Статический анализ программ на языке Go

Оригинальный репозиторий
https://meowgit.nekoea.red/nihonium/linux-auth

Ветка для SAST
https://meowgit.nekoea.red/nihonium/linux-auth/src/branch/sast

Основная информация по инструментам анализа из статьи
ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ ДЛЯ СТАТИЧЕСКОГО АНАЛИЗА КОДА GOLANG, Р. М. Галиев, О. И. Евдошенко (https://cyberleninka.ru/article/n/obzor-suschestvuyuschih-resheniy-dlya-staticheskogo-analiza-koda-golang/pdf)

Кроме svace в статье рассмотрены следующие анализаторы:

Анализатор	Количество срабатываний	Вызывает те же бинарные файлы других анализаторов (да/нет)	Пригоден для дальнейшего исследования (да/нет)
bearer	52	Нет	Да
scanmycode_ce (betterscan.io)	2	Нет	Нет
gokart	3	Нет	Нет
golangcilint	851	Нет	Да
goreporter	72	Да, вызывает те же анализаторы, что и golangcilint	Нет
gometalinter	57	Да, вызывает те же анализаторы, что и golangcilint	Нет

Авторы решили исключить 2-й и 3-й анализаторы по причине малого количества срабатываний, 5-й и 6-й из-за того, что они входят в 4-й. В итоге остались анализаторы bearer и golangci-lint. У меня они успешно заработали. В качестве третьего анализатора я сначала попробовал gokart, но он устарел и не поддерживает новые версии Go (в том числе исследуемый проект). Поэтому после поиска информации в интернете я добавил анализатор gosec. Все 3 анализатора устанавливаются через скачивание бинарного файла и поддерживают формат sarif. CI/CD не использовал, но тоже должно работать.

Результаты bearer

HIGH: Unsanitized user input in file path [CWE-73]
    https://docs.bearer.com/reference/rules/go_gosec_filesystem_filereadtaint
    To ignore this finding, run: bearer ignore add 690cb9207bb6cb72edd1002fae0a0fa3_0

    File: internal/config/config.go:41

     41 data, err := os.ReadFile(path)

LOW: Leakage of information in logger message [CWE-532]
    https://docs.bearer.com/reference/rules/go_lang_logger_leak
    To ignore this finding, run: bearer ignore add 219087ffdfad090e6436320f68eae990_0

    File: cmd/add_user/main.go:33

     33   log.Fatalf("Ошибка инициализации БД: %v\n", err)

И ещё 4 аналогичных срабатывания на log.Fatalf

Результаты golangci-lint

```
internal/ui/console.go:62:3: QF1003: could use tagged switch on resp (staticcheck)
                if resp == "y" || resp == "yes" {
                ^
1 issues:
* staticcheck: 1
```

Результаты gosec

[/home/vovuas/linux-auth/internal/config/config.go:41] - G304 (CWE-22): Potential file inclusion via variable (Confidence: HIGH, Severity: MEDIUM)

```
    40: func Load(path string) (*Config, error) {
  > 41:     data, err := os.ReadFile(path)
    42:     if err != nil {
```

Autofix: Consider using os.Root to scope file access under a fixed root (Go >=1.24). Prefer root.Open/root.Stat over os.Open/os.Stat to prevent directory traversal.

Итоги:

- у bearer и gosec есть срабатывание на os.ReadFile с немного разной интерпретацией проблемы, gosec предложил исправление; возможно, стоит это сделать (как я понял, при запуске приложения можно указать путь до файла с конфигурацией)

- bearer ещё указывает на потенциальную утечку информации в логах ошибок; возможно, стоит оставить подробную информацию только в debug версии приложения (сейчас единственное отличие сборок состоит в флагах go build, которые, как я понял, выключают оптимизации компилятора)

- у golangci-lint единственное и уникальное срабатывание (по-моему false positive)

Авторы статьи пишут, что golangi-lint находит много простых ошибок, bearer хорош в работе с ОС, а svace должен находить уязвимости в потоке данных (в статье есть 2 примера программ, специальные ошибки в которых не нашлись с использованием рассмотренных анализаторов).

Лично мне понравилось, что gosec предлагает autofix.