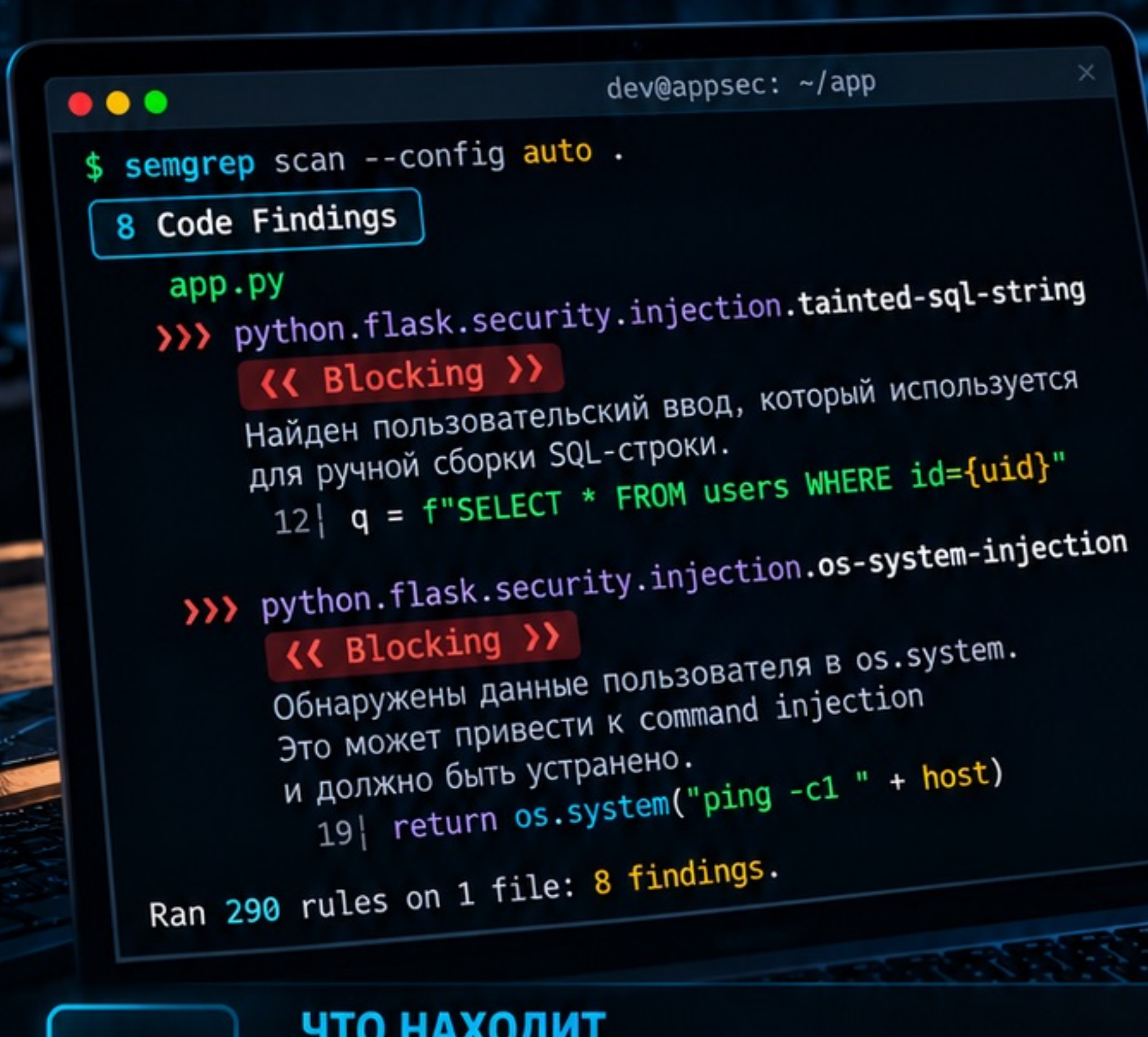
Semgrep

01/07

Ищет опасные паттерны и taint в коде более чем на 30 языках

SAST | Pattern Matching | Taint Analysis | 30+ Languages



ЧТО НАХОДИТ

Уязвимости, такие как SQL и command injection, XSS, небезопасную работу с криптографией, жёстко захардкоженные credentials и другие риски.

