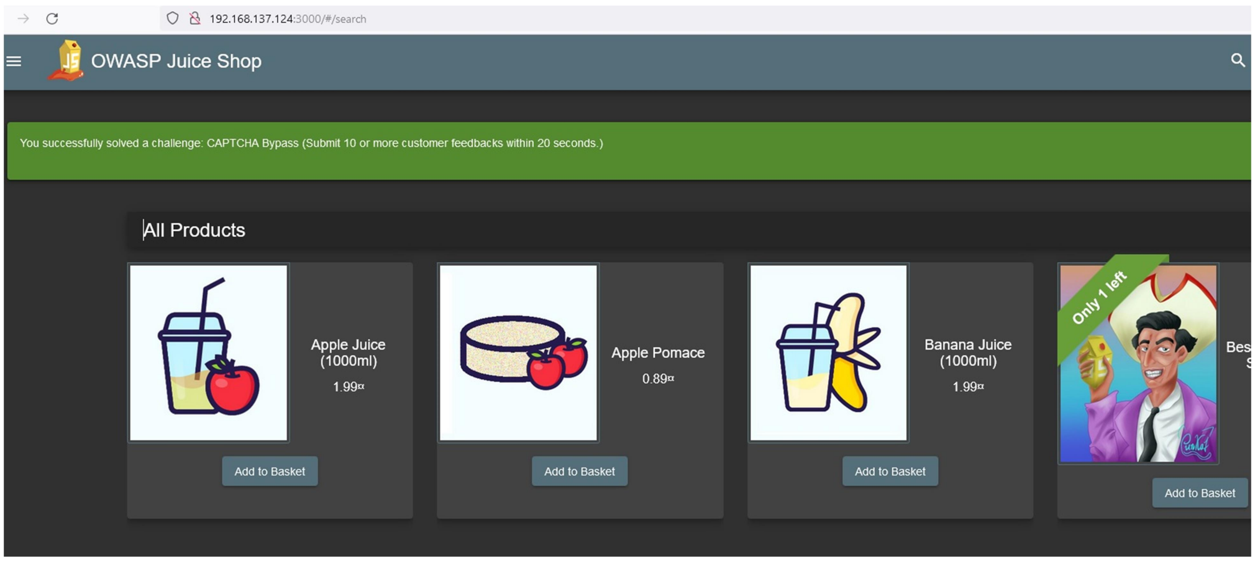
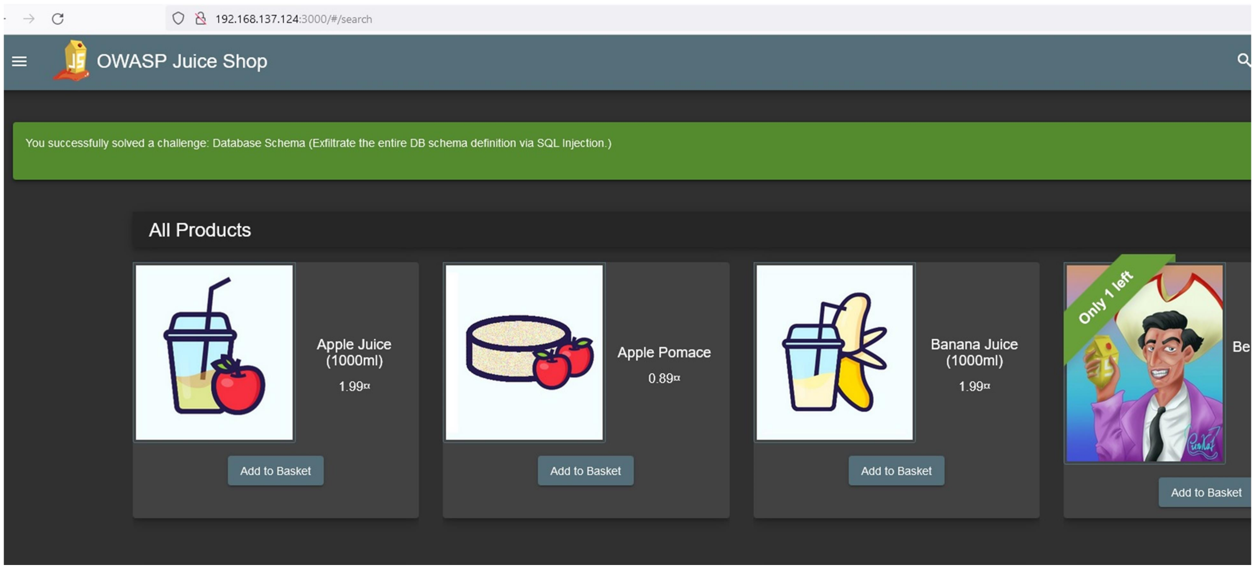
1. Скриншот выполненной работы CAPTCHA Bypass из таблицы Scoreboard Juice Shop.



