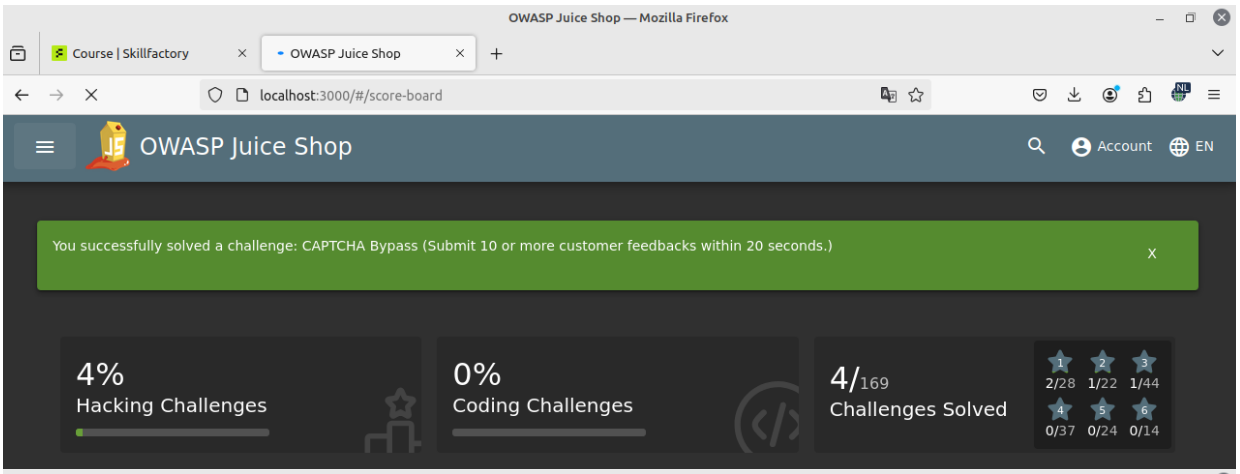
Средства автоматизированного поиска уязвимостей в веб-приложениях. Практическое задание.

1. Прохождение лабораторных работ в Juice Shop. Запустил Juice shop из скачанного ранее образа docker run --rm -p 3000:3000 bkimminich/juice-shop, открыл через браузер по адресу http://localhost:3000

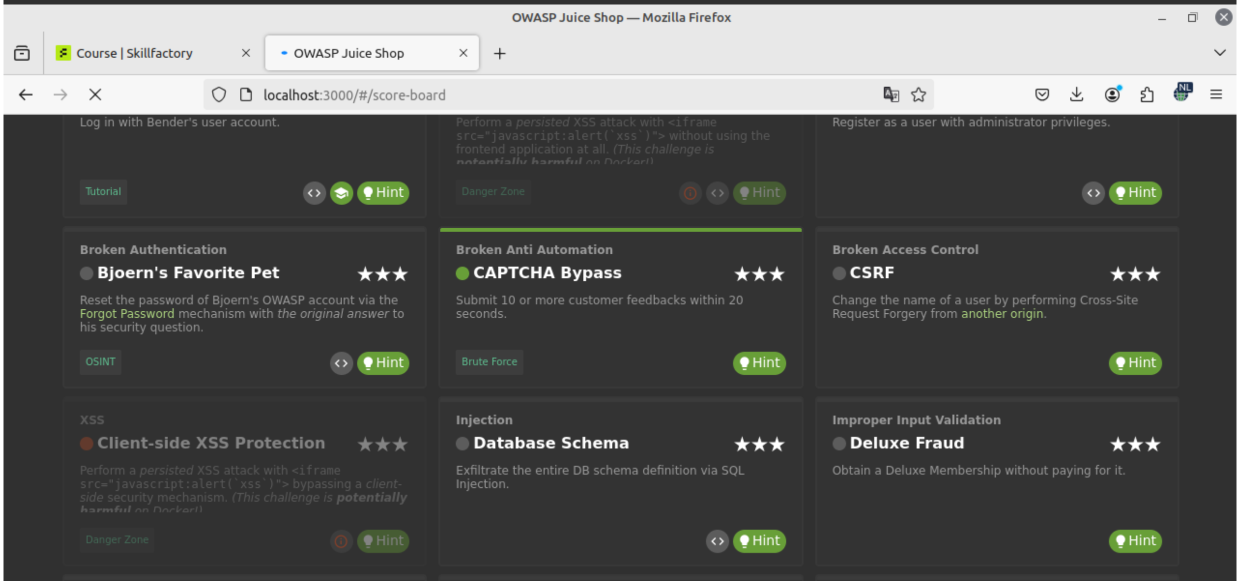
Задание CAPTCHA Bypass


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

4% Hacking Challenges

0% Coding Challenges

4/169 Challenges Solved


Log in with Bender's user account.

Perform a *persisted* XSS attack with `<iframe src="javascript:alert(`xss`)">` without using the frontend application at all. *(This challenge is potentially harmful on Docker!)*

Register as a user with administrator privileges.

