



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение.

высшего 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

Москва 2024

Оглавление

| | |
|------------------------------|----|
| Введение..... | 3 |
| Подготовка к работе | 4 |
| Выполнение работы | 5 |
| Библиотека Circl | 5 |
| Анализатор Sengrep..... | 5 |
| Анализатор Bearer | 9 |
| Библиотека Cryptoswitch..... | 12 |
| Анализатор Sengrep..... | 12 |
| Анализатор Bearer | 13 |
| Сравнительный анализ | 15 |
| Заключение..... | 17 |
| Список литературы | 18 |

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.

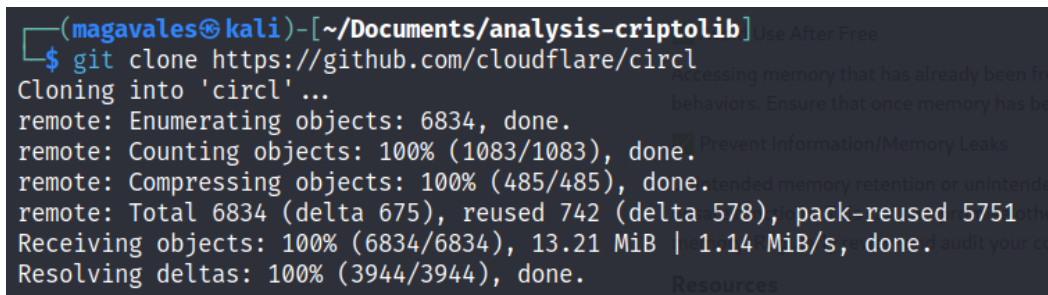
Цель данной практической работы заключается в проведении анализа открытой библиотеки с использованием статического анализатора кода. Статический анализ позволяет выявить потенциальные проблемы в исходном коде до его выполнения, обеспечивая раннее обнаружение ошибок и улучшение общего качества программы. 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 Cryptoswitch и Circl, которые реализованы на языке программирования Golang. Статические анализаторы, используемые в рамках данного анализа – Semgrep и Bearer.

Подготовка к работе

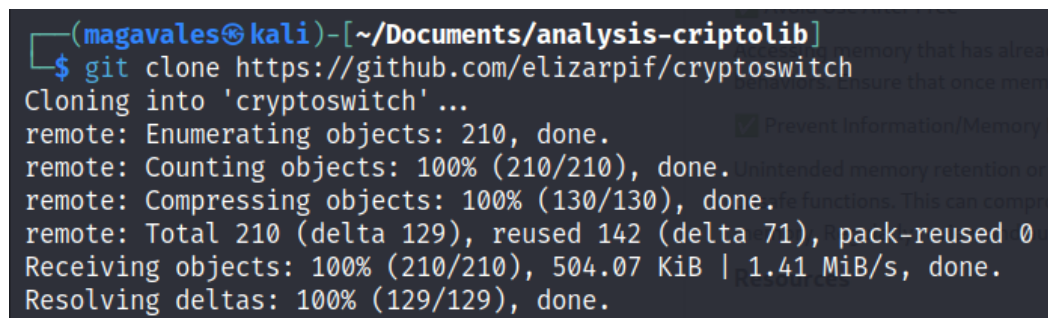
В качестве исследуемых библиотек будет использоваться Circl и Cryptoswitch с реализацией на языке Golang. Данные библиотеки предоставляют абстракции для ряда криптографических задач, таких как шифрование, создание цифровой подписи, обмен ключами и другие.

