Dreamhack xss-1

먼저 문제 파일에 있는 app.py를 분석해보았다.

1. 초기 선언 및 FLAG

```python
#!/usr/bin/python3
from flask import Flask, request, render_template
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
import urllib
import os

app = Flask(__name__)
app.secret_key = os.urandom(32)

try:
    FLAG = open("./flag.txt", "r").read()
except:
    FLAG = "[**FLAG**]"
```

파일 flag.txt에서 FLAG를 읽어온다. 파일이 없으면 기본적으로 "[**FLAG**]" 값을 사용한다.

2. read_url 함수

```python
def read_url(url, cookie={"name": "name", "value": "value"}):
    cookie.update({"domain": "127.0.0.1"})
    try:
        service = Service(executable_path="/chromedriver")
        options = webdriver.ChromeOptions()
        for _ in [
            "headless",
            "window-size=1920x1080",
            "disable-gpu",
            "no-sandbox",
            "disable-dev-shm-usage",
        ]:
            options.add_argument(_)
        driver = webdriver.Chrome(service=service, options=options)
        driver.implicitly_wait(3)
        driver.set_page_load_timeout(3)
        driver.get("http://127.0.0.1:8000/")
        driver.add_cookie(cookie)
        driver.get(url)
    except Exception as e:
        driver.quit()
        # return str(e)
        return False
    driver.quit()
    return True
```

이 함수는 Selenium을 이용해 URL을 열고, 주어진 쿠키를 설정한다. 먼저 Service 객체와 ChromeOptions 객체를 사용해 Chrome 브라우저를 실행한다. driver.get(url)로 URL을 요청하고, driver.add_cookie(cookie)를 통해 쿠키를 추가한다. 페이지가 로드된 후 브라우저를 닫으며, 오류 발생 시 예외를 처리하고 False를 반환한다

3. check_xss 함수

```
def check_xss(param, cookie={"name": "name", "value": "value"}):
    url = f"http://127.0.0.1:8000/vuln?param={urllib.parse.quote(param)}"
    return read_url(url, cookie)
```

check_xss 함수는 read_url 함수를 호출하여 vuln 엔드포인트에 접속한다.

4. route /

```
@app.route("/")
def index():
    return render_template("index.html")
```

기본 페이지는 index.html 템플릿을 렌더링하여 반환한다.

5. route /vuln

```
@app.route("/vuln")
def vuln():
    param = request.args.get("param", "")
    return param
```

URL 파라미터 param 값을 반환하는 엔드포인트이다. /vuln 엔드포인트는 입력값을 검증 없이 그대로 반환하므로 여기에 스크립트를 삽입하면 XSS 취약점이 발생할 수 있다.

6. route /flag

```python
@app.route("/flag", methods=["GET", "POST"])
def flag():
    if request.method == "GET":
        return render_template("flag.html")
    elif request.method == "POST":
        param = request.form.get("param")
        if not check_xss(param, {"name": "flag", "value": FLAG.strip()}):
            return '<script>alert("wrong??");history.go(-1);</script>'

        return '<script>alert("good");history.go(-1);</script>'
```

- GET 요청: flag.html 템플릿을 렌더링한다.
- POST 요청: 사용자가 제출한 폼의 param 값을 가져와 check_xss 함수를 호출한다. 성공 시 FLAG 쿠키를 서버에서 설정해 반환한다. 실패하면 "wrong??" 알림이 뜨고, 성공하면 "good" 알림이 뜬다.