Broken Authentication
Bjoern's Favorite Pet ★★★
Reset the password of Bjoern's OWASP account via the Forgot Password mechanism with *the original answer* to his security question.
OSINT

Broken Anti Automation
CAPTCHA Bypass ★★★
Submit 10 or more customer feedbacks within 20 seconds.
Brute Force

Broken Access Control
CSRF ★★★
Change the name of a user by performing Cross-Site Request Forgery from another origin.

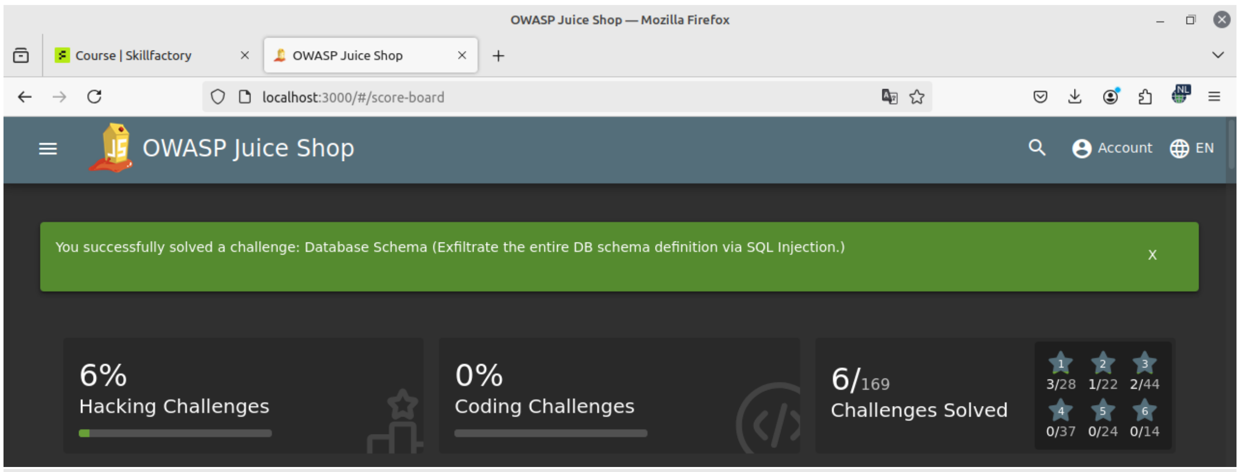
XSS
Client-side XSS Protection ★★★
Perform a *persisted* XSS attack with `<iframe src="javascript:alert(`xss`)">` bypassing a *client-side* security mechanism. *(This challenge is* ***potentially harmful*** *on Docker!)*
Danger Zone

Injection
Database Schema ★★★
Exfiltrate the entire DB schema definition via SQL Injection.

Improper Input Validation
Deluxe Fraud ★★★
Obtain a Deluxe Membership without paying for it.

OWASP Juice Shop — Mozilla Firefox
Course | Skillfactory
OWASP Juice Shop
localhost:3000/#/score-board
Tutorial
Danger Zone
Hint

Задание Database Schema


OWASP Juice Shop — Mozilla Firefox

localhost:3000/#/score-board

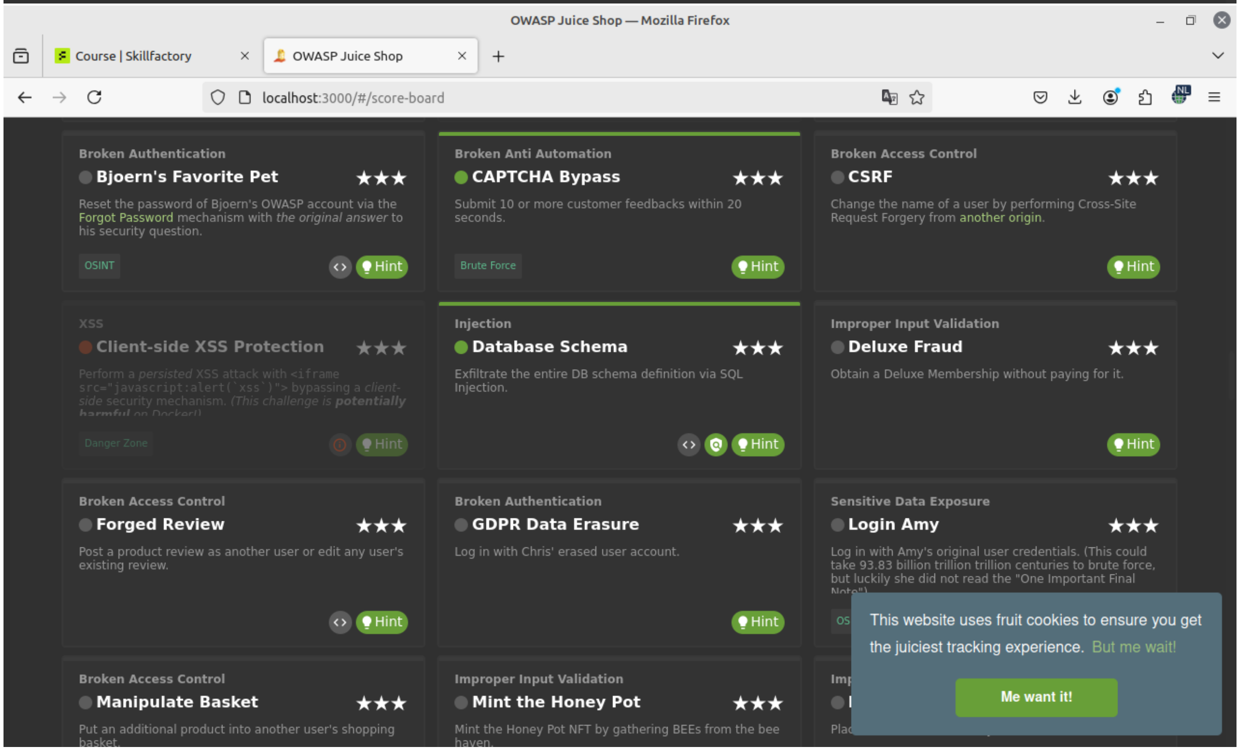
You successfully solved a challenge: Database Schema (Exfiltrate the entire DB schema definition via SQL Injection.)