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



```
(magavales@kali)-[~/Documents/analysis-criptolib]
$ git clone https://github.com/cloudflare/circl
Cloning into 'circl' ...
remote: Enumerating objects: 6834, done.
remote: Counting objects: 100% (1083/1083), done.
remote: Compressing objects: 100% (485/485), done.
remote: Total 6834 (delta 675), reused 742 (delta 578), pack-reused 5751
Receiving objects: 100% (6834/6834), 13.21 MiB | 1.14 MiB/s, done.
Resolving deltas: 100% (3944/3944), done.
```

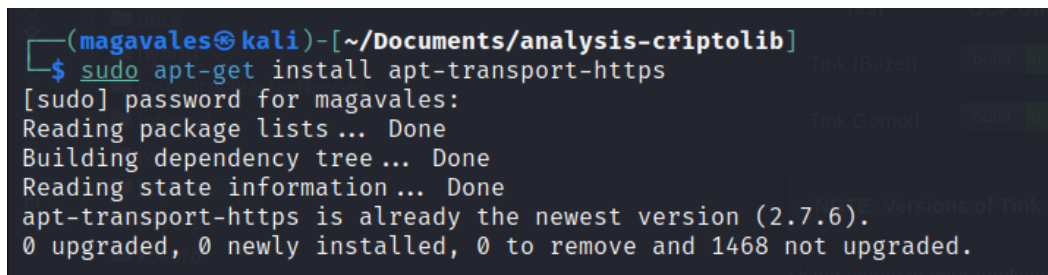
Рисунок 1. Скачивание проекта Circl



```
(magavales@kali)-[~/Documents/analysis-criptolib]
$ git clone https://github.com/elizarpif/cryptoswitch
Cloning into 'cryptoswitch' ...
remote: Enumerating objects: 210, done.
remote: Counting objects: 100% (210/210), done.
remote: Compressing objects: 100% (130/130), done.
remote: Total 210 (delta 129), reused 142 (delta 71), pack-reused 0
Receiving objects: 100% (210/210), 504.07 KiB | 1.41 MiB/s, done.
Resolving deltas: 100% (129/129), done.
```

Рисунок 2. Скачивание проекта Cryptoswitch

2. Установка анализаторов Bearer и Semgrep.



```
(magavales@kali)-[~/Documents/analysis-criptolib]
$ sudo apt-get install apt-transport-https
[sudo] password for magavales:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
apt-transport-https is already the newest version (2.7.6).
0 upgraded, 0 newly installed, 0 to remove and 1468 not upgraded.
```

Рисунок 3. Установка анализатора Bearer ч.1



```
(magavales@kali)-[~/Documents/analysis-criptolib]
$ echo "deb [trusted=yes] https://apt.fury.io/bearer/ /" | sudo tee -a /etc/apt/sources.list.d/fury.list
deb [trusted=yes] https://apt.fury.io/bearer/ /
```

Рисунок 4. Установка анализатора Bearer ч.2

```
(magavales@kali)-[~/Documents/analysis-criptolib]
$ sudo apt-get install bearer
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  bearer
0 upgraded, 1 newly installed, 0 to remove and 1649 not upgraded.
Need to get 14.6 MB of archives.
After this operation, 53.3 MB of additional disk space will be used.
Get:1 https://apt.fury.io/bearer/ bearer 1.34.0 [14.6 MB]
Fetched 14.6 MB in 6s (2,382 kB/s)
Selecting previously unselected package bearer.
(Reading database ... 409390 files and directories currently installed.)
Preparing to unpack .../bearer_1.34.0_amd64.deb ...
Unpacking bearer (1.34.0) ...
Setting up bearer (1.34.0) ...
Processing triggers for kali-menu (2023.1.7) ...
Scanning processes...
Scanning candidates...
Scanning linux images...

Running kernel seems to be up-to-date.

Restarting services...
Service restarts being deferred:
systemctl restart NetworkManager.service
systemctl restart lightdm.service

No containers need to be restarted.

User sessions running outdated binaries:
magavales @ session #2: qterminal[22989], xfce4-panel[1292], xfce4-session[1102]
magavales @ user manager service: gvfsd[1251], systemd[1065]

No VM guests are running outdated hypervisor (qemu) binaries on this host.
```

