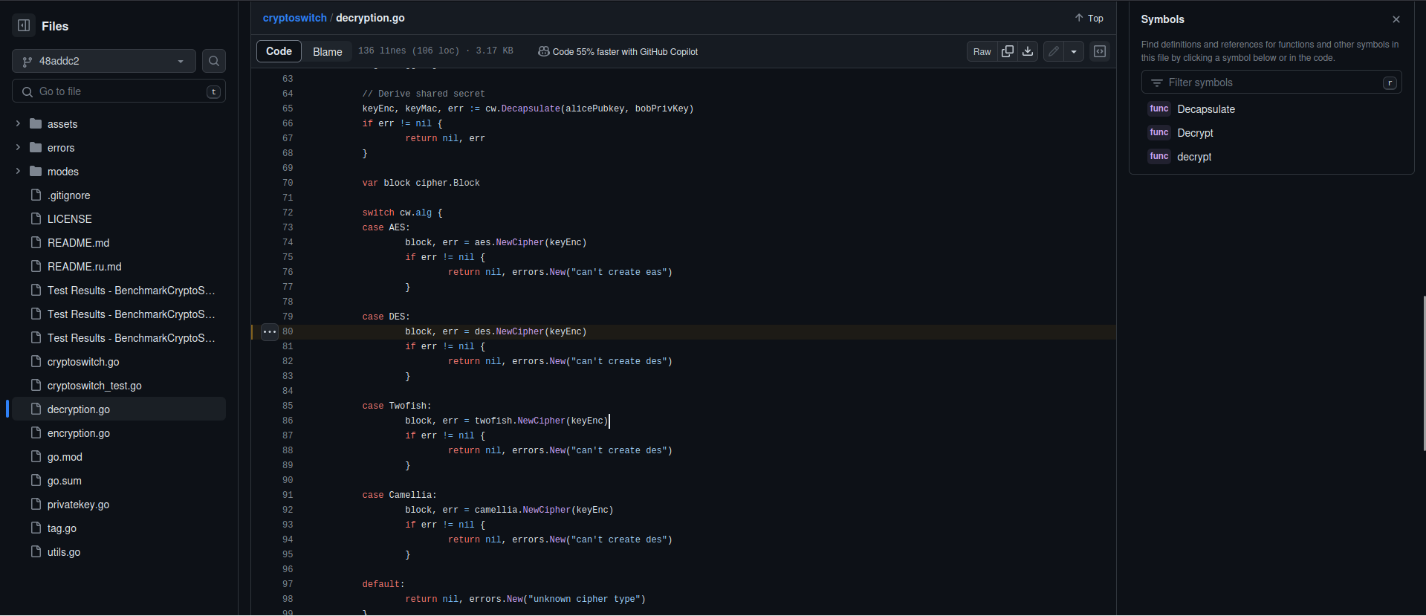


Рисунок 21. Список используемых алгоритмов

Все перечисленные алгоритмы являются достаточными стойкими за исключением DES, так как он использует ключ длиной 56 бит, что для современного применения в системах является небезопасным, а также алгоритм не имеет достаточно хорошего лавинного эффекта. Для нивелирования данной проблемы стоит задуматься о том, чтобы отказаться от применения алгоритма DES.

### Анализатор Bearer

1. Запускаем статический анализатор:



Рисунок 22. Работа анализатора Bearer

1. Рассмотрим полученные результаты:

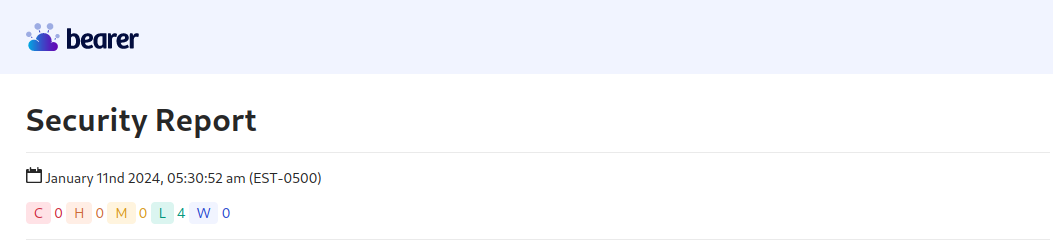


Рисунок 23. Результаты полученные от Bearer

Анализатор обнаружил 4 ошибки, которые связаны с использование алгоритма DES.