6% Hacking Challenges

0% Coding Challenges

6/169 Challenges Solved

1	2	3
3/28	1/22	2/44
4	5	6
0/37	0/24	0/14


OWASP Juice Shop — Mozilla Firefox

localhost:3000/#/score-board

Broken Authentication
Bjoern's Favorite Pet ★★★
Reset the password of Bjoern's OWASP account via the Forgot Password mechanism with *the original answer* to his security question.
OSINT

Broken Anti Automation
CAPTCHA Bypass ★★★
Submit 10 or more customer feedbacks within 20 seconds.
Brute Force

Broken Access Control
CSRF ★★★
Change the name of a user by performing Cross-Site Request Forgery from another origin.

XSS
Client-side XSS Protection ★★★
Perform a *persisted* XSS attack with `<iframe src="javascript:alert(`xss`)">` bypassing a *client-side* security mechanism. *(This challenge is **potentially harmful** on Docker!)*
Danger Zone

Injection
Database Schema ★★★
Exfiltrate the entire DB schema definition via SQL Injection.

Improper Input Validation
Deluxe Fraud ★★★
Obtain a Deluxe Membership without paying for it.

Broken Access Control
Forged Review ★★★
Post a product review as another user or edit any user's existing review.

Broken Authentication
GDPR Data Erasure ★★★
Log in with Chris' erased user account.

Sensitive Data Exposure
Login Amy ★★★
Log in with Amy's original user credentials. (This could take 93.83 billion trillion trillion centuries to brute force, but luckily she did not read the "One Important Final Note")

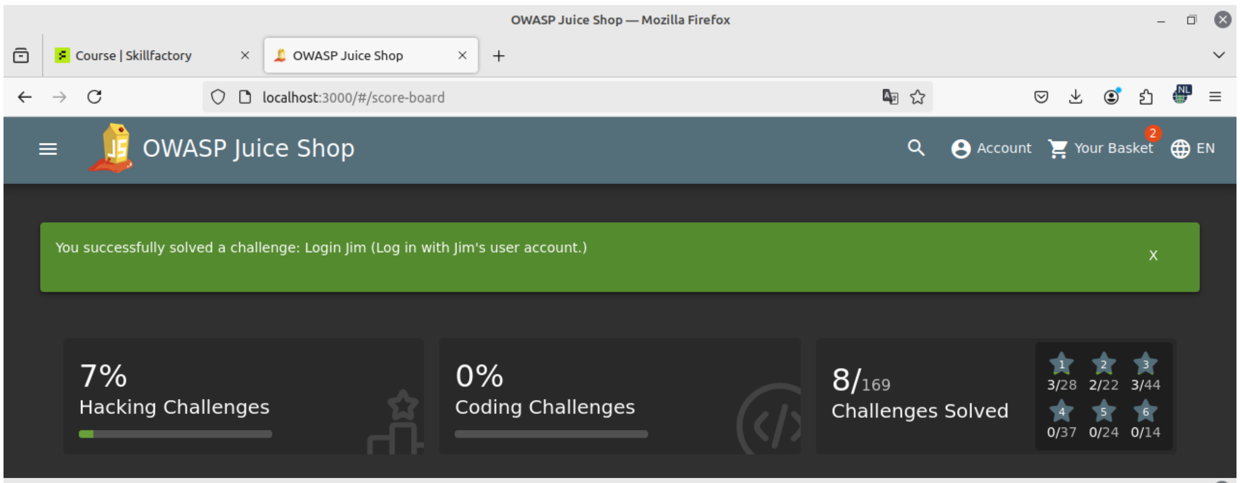
Broken Access Control
Manipulate Basket ★★★
Put an additional product into another user's shopping basket.

Improper Input Validation
Mint the Honey Pot ★★★
Mint the Honey Pot NFT by gathering BEEs from the bee haven.

This website uses fruit cookies to ensure you get the juiciest tracking experience. But me wait!

Me want it!