Рисунок 5. Установка анализатора Bearer ч.3

```
(magavales@kali)-[~/Documents/GoLand-2023.1.1/bin]
$ brew install semgrep
Running 'brew update --auto-update' ...
Auto-updated Homebrew!
Updated 3 taps (snik/tap, homebrew/core and homebrew/cask).
New Formulae
rathole
New Casks
aqua
markedit
You have 3 outdated formulae installed.
semgrep 1.55.2 is already installed but outdated (so it will be upgraded).
Downloading https://ghcr.io/v2/homebrew/core/semgrep/manifests/1.56.0
##### 100.0%
Fetching semgrep
Downloading https://ghcr.io/v2/homebrew/core/semgrep/blobs/sha256:b676c0c4b8776fbc87a5e4aa6685cfd63969435b2241311a63a63289dbf5aa6d
##### 17.4%
```

Рисунок 6. Установка анализатора Semgrep

Выполнение работы

Библиотека Circl

Анализатор Semgrep

1. Запускаем статический анализатор:

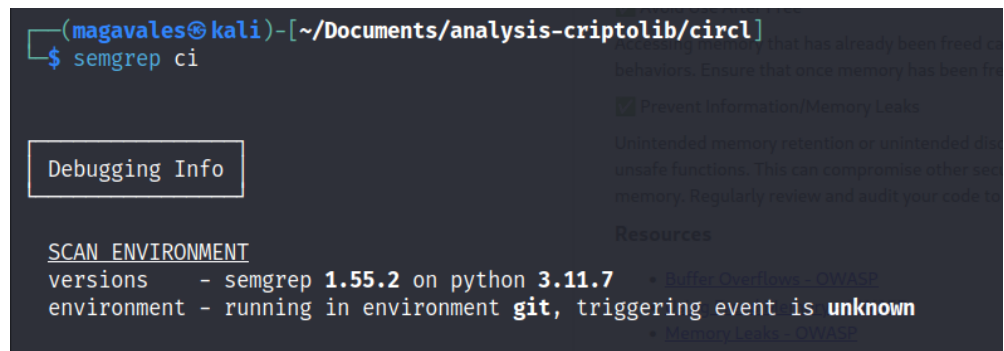


Рисунок 7. Работа анализатора Semgrep

2. Рассмотрим полученные результаты:

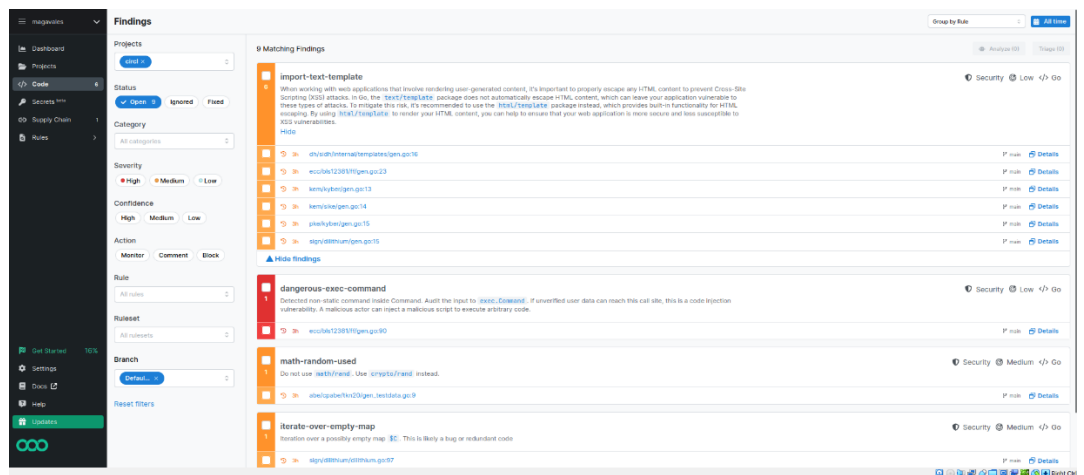


Рисунок 8. Полученные результаты от Semgrep

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

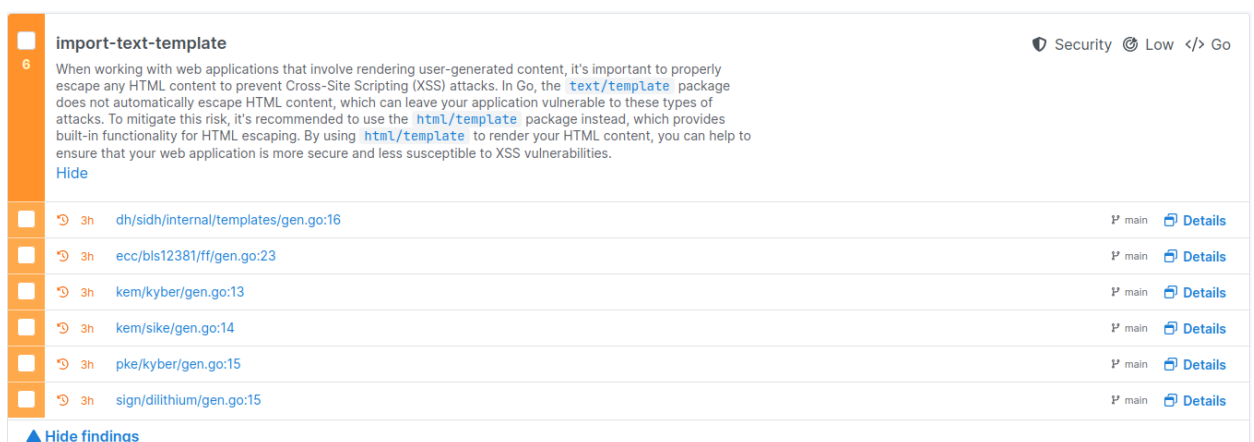


Рисунок 9. Ошибка, связанная с импортом и использование библиотеки

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.

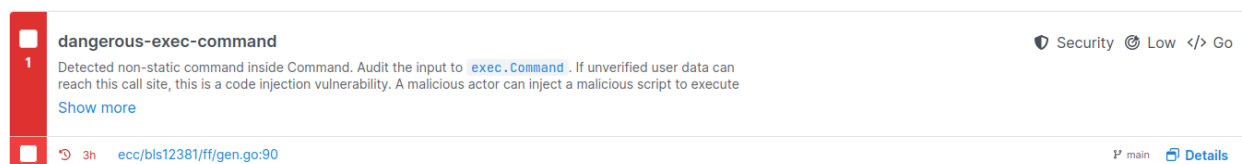


Рисунок 10. Уязвимость класса инъекция

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 рисунке представлена детальная ситуация по использованию введенных пользователем данных.