ПОЧЕМУ ЭТО РАБОТАЕТ

Использует правила, похожие на ваш код.
Можно быстро писать свои правила под конкретные риски — это занимает минуты, а не часы.

КАК ЗАПУСТИТЬ

`pip install semgrep` затем

```
semgrep scan --config auto .
```

30+ languages

Free Community Edition

Rules in YAML

IDE + CI Integration

@lionsoulbatyshchikov

SonarQube

Бесплатный self-hosted сервер для анализа кода сообщества

02/07

Code Quality | Security | 20+ Languages | Open Source



ЧТО НАХОДИТ

Уязвимости, security issues и потенциальные проблемы в коде более чем на **20 языках**.
Каждое правило сопровождается понятным описанием.

ЗАЧЕМ ИСПОЛЬЗОВАТЬ

Дашборд уже используют большинство команд.
Quality gate помогает не пропустить новые **security issues** при каждом изменении кода.

ПЛЮС

Бесплатная **community-версия** анализирует одну ветку без **taint analysis**. Поэтому для глубокой проверки на уязвимости (например, **injection flows**) потребуется платная версия.

20+ languages | Self-hosted | LGPL-3.0 | Quality gate in CI

@lionsoulbatyshchikov

03/07

CodeQL

Движок для семантического анализа кода от GitHub

Code Scanning | Security | Open Source | CLI

```
dev@appsec : ~/app

$ codeql database create cqdb --language=python
Successfully created database at /home/dev/app/cqdb.
$ codeql database analyze cqdb --format=csv --output=results.csv
Running queries...
Starting evaluation of codeql/python-queries/Security/CWE-
078/CommandInjection.ql..
[11/45 eval 831ms] Evaluation done; writing results to codeql/python-
queries/Security/CWE-078/CommandInjection.bqrs..
Starting evaluation of codeql/python-queries/Security/CWE-
089/SqlInjection.ql..
[13/45 eval 676ms] Evaluation done; writing results to codeql/python-
queries/Security/CWE-089/SqlInjection.bqrs..
[45/45 eval 148ms] Evaluation done; writing results to codeql/python-
queries/Summary/LinesOfUserCode. bqrs..
Interpreting results.
$ cut -d, -f1,3,5,6 results.csv
"SQL query built from user-controlled sources","error","/app.py","13"
"Uncontrolled command line","error","/app.py","19"
```

ЧТО НАХОДИТ

Уязвимости, такие как **injection**, **SSRF**, **path traversal**, небезопасная **deserialization** и другие проблемы, отслеживая **data flows** между файлами.

КАК ЭТО РАБОТАЕТ

Построение базы данных кода и запуск **queries**, которые анализируют код с учётом **контекста** и **семантики**, а не простого поиска по строкам.

ГДЕ ИСПОЛЬЗУЕТСЯ

Интегрирован в **GitHub code scanning**.
Результаты приходят в **pull request** в виде алертов, а не в отдельный дашборд.

Для **public** репозиториев
Бесплатно

Open Source
CLI в комплекте

Поддержка
20+ языков

Для **private** репозиториев
Нужен
GitHub Code Security

@lionsoulbatyshchikov

Bandit

04/07

Стандартный security linter для Python

```
dev@appsec : ~/app

$ bandit -r .
Test results:
>> Issue: [B608: hardcoded_sql_expressions] Possible SQL injection
vector through string-based query construction.
   Severity: Medium   Confidence: Low
   CWE: CWE-89 (https://cwe.mitre.org/data/definitions/89.html)
   Location: ./app.py:13:8
13        q = "SELECT * FROM users WHERE id=" + uid
--------------------------------------------------
>> Issue: [B605:start_process_with_a_shell] Starting a process with
a shell, possible injection detected, security issue.
   Severity: High    Confidence: High
   CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
   Location: ./app.py:19:11
19        return os.system("ping -c1 " + request.args.get("host"))
--------------------------------------------------
Run metrics:
    Total issues (by severity):
        Low: 1   Medium: 3   High: 2
```

SECURE CODE

ЧТО НАХОДИТ

Уязвимости, такие как **shell injection**, небезопасное формирование **SQL**-строк, **unsafe pickle**, небезопасные **yaml loads**, слабые хеши и **hardcoded passwords**.