Задание Login Jim


OWASP Juice Shop — Mozilla Firefox

localhost:3000/#/score-board

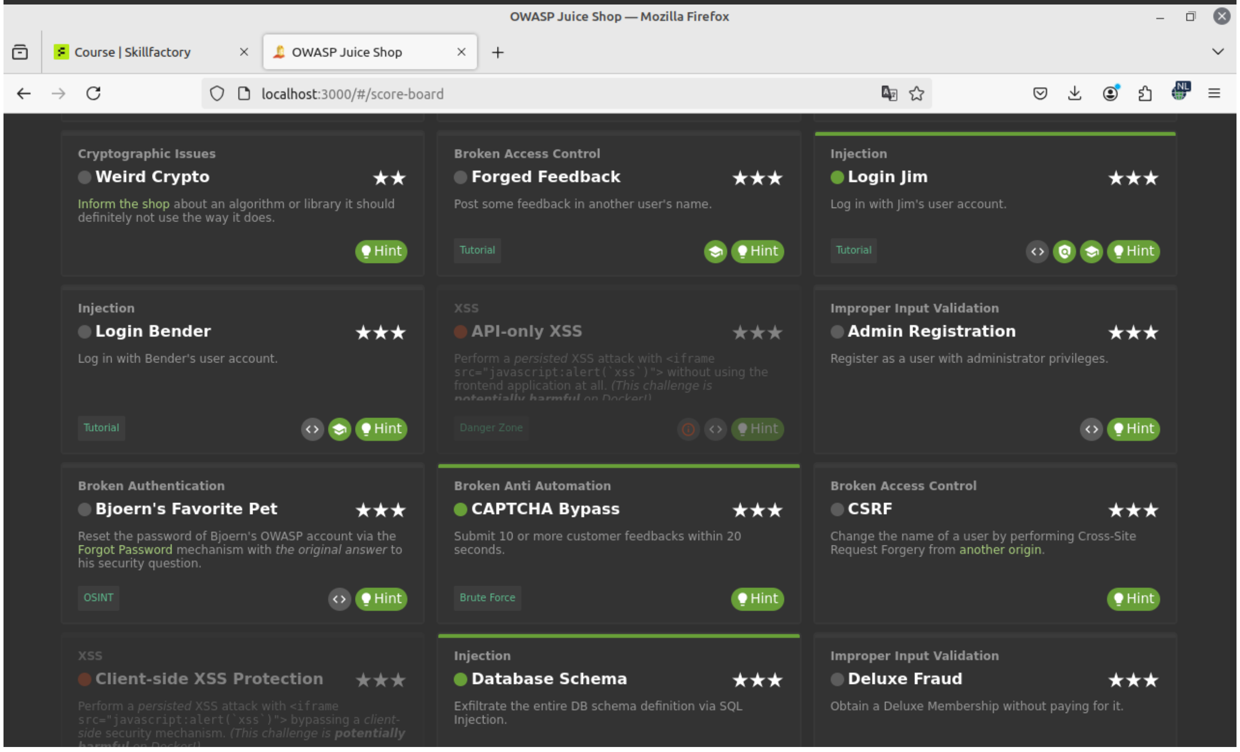
You successfully solved a challenge: Login Jim (Log in with Jim's user account.)

7% Hacking Challenges

0% Coding Challenges

8/169 Challenges Solved

3/28 2/22 3/44 0/37 0/24 0/14


OWASP Juice Shop — Mozilla Firefox

localhost:3000/#/score-board

Cryptographic Issues
Weird Crypto
Inform the shop about an algorithm or library it should definitely not use the way it does.

Broken Access Control
Forged Feedback
Post some feedback in another user's name.

Injection
Login Jim
Log in with Jim's user account.

Injection
Login Bender
Log in with Bender's user account.

XSS
API-only XSS
Perform a *persisted* XSS attack with <iframe src="javascript:alert(`xss`)"> without using the frontend application at all. *(This challenge is potentially harmful on Docker!)*

Improper Input Validation
Admin Registration
Register as a user with administrator privileges.

Broken Authentication
Bjoern's Favorite Pet
Reset the password of Bjoern's OWASP account via the Forgot Password mechanism with *the original answer* to his security question.

Broken Anti Automation
CAPTCHA Bypass
Submit 10 or more customer feedbacks within 20 seconds.

Broken Access Control
CSRF
Change the name of a user by performing Cross-Site Request Forgery from another origin.

