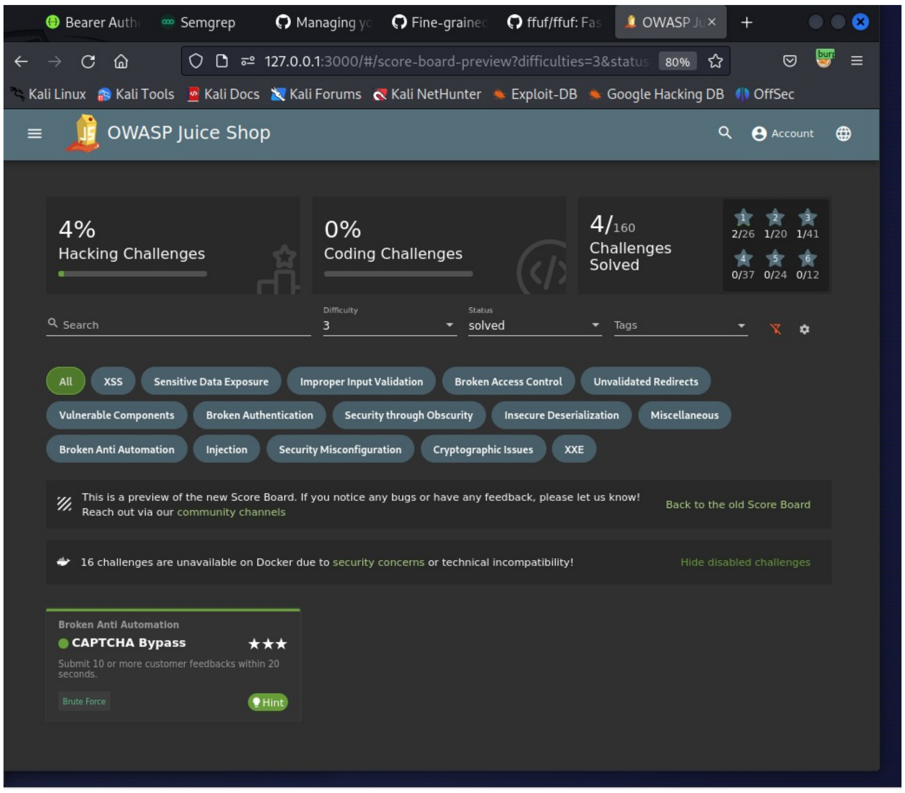
Лабораторные работы в Juice Shop

1. CAPTCHA ByPass



4% Hacking Challenges

0% Coding Challenges

4/160 Challenges Solved

2/26 1/20 1/41 0/37 0/24 0/12

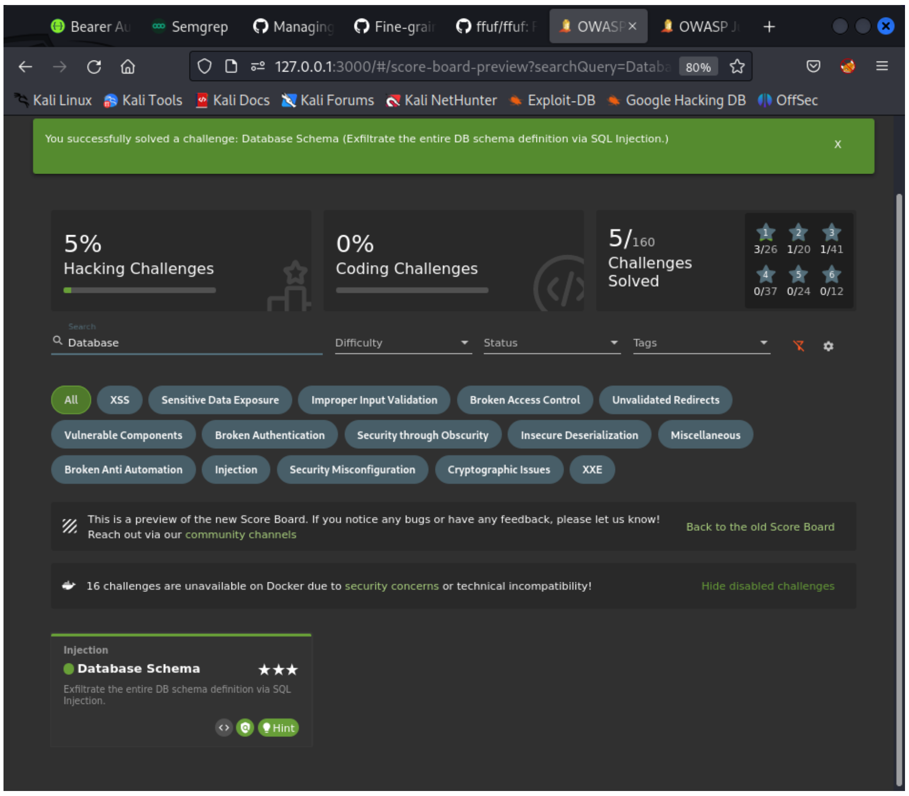
Difficulty: 3 Status: solved

This is a preview of the new Score Board. If you notice any bugs or have any feedback, please let us know! Reach out via our community channels

16 challenges are unavailable on Docker due to security concerns or technical incompatibility!