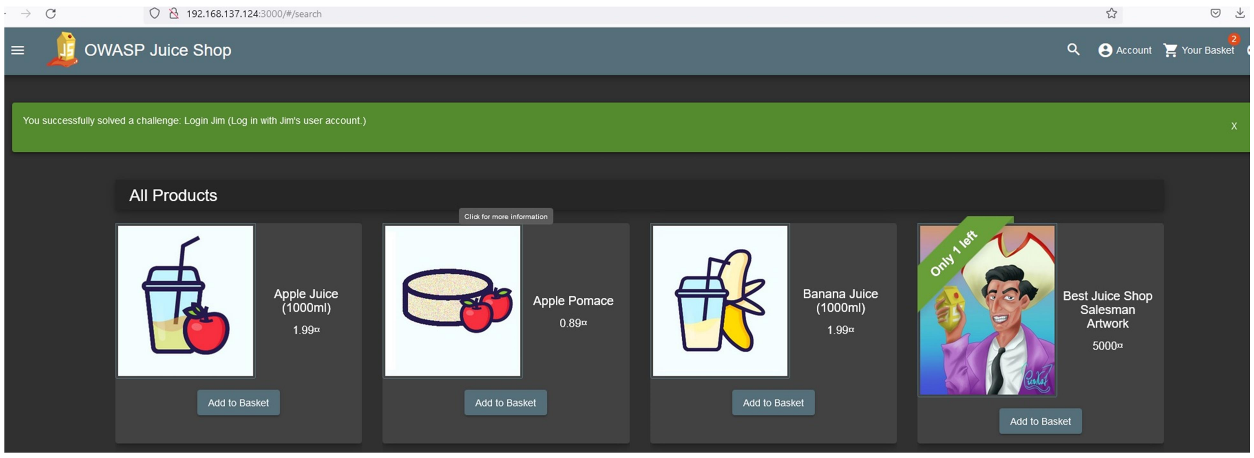
You successfully solved a challenge: CAPTCHA Bypass (Submit 10 or more customer feedbacks within 20 seconds.)

2. Скриншот выполненной работы Database Schema из таблицы Scoreboard Juice Shop.



You successfully solved a challenge: Database Schema (Exfiltrate the entire DB schema definition via SQL Injection.)

3. Скриншот выполненной работы Login Jim из таблицы Scoreboard Juice Shop


192.168.137.124:3000/#/search
OWASP Juice Shop
Account
Your Basket
2

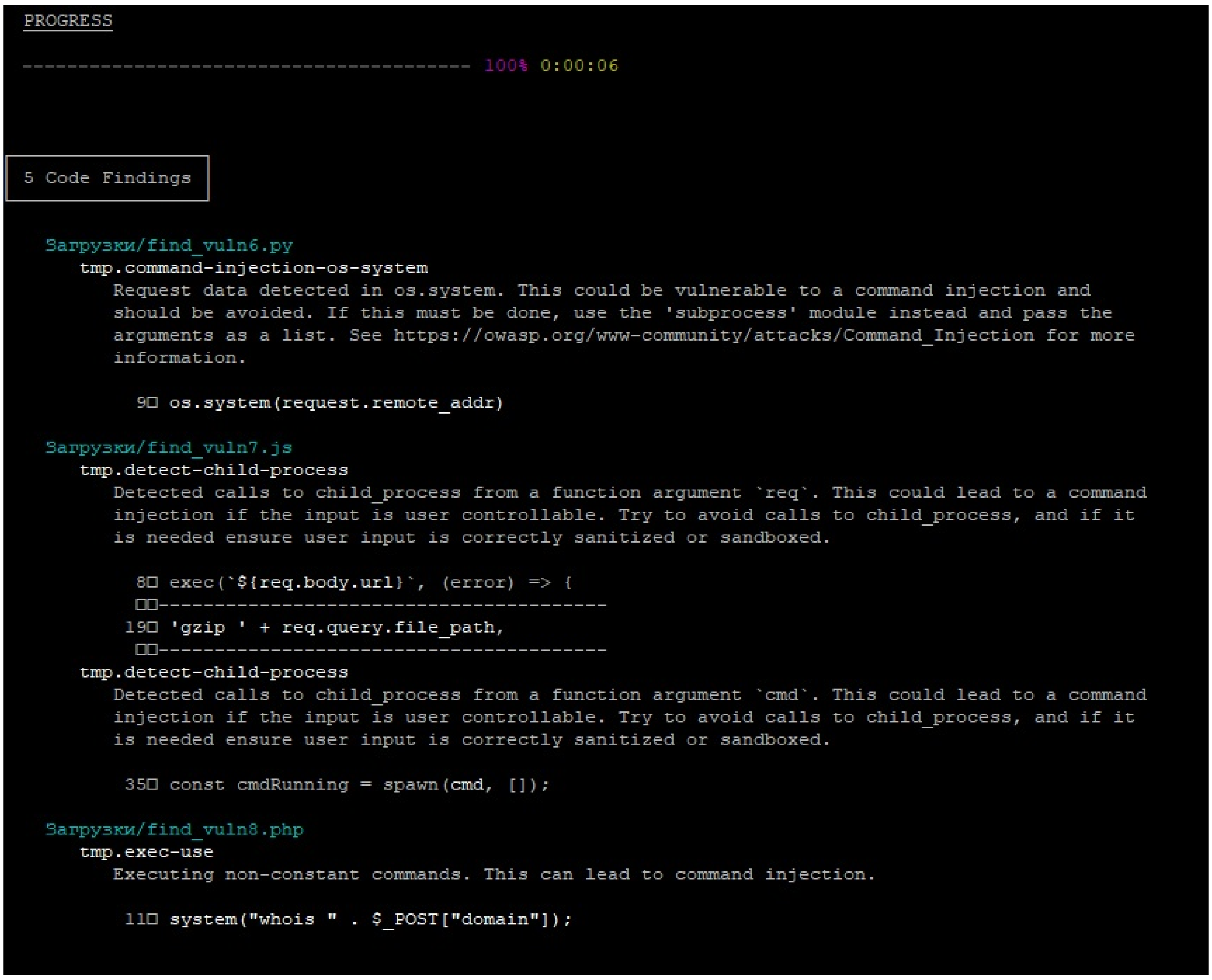
You successfully solved a challenge: Login Jim (Log in with Jim's user account.)