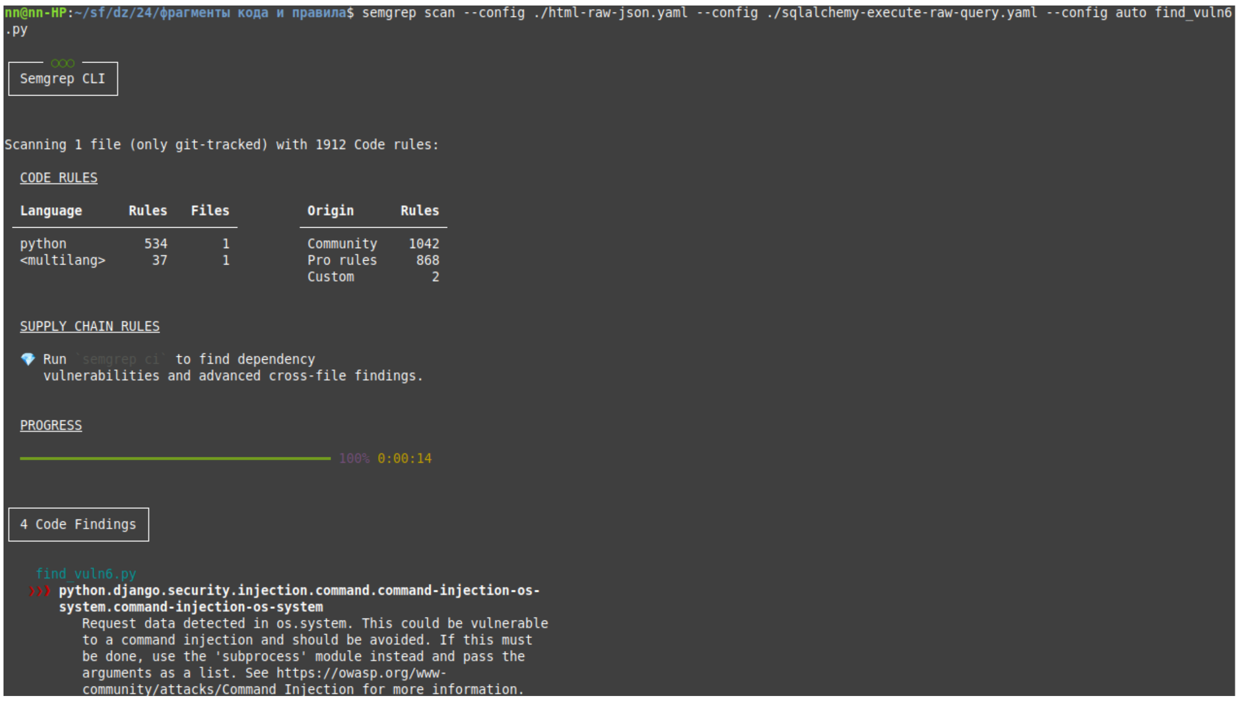
XSS
Client-side XSS Protection
Perform a *persisted* XSS attack with <iframe src="javascript:alert(`xss`)"> bypassing a *client-side* security mechanism. *(This challenge is **potentially harmful** on Docker!)*

Injection
Database Schema
Exfiltrate the entire DB schema definition via SQL Injection.

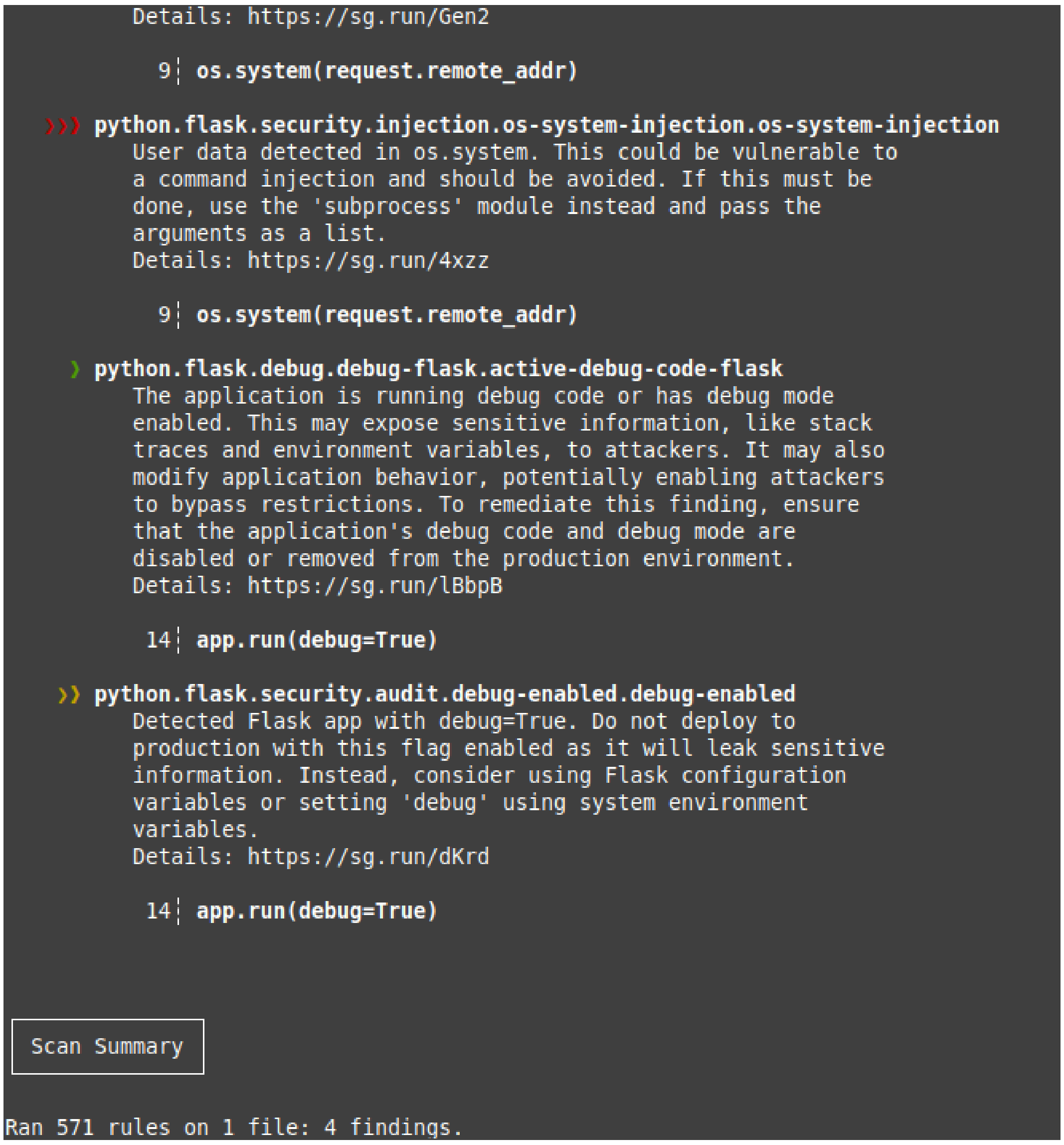
Improper Input Validation
Deluxe Fraud
Obtain a Deluxe Membership without paying for it.