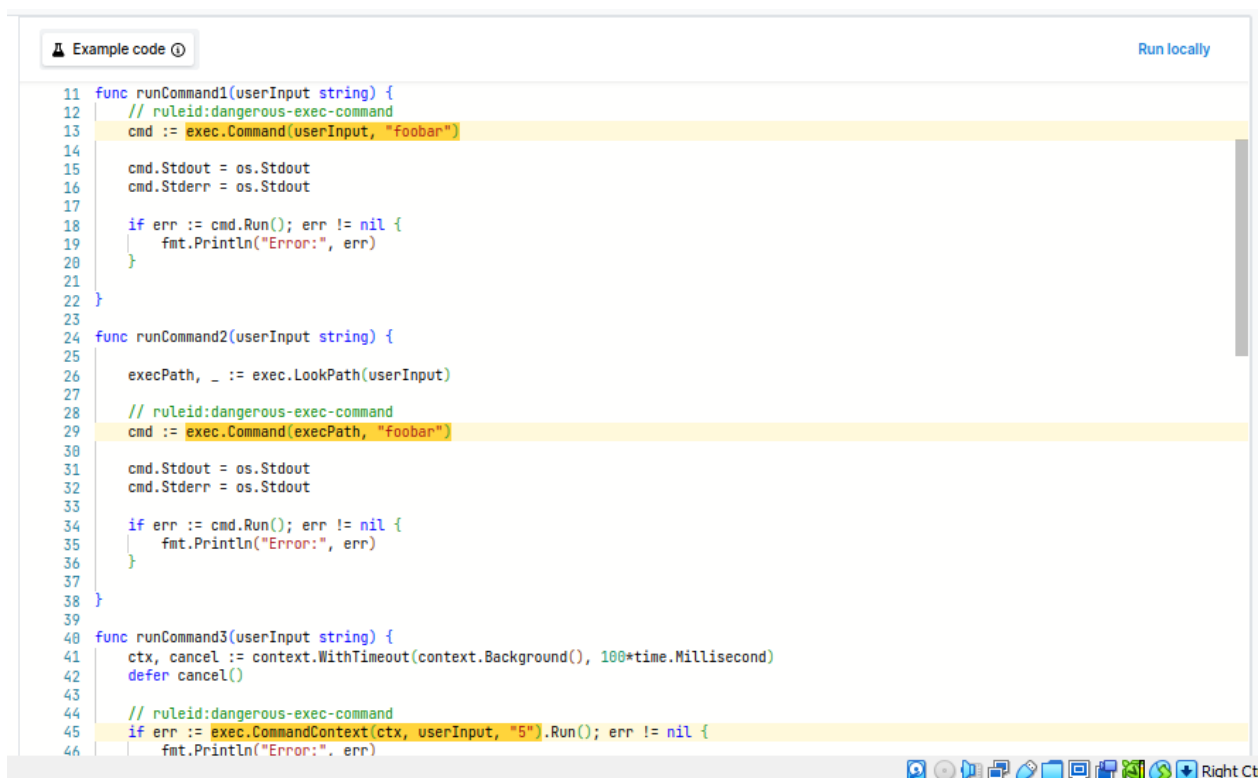


Рисунок 11. Детали отчета по уязвимости класса инъекция

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](#). 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 использованием 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*.