X
All Products
Click for more information
Apple Juice (1000ml) 1.99¤
Add to Basket
Apple Pomace 0.89¤
Add to Basket
Banana Juice (1000ml) 1.99¤
Add to Basket
Only 1 left
Best Juice Shop Salesman Artwork 5000¤
Add to Basket

А также:

Команды для сканирования файлов на наличие уязвимостей.

semgrep scan --config="/tmp/command-injection-os-system.yaml" --config="/tmp/exec-use.yaml" --config="/tmp/detect-child-process.yaml"



```
 PROGRESS

 ---------------------------------------- 100% 0:00:06


 5 Code Findings

   Загрузки/find_vuln6.py
      tmp.command-injection-os-system
         Request data detected in os.system. This could be vulnerable to a command injection and
         should be avoided. If this must be done, use the 'subprocess' module instead and pass the
         arguments as a list. See https://owasp.org/www-community/attacks/Command_Injection for more
         information.

           9┆ os.system(request.remote_addr)

   Загрузки/find_vuln7.js
      tmp.detect-child-process
         Detected calls to child_process from a function argument `req`. This could lead to a command
         injection if the input is user controllable. Try to avoid calls to child_process, and if it
         is needed ensure user input is correctly sanitized or sandboxed.

           8┆ exec(`${req.body.url}`, (error) => {
           ⋮┆----------------------------------------
          19┆ 'gzip ' + req.query.file_path,
           ⋮┆----------------------------------------
      tmp.detect-child-process
         Detected calls to child_process from a function argument `cmd`. This could lead to a command
         injection if the input is user controllable. Try to avoid calls to child_process, and if it
         is needed ensure user input is correctly sanitized or sandboxed.

          35┆ const cmdRunning = spawn(cmd, []);

   Загрузки/find_vuln8.php
      tmp.exec-use
         Executing non-constant commands. This can lead to command injection.

          11┆ system("whois " . $_POST["domain"]);
```

Уязвимости:

find_vuln6.py –уязвимость command-injection-os-system (9 строка)

```
semgrep --config=" r/python.django.security.injection.command.command-injection-os-system.command-injection-os-system"
```

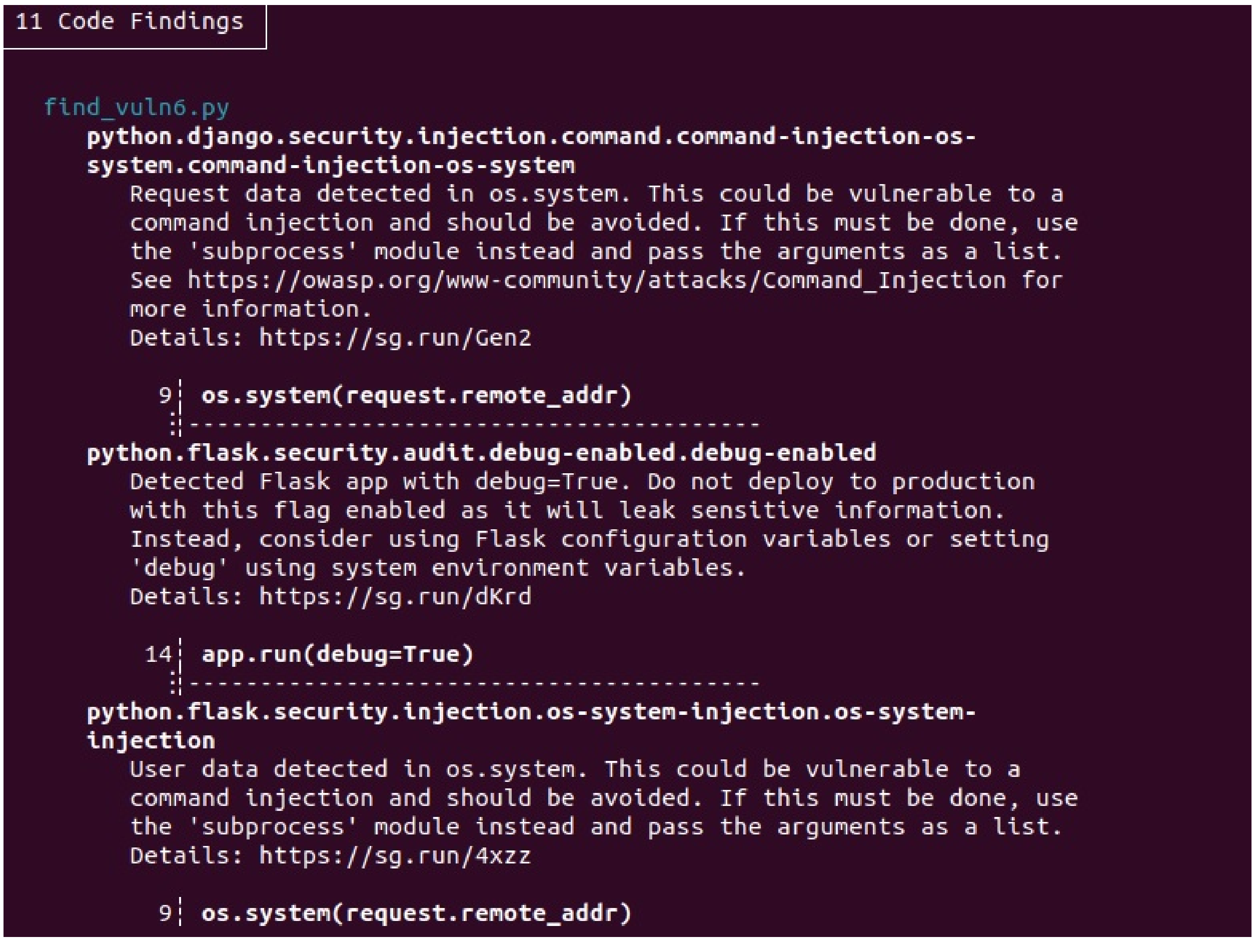
find_vuln7.js – уязвимость express-child-process (8, 19 строка)

```
semgrep --config="r/javascript.lang.security.detect-child-process.detect-child-process"
```