ПОЧЕМУ ИСПОЛЬЗОВАТЬ

Запускается в пару команд и быстро выявляет проблемы в Python-коде. Каждая находка имеет **CWE**, уровень **severity** и объяснение, поэтому её можно сразу добавить в **CI**.

КАК ЗАПУСТИТЬ

`pip install bandit` затем

```
bandit -r .
```

Python only

PyCQA project

CWE-mapped findings

pre-commit ready

@lionsoulbatyshchikov

gosec

05/07

Проверка безопасности кода на Go

SAST · Go · Security Linter · Open Source

```
dev@appsec : ~/goapp

$ gosec ./...
Results:
[/home/dev/goapp/main.go:14] - G701 (CWE-89): SQL injection via taint
analysis (Confidence: HIGH, Severity: HIGH)
    13:     id := r.URL.Query().Get("id")
  > 14:     rows, := db.Query("SELECT * FROM users WHERE id=" + id)
    15:     fmt.Println(w, rows)

[/home/dev/goapp/main.go:20] - G702 (CWE-78): Command injection via
taint analysis (Confidence: HIGH, Severity: HIGH)
    19:     host := r.URL.Query().Get("host")
  > 20:     out, := exec.Command("sh", "-c", "ping " + host).Output()
    21:     w.Write(out)

Summary:
   Gosec   : 2.29.0
   Files   : 1
   Issues : 9
```

ЧТО НАХОДИТ

Уязвимости, такие как **injection** (G701, G702), небезопасное использование криптографии, жёстко заданные **credentials**, небезопасные права на файлы и другие проблемы в **Go**-коде.

ПОЧЕМУ ИСПОЛЬЗОВАТЬ

Работает на собственном **AST** и **type information** для **Go**, выявляет реальные **security**-риски ещё на этапе разработки. Запускается как **linter** в **golangci-lint**.

КАК ЗАПУСТИТЬ

Через Brew: `brew install gosec`

Или через Go: `go install` , затем `gosec ./...`

- Только для Go
- Правила taint analysis
- Интеграция с golangci-lint
- Вывод в SARIF + JSON

@lionsoulbatyshchikov

06/07

Brakeman

Статический анализ безопасности для Ruby on Rails приложений

Security Scanner | Ruby on Rails | Static Analysis | CI Ready

```
dev@appsec : ~/railsapp

$ brakeman --no-color -q
== Brakeman Report ===
Rails Version: 8.0
Brakeman Version: 8.0.6
== Overview ==
Security Warnings: 5
== Warnings ==
Confidence: High
Category: Command Injection
Message: Possible command injection
Code: `ping -c1 #{params[:host]}`
File: app/controllers/users_controller.rb
Line: 7

Confidence: Medium
Category: SQL Injection
Message: Possible SQL injection
Code: User.where("name = '#{params[:name]}'")
File: app/controllers/users_controller.rb
Line: 3
```

ЧТО НАХОДИТ

Уязвимости, такие как **SQL** и **command injection**, **open redirects**, **mass assignment**, **небезопасные render calls** и отсутствие **CSRF protection**.

ПОЧЕМУ ИСПОЛЬЗОВАТЬ

Не требует запущенного приложения, базы данных или конфигурации. Читает исходный код **Rails** напрямую, поэтому сканирование занимает всего **несколько секунд**.

КАК ЗАПУСТИТЬ

```
gem install brakeman
```

затем

```
brakeman .
```

- Только для Rails
- Zero configuration
- Поддерживается с 2010
- Работает в CI с флагом -z