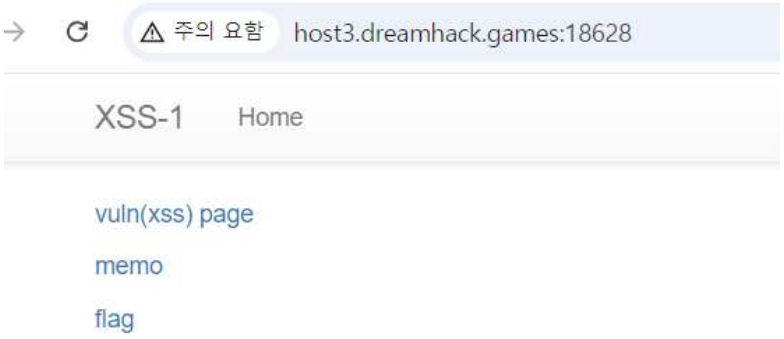
7. route /memo

```python
memo_text = ""


@app.route("/memo")
def memo():
    global memo_text
    text = request.args.get("memo", "")
    memo_text += text + "\n"
    return render_template("memo.html", memo=memo_text)
```

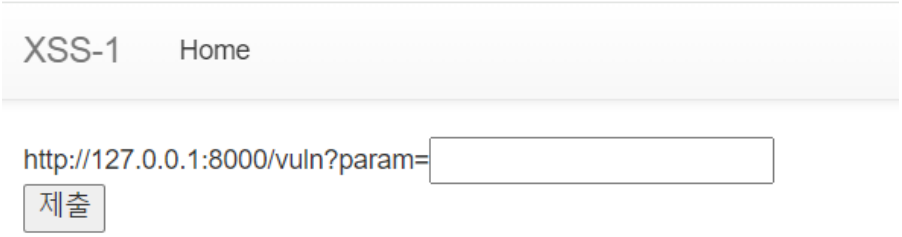
메모를 저장하는 기능이다. GET 요청으로 받은 memo 값을 memo_text에 추가하고, 이를 렌더링하여 메모 내용을 반환한다.

이후 드림핵이 제공하는 서버를 열고 웹 화면을 확인해 보았다.



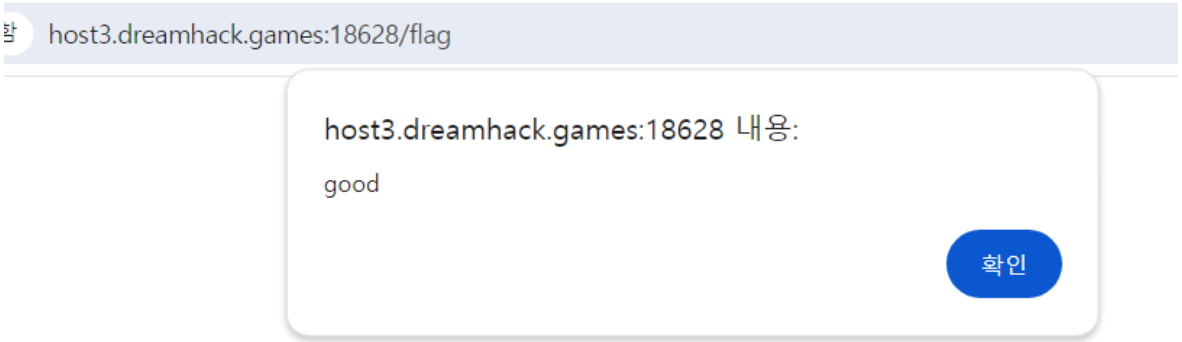
앞에서 소스 코드를 분석했던 것처럼 /vuln, /memo, /flag 가 존재하는 웹사이트이다.

취약한 페이지는 /vuln 페이지이며 이를 이용해 /flag 페이지에 flag의 값을 요청해야 한다. /flag 경로에 param을 입력하면 flag 값이 쿠키로 저장되며, 이를 /memo 페이지에서 출력해야 한다. /memo는 url에 문자열을 입력하면 출력되는 방식이다.



Flag 페이지는 이렇게 되어있다. 이 플래그 페이지에 스크립트를 삽입하여 쿠키를 /memo 페이지의 경로로 보내도록 해야 한다. 다음과 같은 스크립트를 입력한다.

<script>document.location='/memo?memo='+document.cookie;</script>



스크립트를 입력하면 이렇게 good 알림이 뜬다.

그리고 다시 /memo 페이지에 가보면 이렇게 flag가 표시되는 것을 볼 수 있다. 따라서 flag는 **DH{2c01577e9542ec24d68ba0ffb846508e}** 이다.