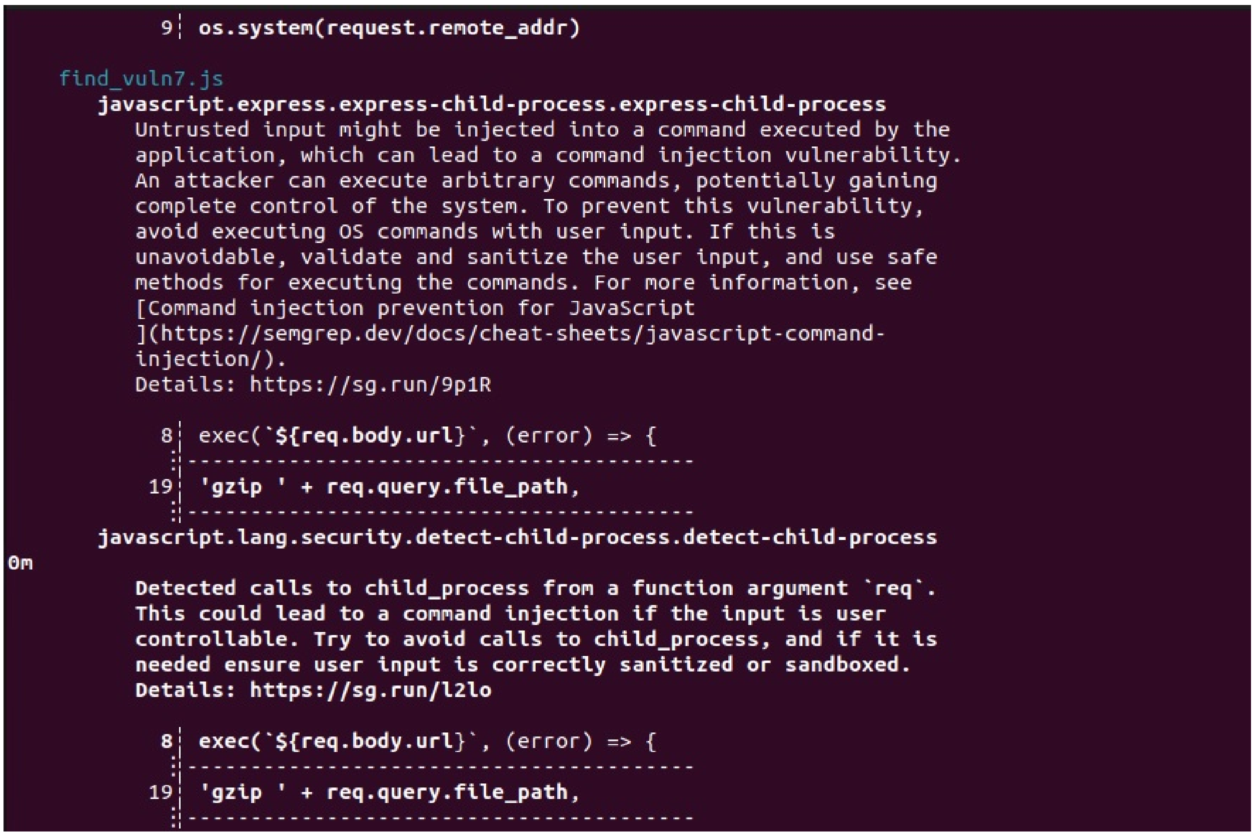
find_vuln8.php – уязвимость exec-use (11 строка)

```
semgrep -- config="r/php.lang.security.exec-use.exec-use"
```