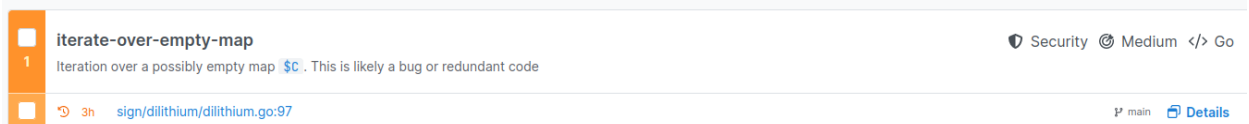


Рисунок 13. Возможное итерирование по пустой *map*

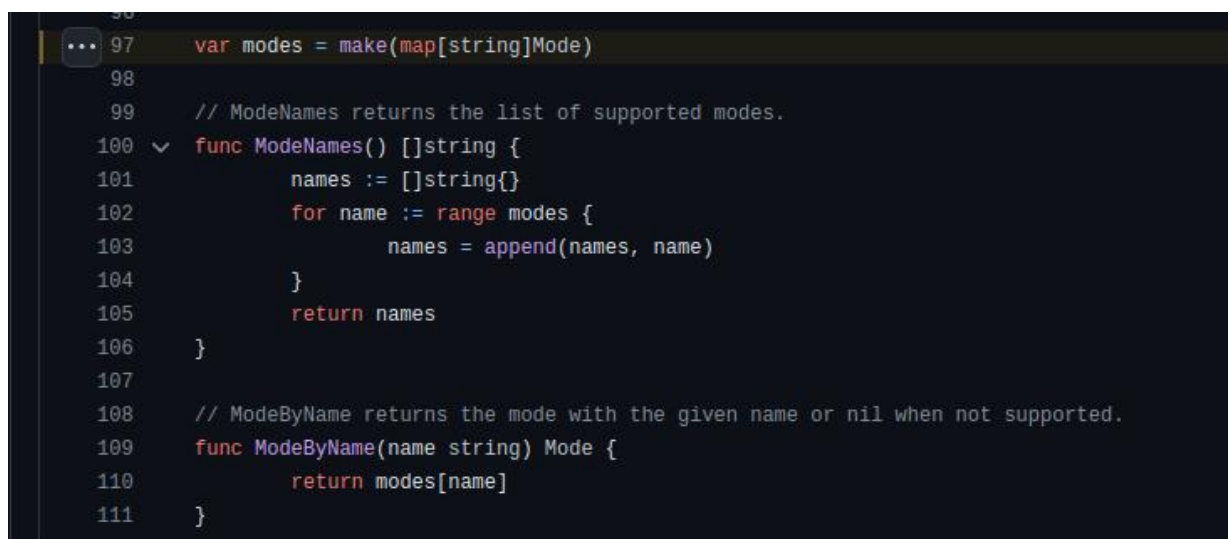


Рисунок 14. Фрагмент кода из библиотеки Circl

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:

```
(magavales@kali)-[~/Documents/analysis-criptolib]
$ bearer scan circl --format html | tee output-circl.html
Analyzing codebase
  (122281/-) [0s]
Loading rules
Scanning target circl
  100% [=====] (609/609) [44s]
Running Detectors
Generating dataflow
Evaluating rules
  100% [=====] (102/102) [1s]
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>Bearer - Security Report</title>
    <style>
      body {
        margin:0;
```

Рисунок 15. Работа анализатора Bearer

2. Рассмотрим полученные результаты в ходе работы анализатора Bearer:

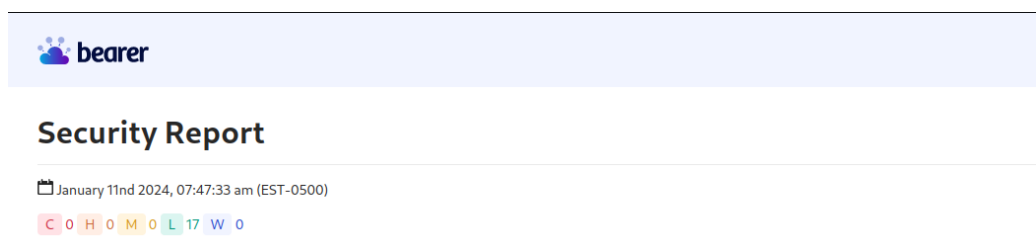


Рисунок 16. Результаты работы анализатора Bearer

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

Use of inherently dangerous function (unsafe package) low

Rule ID: go_gosec_unsafe_unsafe CWE: CWE 242 Fingerprint: ef55de70a61a8f4dee6b4dfadf122be2_0

Filename: ecc/fourq/point_amd64.go:9

```
9      _x      = unsafe.Offsetof(pointR1{}).X)
```

Description

The Go programming language features the `unsafe` package which grants low-level memory management capabilities, inclusive of direct memory access and pointer manipulation. Though the `unsafe` package can be quite potent, its usage sidesteps the Go compiler's type safety checks. This can lead to an array of security vulnerabilities and unpredictable system behavior.

Remediations

✓ Avoid `unsafe` Unless Absolutely Necessary

The overarching guidance here is to steer clear of the `unsafe` package unless there's an absolute necessity for its functions. When opting for low-level memory operations, ensure that their implications are well-understood and that their deployment is preceded by rigorous testing.

✓ Be Wary of Buffer Overflows

Direct manipulation of memory can lead to buffer overflows, potentially enabling unauthorized code execution. Ensure buffer boundaries are always respected.

✓ Avoid Use After Free

Accessing memory that has already been freed can result in unintended code execution or unpredictable behaviors. Ensure that once memory has been freed, it isn't accessed further.