2. Для сканирования фрагментов кода использовал утилиту semgrep. Команда для выполнения содержит дополнительные файлы конфигурации с гита и автоматически созданный файл. semgrep scan --config ./html-raw-json.yaml --config ./sqlalchemy-execute-raw-query.yaml --config auto ИМЯ ФАЙЛА (при необходимости путь, если запускается из другой директории).

Фрагмент № 1. Файл find_vuln6.py

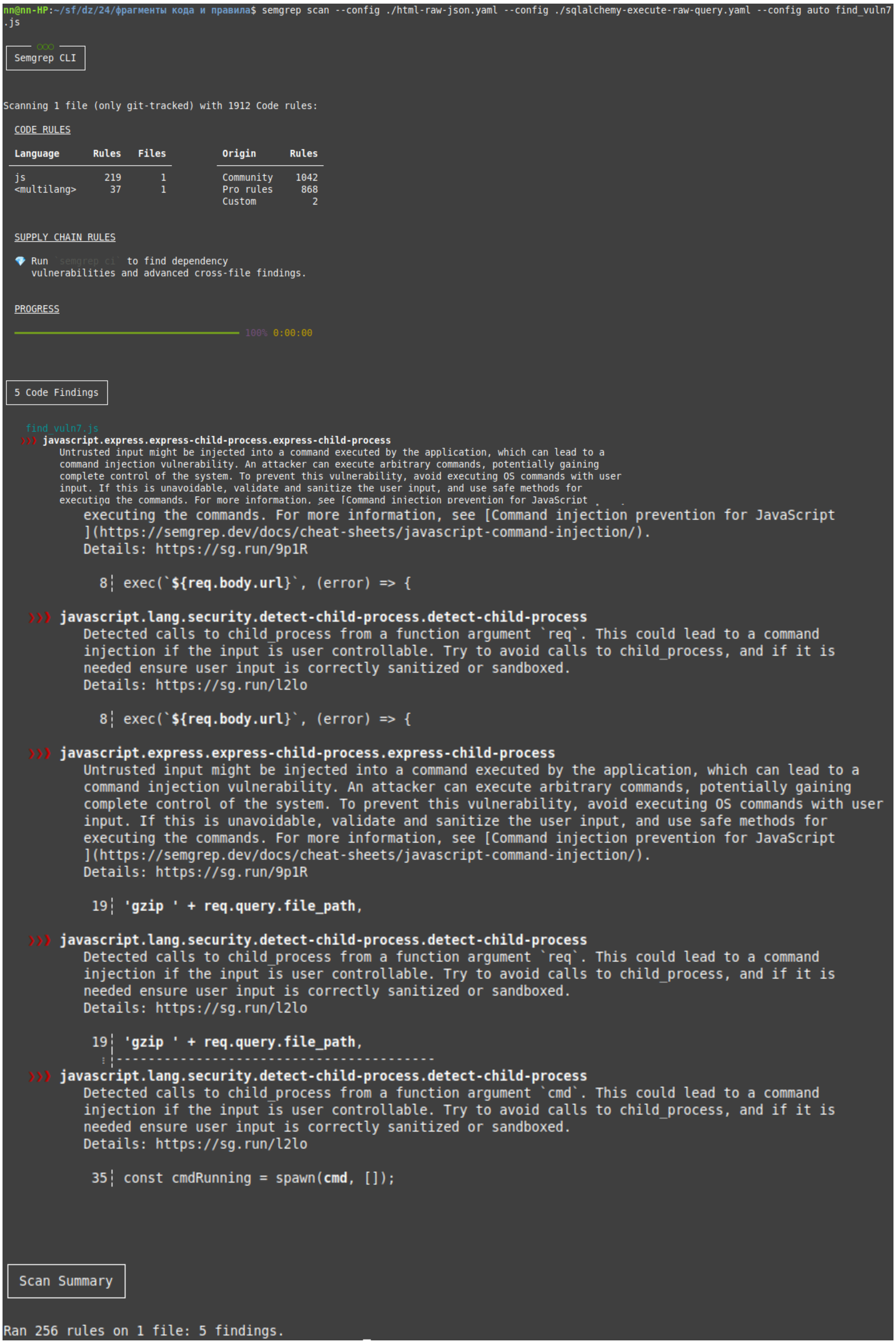


```
nn@nn-HP:~/sf/dz/24/фрагменты кода и правила$ semgrep scan --config ./html-raw-json.yaml --config ./sqlalchemy-execute-raw-query.yaml --config auto find_vuln6
.py

┌──── ○○○ ────┐
│ Semgrep CLI │
└─────────────┘


Scanning 1 file (only git-tracked) with 1912 Code rules:

  CODE RULES

  Language      Rules   Files          Origin      Rules
 ─────────────────────────────        ───────────────────
  python          534       1          Community    1042
  <multilang>      37       1          Pro rules     868
                                       Custom          2


  SUPPLY CHAIN RULES

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


  PROGRESS

  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:14


┌─────────────────┐
│ 4 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.
```