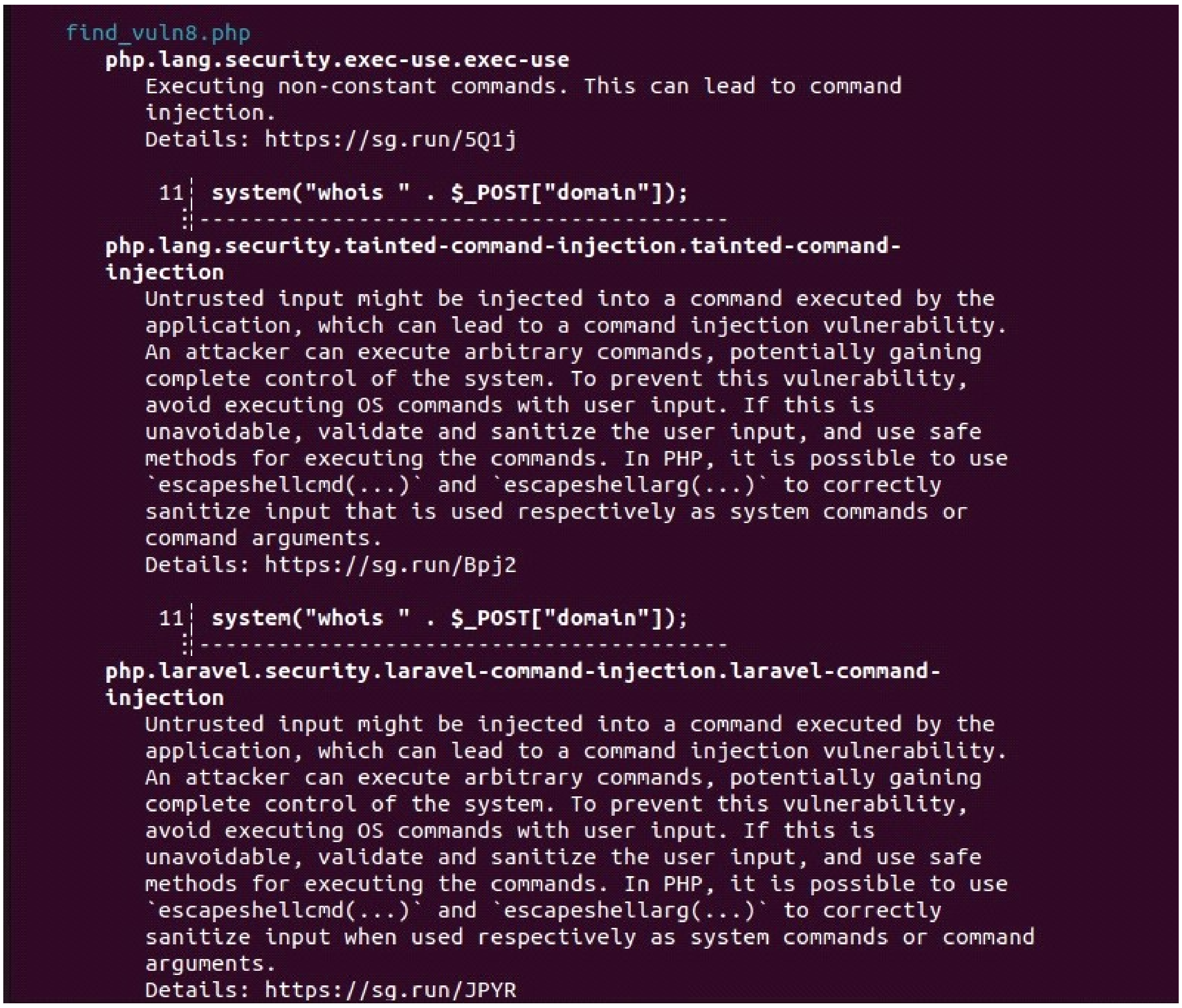
Результат сканирования `semgrep scan --config auto`



```
┌──────────────────┐
│ 11 Code Findings │
└──────────────────┘

   find_vuln6.py
      python.django.security.injection.command.command-injection-os-
      system.command-injection-os-system
         Request data detected in os.system. This could be vulnerable to a
         command injection and should be avoided. If this must be done, use
         the 'subprocess' module instead and pass the arguments as a list.
         See https://owasp.org/www-community/attacks/Command_Injection for
         more information.
         Details: https://sg.run/Gen2

           9┆ os.system(request.remote_addr)
           ⋮┆----------------------------------------
      python.flask.security.audit.debug-enabled.debug-enabled
         Detected Flask app with debug=True. Do not deploy to production
         with this flag enabled as it will leak sensitive information.
         Instead, consider using Flask configuration variables or setting
         'debug' using system environment variables.
         Details: https://sg.run/dKrd

          14┆ app.run(debug=True)
           ⋮┆----------------------------------------
      python.flask.security.injection.os-system-injection.os-system-
      injection
         User data detected in os.system. This could be vulnerable to a
         command injection and should be avoided. If this must be done, use
         the 'subprocess' module instead and pass the arguments as a list.
         Details: https://sg.run/4xzz

           9┆ os.system(request.remote_addr)
```