✓ Prevent Information/Memory Leaks

Unintended memory retention or unintended disclosure of information in memory can occur when using unsafe functions. This can compromise other security defenses or lead to system failures due to exhausted memory. Regularly review and audit your code to check for such leaks.

Resources

- [Buffer Overflows - OWASP](#)
- [Using Freed Memory - OWASP](#)
- [Memory Leaks - OWASP](#)

Рисунок 17. Использование пакета "unsafe" ч.1

- [Buffer Overflows - OWASP](#)
- [Using Freed Memory - OWASP](#)
- [Memory Leaks - OWASP](#)

Use of inherently dangerous function (unsafe package) low

Rule ID: go_gosec_unsafe_unsafe CWE: CWE 242 Fingerprint: ef55de70a61a8f4dee6b4dfadf122be2_4

Filename: ecc/fourq/point_amd64.go:13

```
13      _tb      = unsafe.Offsetof(pointR1{}).Tb)
```

Description

The Go programming language features the `unsafe` package which grants low-level memory management capabilities, inclusive of direct memory access and pointer manipulation. Though the `unsafe` package can be quite potent, its usage sidesteps the Go compiler's type safety checks. This can lead to an array of security vulnerabilities and unpredictable system behavior.

Remediations

✓ Avoid `unsafe` Unless Absolutely Necessary

The overarching guidance here is to steer clear of the `unsafe` package unless there's an absolute necessity for its functions. When opting for low-level memory operations, ensure that their implications are well-understood and that their deployment is preceded by rigorous testing.

✓ Be Wary of Buffer Overflows

Direct manipulation of memory can lead to buffer overflows, potentially enabling unauthorized code execution. Ensure buffer boundaries are always respected.

✓ Avoid Use After Free

Accessing memory that has already been freed can result in unintended code execution or unpredictable behaviors. Ensure that once memory has been freed, it isn't accessed further.

✓ Prevent Information/Memory Leaks

Unintended memory retention or unintended disclosure of information in memory can occur when using `unsafe` functions. This can compromise other security defenses or lead to system failures due to exhausted memory. Regularly review and audit your code to check for such leaks.

Resources

- [Buffer Overflows - OWASP](#)
- [Using Freed Memory - OWASP](#)
- [Memory Leaks - OWASP](#)

Use of inherently dangerous function (unsafe package) low

Rule ID: go_gosec_unsafe_unsafe CWE: CWE 242 Fingerprint: ef55de70a61a8f4dee6b4dfadf122be2_5

Filename: ecc/fourq/point_amd64.go:14

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 позволяет 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:

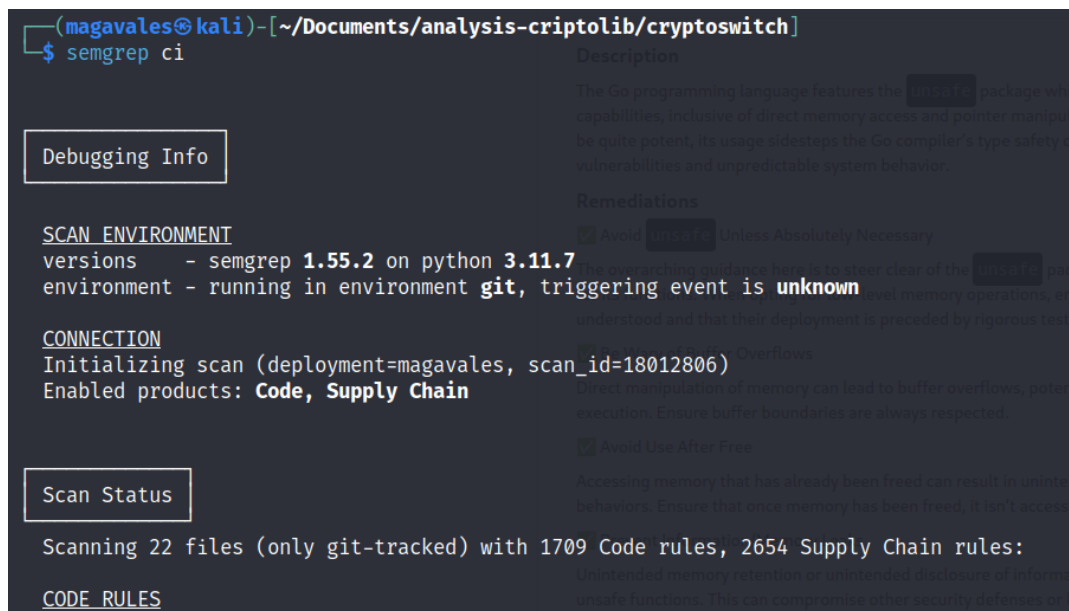


Рисунок 19. Работа анализатора Semgrep

2. Рассмотрим полученные результаты:

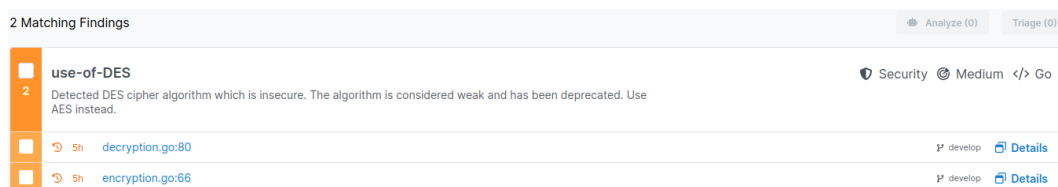


Рисунок 20. Полученные результаты от Semgrep

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

2. Рассмотрим полученные результаты:



Рисунок 23. Результаты полученные от Bearer

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

Use of a Broken or Risky Cryptographic Algorithm low

Rule ID: go_gosec_blocklist_des CWE: CWE 327 Fingerprint: 33b1cb607d3338d26c5bba57d324270e_0

Filename: decryption.go:7

7 "crypto/des"

Description

The Data Encryption Standard (DES) is an outdated symmetric-key algorithm for encryption. Officially deemed insecure and no longer recommended for use, DES was withdrawn by the National Institute of Standards and Technology (NIST) as a standard in 2005, primarily because of its 56-bit key size, which is vulnerable to brute-force attacks.

Remediation

To ensure the confidentiality and integrity of sensitive data, it is crucial to utilize a modern and secure encryption algorithm. The use of Advanced Encryption Standard (AES) with a key size of 256 bits (AES-256) is recommended for its strong security properties and widespread acceptance as a replacement for DES.

✅ Implement AES-256 for Strong Encryption

```
// Use AES-256 for secure encryption by initializing a 32-byte key for AES-256
key := make([]byte, 32)
if , err := io.ReadFull(rand.Reader, key); err != nil {
    log.Fatal(err)
}

// Create a new cipher block from the key
blockCipher, err := aes.NewCipher(key)
if err != nil {
    log.Fatal(err)
}

// Use Galois/Counter Mode (GCM) for both encryption and decryption
aad, err := cipher.NewGCM(blockCipher)
if err != nil {
    log.Fatal(err)
}

var encrypted = []byte{}
var nonce = []byte{}

// Encrypt a message with AES-256 using GCM
msg := []byte("Some secret message")
// Ensure nonces are unique for each encryption to maintain security
nonce = make([]byte, 12)
if , err := io.ReadFull(rand.Reader, nonce); err != nil {
    log.Fatal(err)
}
encrypted = aad.Seal(nil, nonce, msg, nil)

// Decrypt the message securely
msg, err := aad.Open(nil, nonce, encrypted, nil)
if err != nil {
    log.Fatal(err)
}
fmt.Printf("Decrypted: %s\n", msg)
```

❌ Do Not Use Deprecated Algorithms Avoid using deprecated cryptographic algorithms such as DES, as they do not provide adequate security against modern threats and attacks.

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>