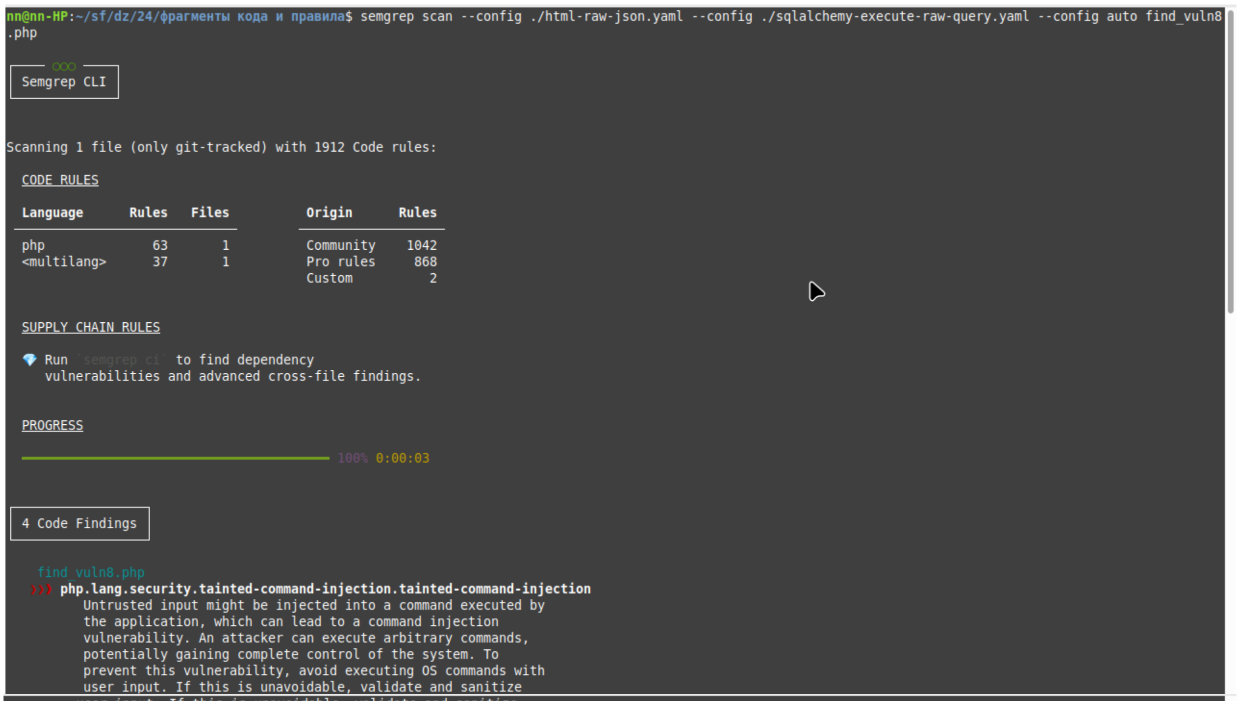
```
          Details: https://sg.run/Gen2

            9┆ os.system(request.remote_addr)

   ❯❯❱ python.flask.security.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)

     ❱ python.flask.debug.debug-flask.active-debug-code-flask
          The application is running debug code or has debug mode
          enabled. This may expose sensitive information, like stack
          traces and environment variables, to attackers. It may also
          modify application behavior, potentially enabling attackers
          to bypass restrictions. To remediate this finding, ensure
          that the application's debug code and debug mode are
          disabled or removed from the production environment.
          Details: https://sg.run/lBbpB

           14┆ app.run(debug=True)

    ❯❱ 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)



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

Ran 571 rules on 1 file: 4 findings.
```