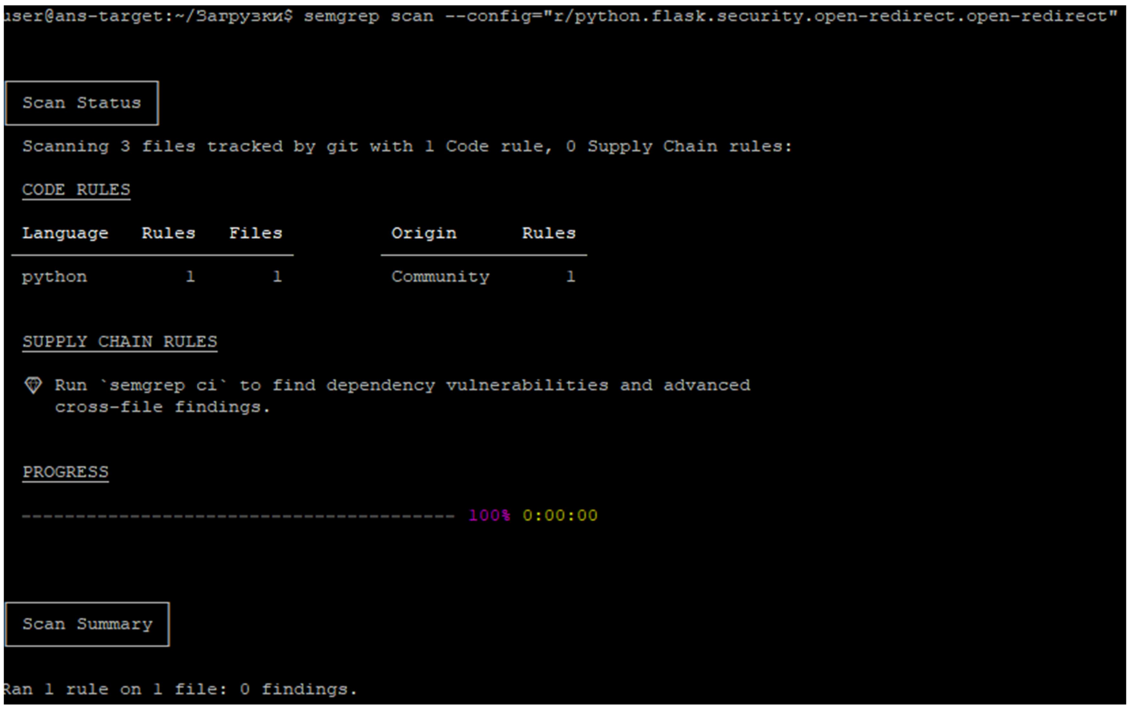
```
           9┆ os.system(request.remote_addr)

   find_vuln7.js
      javascript.express.express-child-process.express-child-process
         Untrusted input might be injected into a command executed by the
         application, which can lead to a command injection vulnerability.
         An attacker can execute arbitrary commands, potentially gaining
         complete control of the system. To prevent this vulnerability,
         avoid executing OS commands with user input. If this is
         unavoidable, validate and sanitize the user input, and use safe
         methods for executing the commands. For more information, see
         [Command injection prevention for JavaScript
         ](https://semgrep.dev/docs/cheat-sheets/javascript-command-
         injection/).
         Details: https://sg.run/9p1R

           8┆ exec(`${req.body.url}`, (error) => {
           ⋮┆----------------------------------------
          19┆ 'gzip ' + req.query.file_path,
           ⋮┆----------------------------------------
      javascript.lang.security.detect-child-process.detect-child-process
0m

         Detected calls to child_process from a function argument `req`.
         This could lead to a command injection if the input is user
         controllable. Try to avoid calls to child_process, and if it is
         needed ensure user input is correctly sanitized or sandboxed.
         Details: https://sg.run/l2lo

           8┆ exec(`${req.body.url}`, (error) => {
           ⋮┆----------------------------------------
          19┆ 'gzip ' + req.query.file_path,
           ⋮┆----------------------------------------
```



```
find_vuln8.php
   php.lang.security.exec-use.exec-use
      Executing non-constant commands. This can lead to command
      injection.
      Details: https://sg.run/5Q1j

       11┆ system("whois " . $_POST["domain"]);
         ⋮┆----------------------------------------
   php.lang.security.tainted-command-injection.tainted-command-
   injection
      Untrusted input might be injected into a command executed by the
      application, which can lead to a command injection vulnerability.
      An attacker can execute arbitrary commands, potentially gaining
      complete control of the system. To prevent this vulnerability,
      avoid executing OS commands with user input. If this is
      unavoidable, validate and sanitize the user input, and use safe
      methods for executing the commands. In PHP, it is possible to use
      `escapeshellcmd(...)` and `escapeshellarg(...)` to correctly
      sanitize input that is used respectively as system commands or
      command arguments.
      Details: https://sg.run/Bpj2

       11┆ system("whois " . $_POST["domain"]);
         ⋮┆----------------------------------------
   php.laravel.security.laravel-command-injection.laravel-command-
   injection
      Untrusted input might be injected into a command executed by the
      application, which can lead to a command injection vulnerability.
      An attacker can execute arbitrary commands, potentially gaining
      complete control of the system. To prevent this vulnerability,
      avoid executing OS commands with user input. If this is
      unavoidable, validate and sanitize the user input, and use safe
      methods for executing the commands. In PHP, it is possible to use
      `escapeshellcmd(...)` and `escapeshellarg(...)` to correctly
      sanitize input when used respectively as system commands or command
      arguments.
      Details: https://sg.run/JPYR
```