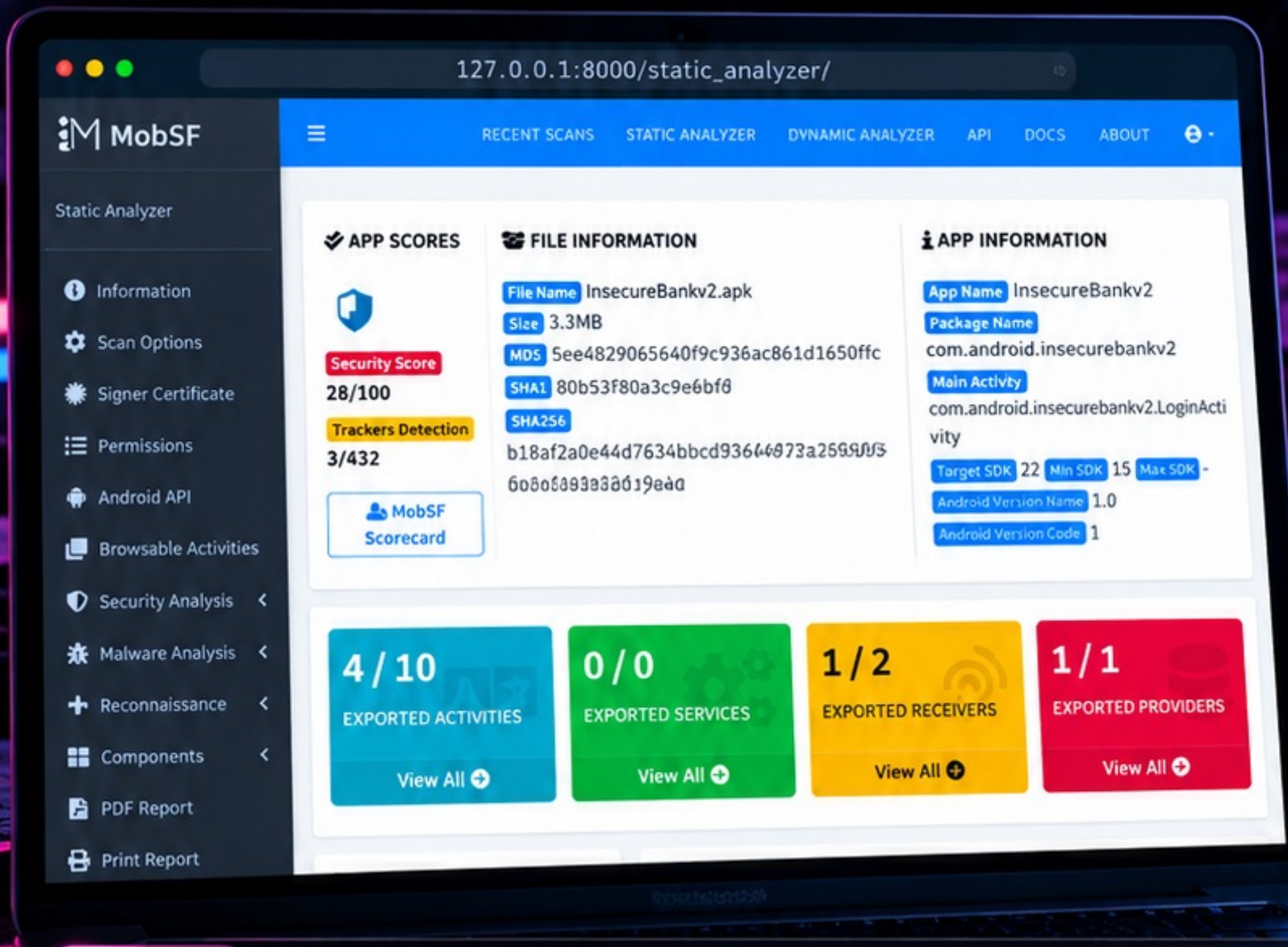
@lionsoulbatyshchikov

07/07

MobSF

Фреймворк для анализа безопасности мобильных приложений Android и iOS

Static Analysis | Dynamic Analysis | Android + iOS | Open Source



ЧТО НАХОДИТ

Экспортированные компоненты, потенциально опасные **manifest flags**, захардкоженные секреты, трекеры и общий **security score** для каждого приложения.

ЗАЧЕМ ИСПОЛЬЗОВАТЬ

Загружаешь **APK** или **IPA** и получаешь **полный отчёт** по безопасности за несколько минут — без доступа к исходному коду.

КАК ЗАПУСТИТЬ

```
docker run -p 8000:8000
opensecurity/mobile-security-framework-mobsf
```

Android + iOS | Static + dynamic analysis | REST API для CI | GPL-3.0 Open Source

@lionsoulbatyshchikov