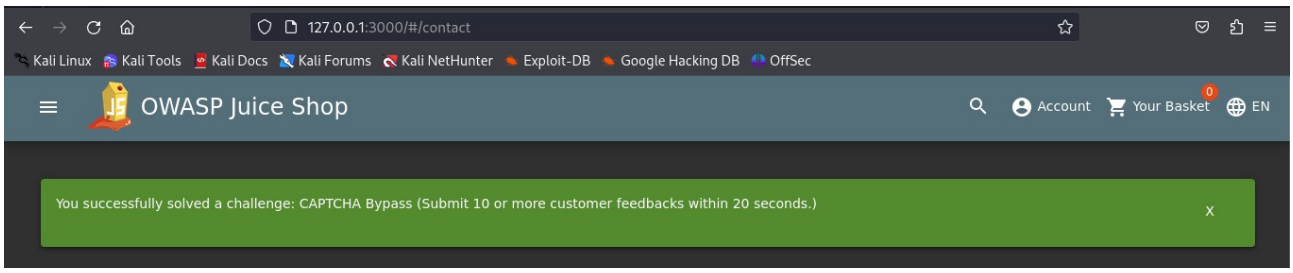
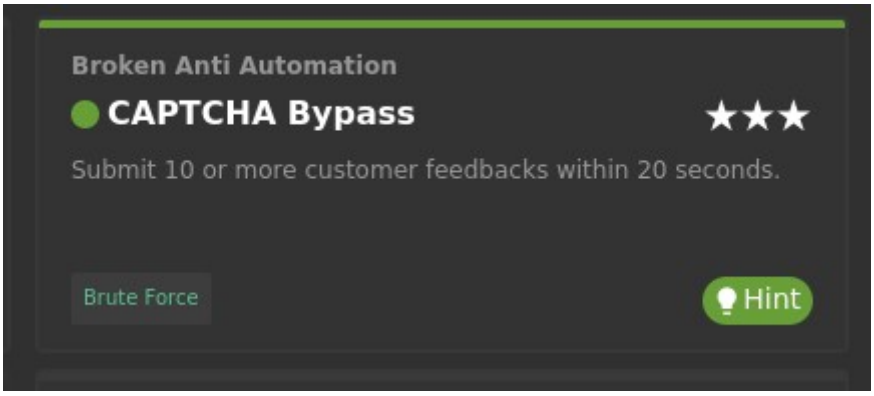
Практическое задание по модулю “Средства автоматизированного поиска уязвимостей в веб-приложениях”

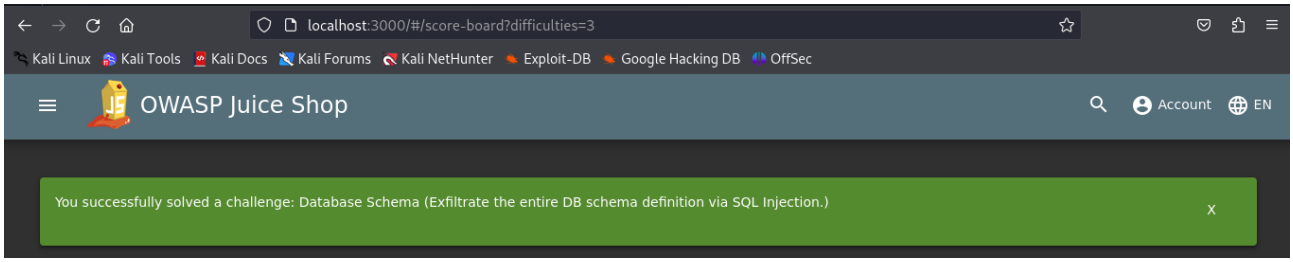
Лабораторные работы в Juice Shop.