Результат сканирования

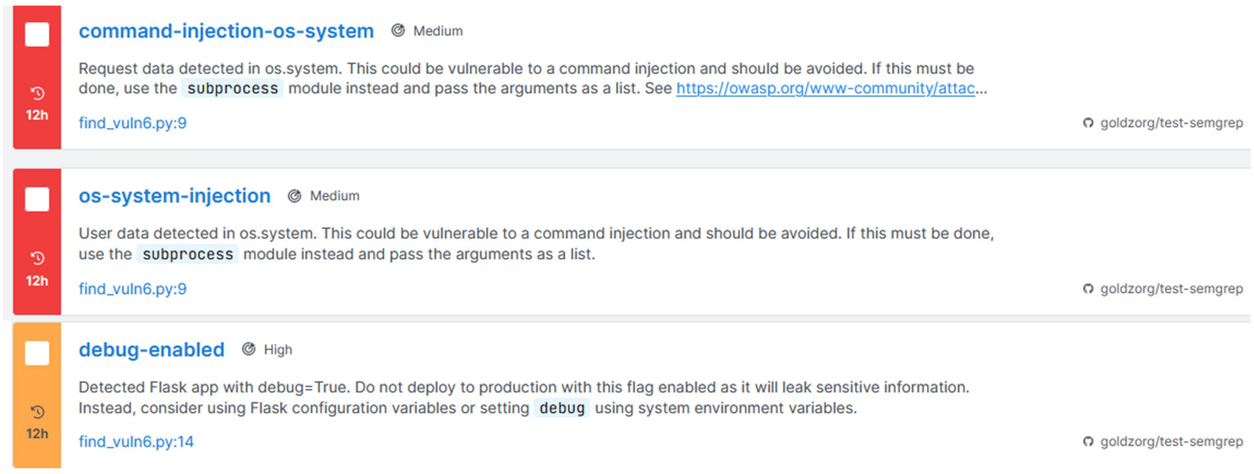
semgrep scan --config="r/python.flask.security.open-redirect.open-redirect



```
user@ans-target:~/Загрузки$ semgrep scan --config="r/python.flask.security.open-redirect.open-redirect"


┌─────────────┐
│ Scan Status │
└─────────────┘
  Scanning 3 files tracked by git with 1 Code rule, 0 Supply Chain rules:

  CODE RULES

  Language    Rules    Files          Origin       Rules
 ─────────────────────────────        ─────────────────────
  python          1        1          Community        1


  SUPPLY CHAIN RULES

  💎 Run `semgrep ci` to find dependency vulnerabilities and advanced
     cross-file findings.


  PROGRESS

  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00



┌──────────────┐
│ Scan Summary │
└──────────────┘

Ran 1 rule on 1 file: 0 findings.
```

Сканирования проекта CI/CD Github Actions

find_vuln6.py



command-injection-os-system Medium

Request data detected in os.system. This could be vulnerable to a command injection and should be avoided. If this must be done, use the `subprocess` module instead and pass the arguments as a list. See https://owasp.org/www-community/attac...

12h find_vuln6.py:9 goldzorg/test-semgrep

os-system-injection Medium

User data detected in os.system. This could be vulnerable to a command injection and should be avoided. If this must be done, use the `subprocess` module instead and pass the arguments as a list.