Фрагмент № 2. Файл find_vuln7.js

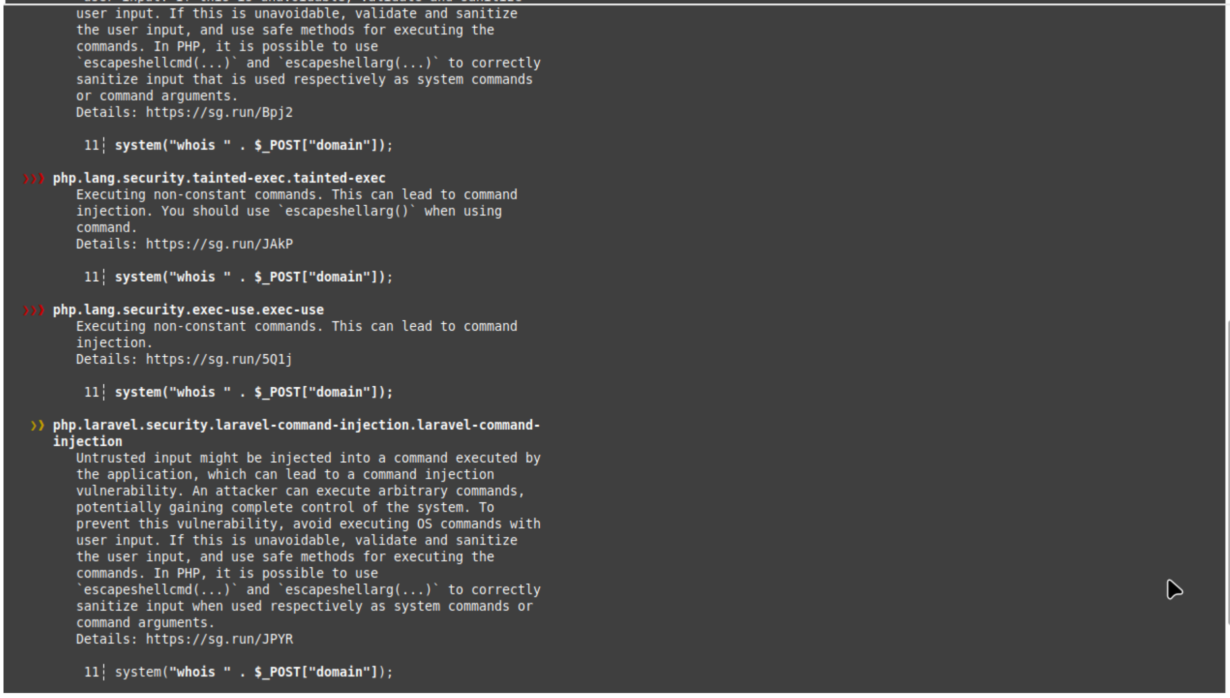


```
nn@nn-HP:~/sf/dz/24/фрагменты кода и правила$ semgrep scan --config ./html-raw-json.yaml --config ./sqlalchemy-execute-raw-query.yaml --config auto find_vuln7
.js

┌──── ○○○ ────┐
│ Semgrep CLI │
└─────────────┘


Scanning 1 file (only git-tracked) with 1912 Code rules:

  CODE RULES

  Language      Rules   Files          Origin      Rules
 ─────────────────────────────        ───────────────────
  js              219       1          Community    1042
  <multilang>      37       1          Pro rules     868
                                       Custom          2


  SUPPLY CHAIN RULES

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


  PROGRESS

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


┌─────────────────┐
│ 5 Code Findings │
└─────────────────┘

    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
          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) => {

   ❯❯❱ javascript.lang.security.detect-child-process.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.
          Details: https://sg.run/l2lo

            8┆ exec(`${req.body.url}`, (error) => {

   ❯❯❱ 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

           19┆ 'gzip ' + req.query.file_path,

   ❯❯❱ javascript.lang.security.detect-child-process.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.
          Details: https://sg.run/l2lo

           19┆ 'gzip ' + req.query.file_path,
            ⋮┆----------------------------------------
   ❯❯❱ javascript.lang.security.detect-child-process.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.
          Details: https://sg.run/l2lo

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


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

Ran 256 rules on 1 file: 5 findings.
```

Фрагмент № 3. Файл find_vuln8.php



```
nn@nn-HP:~/sf/dz/24/фрагменты кода и правила$ semgrep scan --config ./html-raw-json.yaml --config ./sqlalchemy-execute-raw-query.yaml --config auto find_vuln8
.php

┌──── ○○○ ────┐
│ Semgrep CLI │
└─────────────┘


Scanning 1 file (only git-tracked) with 1912 Code rules:

  CODE RULES

  Language      Rules   Files          Origin      Rules
 ─────────────────────────────        ───────────────────
  php              63       1          Community    1042
  <multilang>      37       1          Pro rules     868
                                       Custom          2


  SUPPLY CHAIN RULES

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


  PROGRESS

  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:03


┌─────────────────┐
│ 4 Code Findings │
└─────────────────┘

    find_vuln8.php
   ❯❯❱ 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
```



```
          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.lang.security.tainted-exec.tainted-exec
          Executing non-constant commands. This can lead to command
          injection. You should use `escapeshellarg()` when using
          command.
          Details: https://sg.run/JAkP

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

   ❯❯❱ 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.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

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