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

2. Database Schema



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

5% Hacking Challenges

0% Coding Challenges

5/160 Challenges Solved

3/26 1/20 1/41 0/37 0/24 0/12

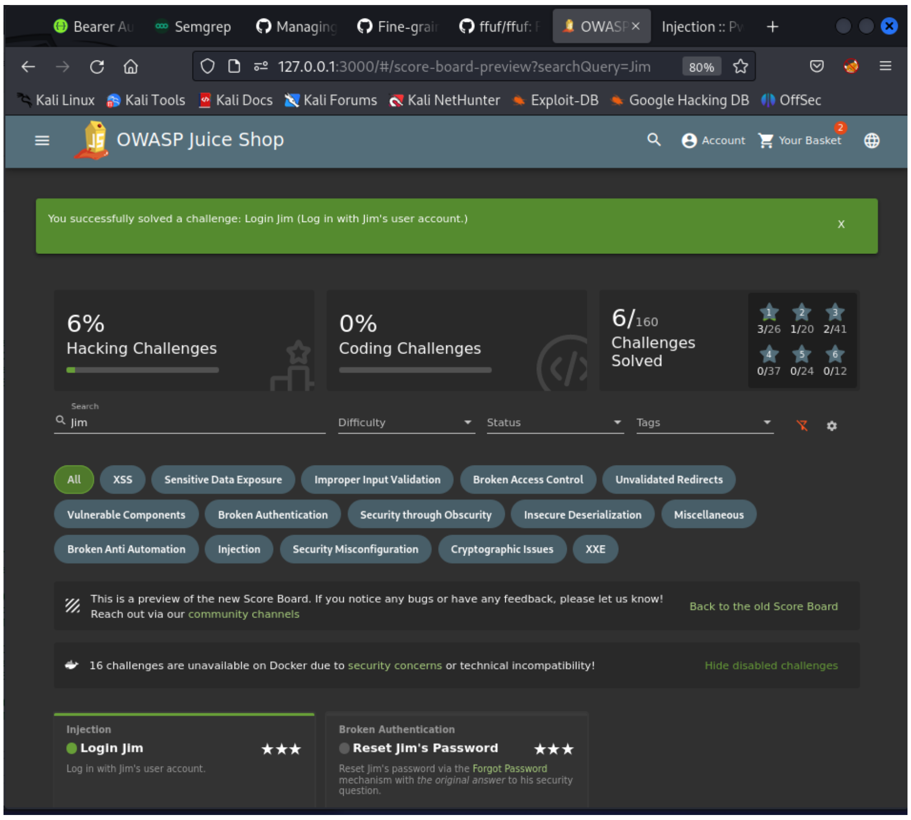
Search: Database

This is a preview of the new Score Board. If you notice any bugs or have any feedback, please let us know! Reach out via our community channels

16 challenges are unavailable on Docker due to security concerns or technical incompatibility!

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

3. Login Jim



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

6% Hacking Challenges

0% Coding Challenges

6/160 Challenges Solved

3/26 1/20 2/41
0/37 0/24 0/12

Jim

This is a preview of the new Score Board. If you notice any bugs or have any feedback, please let us know! Reach out via our community channels

16 challenges are unavailable on Docker due to security concerns or technical incompatibility!

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

Broken Authentication
Reset Jim's Password ★★★
Reset Jim's password via the Forgot Password mechanism with *the original answer* to his security question.

Сканирование уязвимостей

Команда:

```
$ semgrep scan --config="p/javascript" --config="p/php" --config="p/python"
```

Результаты:

```
┌──────────────┐
│ Scan Status │
└──────────────┘

Scanning 3 files tracked by git with 432 Code rules, 0 Supply Chain rules, 202 Pro rules:

CODE RULES

Language   Rules   Files        Origin     Rules
─────────────────────────────────────     ─────────────────────────

python      255      1         Community    230
js          126      1         Pro rules    202
php          43      1

SUPPLY CHAIN RULES

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

```
PROGRESS
 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00

┌──────────────────┐
│ 7 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)

  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,
  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 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

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

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

Ran 432 rules on 3 files: 7 findings.
```