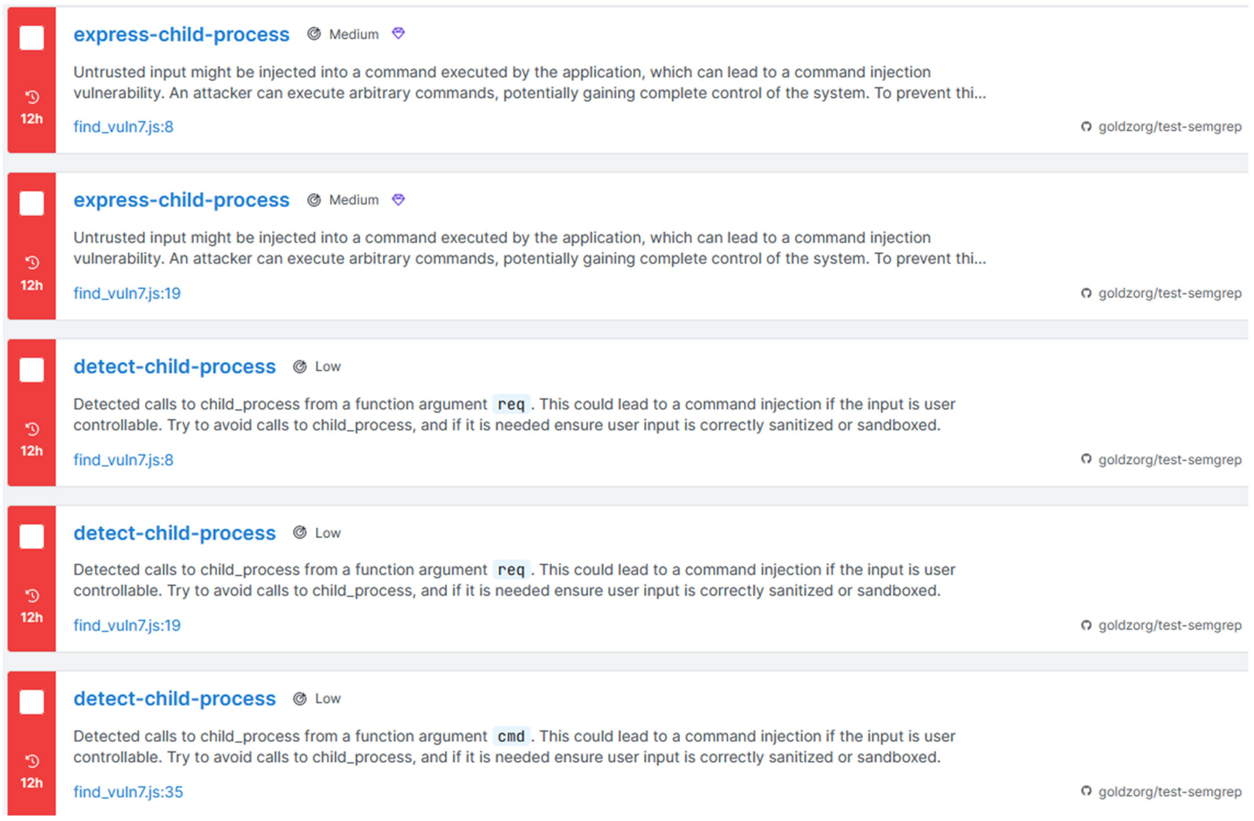
12h find_vuln6.py:9 goldzorg/test-semgrep

debug-enabled High

Detected Flask app with debug=True. Do not deploy to production with this flag enabled as it will leak sensitive information. Instead, consider using Flask configuration variables or setting `debug` using system environment variables.

12h find_vuln6.py:14 goldzorg/test-semgrep

find_vuln7.js



express-child-process Medium

Untrusted input might be injected into a command executed by the application, which can lead to a command injection vulnerability. An attacker can execute arbitrary commands, potentially gaining complete control of the system. To prevent thi...

12h find_vuln7.js:8 goldzorg/test-semgrep

express-child-process Medium

Untrusted input might be injected into a command executed by the application, which can lead to a command injection vulnerability. An attacker can execute arbitrary commands, potentially gaining complete control of the system. To prevent thi...

12h find_vuln7.js:19 goldzorg/test-semgrep

detect-child-process Low

Detected calls to child_process from a function argument `req`. This could lead to a command injection if the input is user controllable. Try to avoid calls to child_process, and if it is needed ensure user input is correctly sanitized or sandboxed.

12h find_vuln7.js:8 goldzorg/test-semgrep

detect-child-process Low

Detected calls to child_process from a function argument `req`. This could lead to a command injection if the input is user controllable. Try to avoid calls to child_process, and if it is needed ensure user input is correctly sanitized or sandboxed.

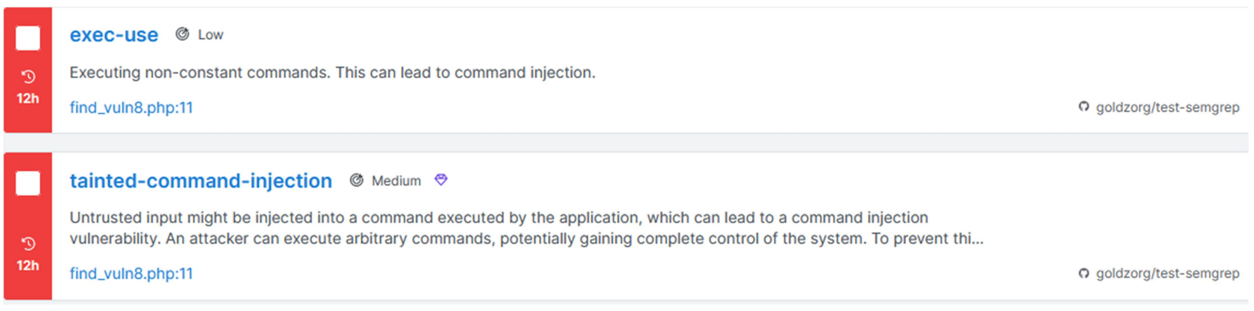
12h find_vuln7.js:19 goldzorg/test-semgrep

detect-child-process Low

Detected calls to child_process from a function argument `cmd`. This could lead to a command injection if the input is user controllable. Try to avoid calls to child_process, and if it is needed ensure user input is correctly sanitized or sandboxed.

12h find_vuln7.js:35 goldzorg/test-semgrep

find_vuln8.php



exec-use Low

Executing non-constant commands. This can lead to command injection.

12h find_vuln8.php:11 goldzorg/test-semgrep

tainted-command-injection Medium

Untrusted input might be injected into a command executed by the application, which can lead to a command injection vulnerability. An attacker can execute arbitrary commands, potentially gaining complete control of the system. To prevent thi...

12h find_vuln8.php:11 goldzorg/test-semgrep