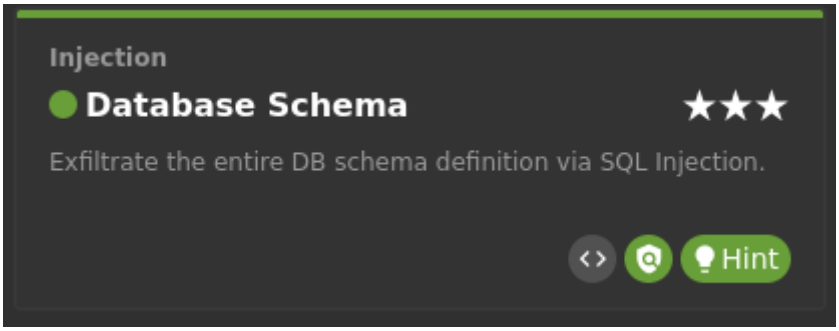
1. CAPTCHA Bypass



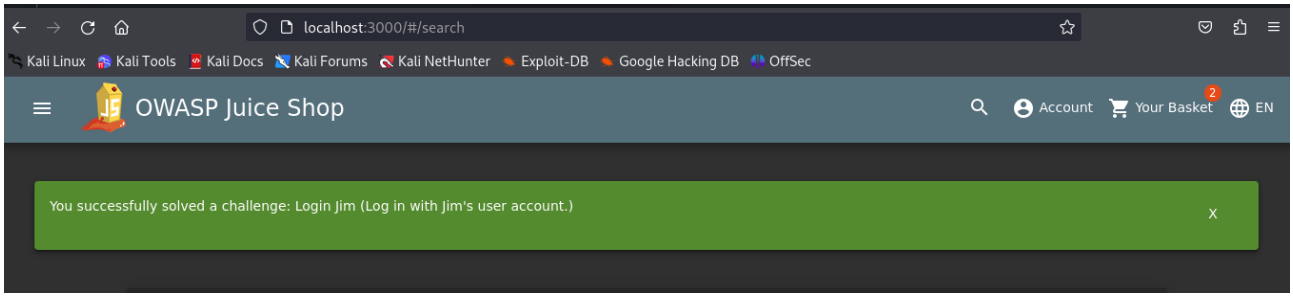


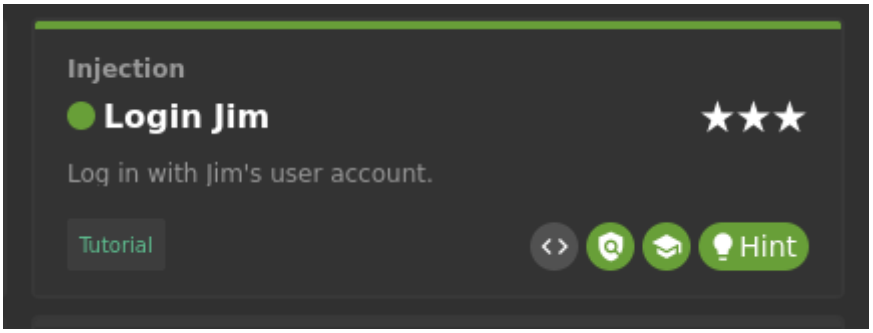
2. Database Schema





3. Login Jim





Анализ фрагментов кода на наличие уязвимостей.

1. Файл “*find_vuln6.py*”.
Уязвимости:

- OS command injection
- Dangerous system call
- Debug enabled

```
┌──(user㉿kali)-[~/Downloads]
└─$ semgrep scan --config /home/user/semgrep-rules/python/ find_vuln6.py
```

```
find_vuln6.py
  home.user.semgrep-rules.python.django.security.injection.command.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)
        ┆┆------------------------------------
  home.user.semgrep-rules.python.flask.security.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.

       9┆ os.system(request.remote_addr)
        ┆┆------------------------------------
  home.user.semgrep-rules.python.lang.security.audit.dangerous-system-call-audit
     Found dynamic content used in a system call. This is dangerous if external data can reach
     this function call because it allows a malicious actor to execute commands. Use the
     'subprocess' module instead, which is easier to use without accidentally exposing a command
     injection vulnerability.

       9┆ os.system(request.remote_addr)
        ┆┆------------------------------------
  home.user.semgrep-rules.python.flask.security.audit.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.

      14┆ app.run(debug=True)
```

2. Файл “*find_vuln7.js*”
Уязвимости:

- Detect child process

```
┌──(user㉿kali)-[~/Downloads]
└─$ semgrep scan --config /home/user/semgrep-rules/javascript/ find_vuln7.js
```

```
find_vuln7.js
   home.user.semgrep-rules.javascript.lang.security.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,
        ⋮┆----------------------------------------
   home.user.semgrep-rules.javascript.lang.security.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, []);
```

3. Файл “*find_vuln8.php*”
Уязвимости:

- Tainted exec
- Exec use

```
┌──(user㉿kali)-[~/Downloads]
└─$ semgrep scan --config /home/user/semgrep-rules/php/ find_vuln8.php
```

```
find_vuln8.php
   home.user.semgrep-rules.php.lang.security.tainted-exec
      Executing non-constant commands. This can lead to command injection. You should use
      `escapeshellarg()` when using command.

       11┆ system("whois " . $_POST["domain"]);
         ⋮┆----------------------------------------
   home.user.semgrep-rules.php.lang.security.exec-use
      Executing non-constant commands. This can lead to command injection.

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