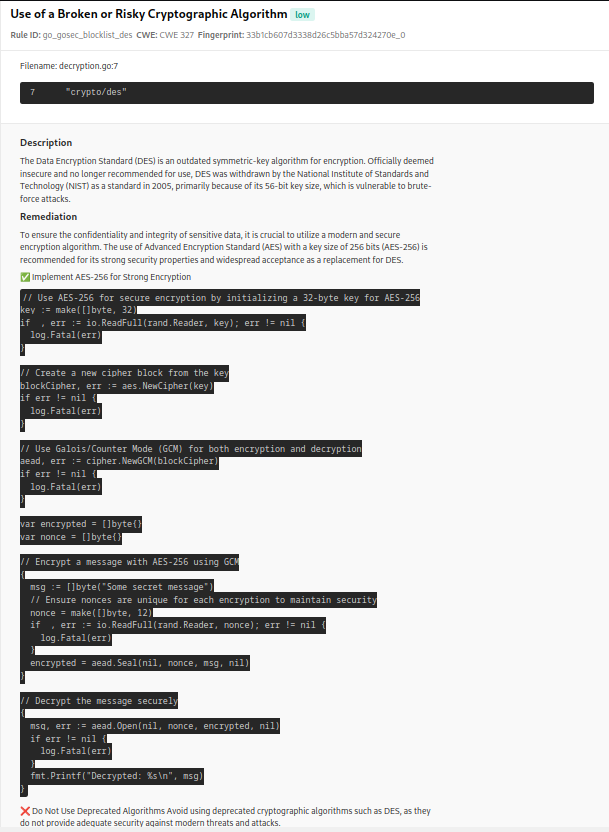


Рисунок 24. Использование алгоритма DES

Найденные анализатором Bearer ошибки идентичны тем, которые были найдены анализатором Semgrep.

# **Сравнительный анализ**

При применении двух различных анализаторов – Semgrep и Bearer – к библиотеке Cryptoswitch получились схожие результаты. Оба статистических анализатора в своих отчетах отметили следующую ошибку – использования алгоритма DES, который в свою очередь является недостаточно безопасным. В рамках сравнения анализаторов получаем, что они отработали одинаково хорошо, так как подтвердили полученные результаты друг друга.

Применяя те же анализаторы к библиотеке Circl, получаем различные результаты. Bearer нашел в общей сумме 17 ошибок, но они относятся к одному типу – использование пакета «unsafe». Найденные угрозы имеют статус *low,* и они достаточно опасны, но все же использование пакета может быть безопасным и пойти на пользу – ускорение работы при применении библиотек в различных аппаратных архитектурах. Поэтому есть вариант продолжать использовать пакет, но принимать во внимание возможные угрозы, которые могут произойти. Semgrep нашел 4 ошибки – 1 уровня high и 2 уровня medium. Как можно заметить, работа анализатора Bearer получилась менее эффективной, так как Semgrep обратил внимание на участок кода, где возможно произвести инъекцию, которая относится к OWASP 10. Данный факт очень интересен и важен, так как он на прямую влияет на безопасность пользователей, использующих рассматриваемую библиотеку для шифрования своих данных. Также была найдена еще одна интересная ошибка – итерация по возможной пустой map. Данная ошибку нельзя игнорировать, потому что есть вероятность аварийного завершения программы, использующей библиотеку Circl.

В рамках данного сравнительного анализа получаем, что анализатор Semgrep отработал лучше и более качественно, чем его визави – Bearer. Такая работа анализатора Bearer может быть обоснована тем, что он еще недостаточно оптимизирован к программам и библиотекам, написанным на языке Golang. Каждый из анализатор имеет право на использование, но Semgrep показал более качественную работу с языком Golang, по сравнению с Bearer.

# **Заключение**

В ходе выполнения данной практической работы мы выявили, важность использования статических анализаторов Semgrep и Bearer, при оценке безопасности криптографических библиотек, в частности, Circl и Cryptoswitch. При помощи применения статического анализатора нам удалось обнаружить потенциальные уязвимые места в кодовой базе библиотек, связанные с различными элементами. Анализируя, мы установили, что были как ложноположительные срабатывания, так и реально-существующие недочёты, которые мы описали и предложили возможные варианты устранения проблемных участков кода или рекомендации по избежанию атак на данные компоненты. Также был проведен сравнительный анализ двух анализаторов, Semgrep и Bearer, который также дал интересные результаты, которые стоит учесть при дальнейшем анализе проектов написанных на Golang.

Подытоживая, важно отметить, что обеспечение безопасности — это постоянный процесс улучшения, и необходимо регулярно проводить аудиты и анализы, чтобы поддерживать высокие стандарты безопасности и оперативно реагировать на новые вызовы и угрозы.

# **Список литературы**

1. Электронный ресурс – Bearer – Bearer CLI – URL: https://docs.bearer.com
2. Электронный ресурс – Semgrep – Semgrep – URL: https://semgrep.dev/docs/
3. Электронный ресурс – CloudFlare – Введение в Circl – URL: https://blog.cloudflare.com/introducing-circl/
4. Электронный ресурс – Github – cloudflare/circl – URL: https://github.com/cloudflare/circl
5. Электронный ресурс – Github – elizarpif/cryptoswitch – URL: https://github.com/elizarpif/cryptoswitch