

most cookies

In this challenge we are given a website that displays information about types of cookies. We are also given a server file that uses flask.

On the website I noticed there was a session cookie stored by the browser that looked encrypted in some way.

I then looked online for tools to decrypt flask session cookies and found a tutorial on hacktricks showing the flask-unsigned command line tool.

<https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/flask>

After installing the tool, I kept getting the error 'command not found'.

In the error message it displayed the file location for the command was not in my PATH environment variable.

I did not know how to append to my PATH at the time, but found a tutorial online showing how to do so. After appending the file location to PATH, the command now worked and I could now begin investigating the cookie.

<https://unix.stackexchange.com/questions/26047/how-to-correctly-add-a-path-to-path>



```
(kali㉿kali)-[~]
$ cookie=eyJ2ZXJ5X2F1dGgiOiJibGFuayJ9.ZVGvNA.RHqQAV12RFXx0Y4WqsRgcStZnng

(kali㉿kali)-[~]
$ echo $cookie
eyJ2ZXJ5X2F1dGgiOiJibGFuayJ9.ZVGvNA.RHqQAV12RFXx0Y4WqsRgcStZnng

(kali㉿kali)-[~]
$ flask-unsigned --decode --cookie $cookie
{'very_auth': 'blank'}

(kali㉿kali)-[~]
$ vim cookielist.txt

(kali㉿kali)-[~]
$ flask-unsigned --wordlist ./cookielist.txt --unsigned --cookie $cookie --no-literal-eval
[*] Session decodes to: {'very_auth': 'blank'}
[*] Starting brute-forcer with 8 threads..
[+] Found secret key after 28 attemptscadamia
b'peanut butter'

(kali㉿kali)-[~]
$ flask-unsigned --sign --cookie '{"very_auth": "admin"}' --secret 'peanut butter'
eyJ2ZXJ5X2F1dGgiOiJhZG1pbjJ9.ZVGwDw.a0Ln5PSffMrusw9e9DCr4PlybVg

(kali㉿kali)-[~]
$
```

I first saved the session cookie to a temporary variable in my terminal so that I would not need to copy and paste it in each command.

Using the flask-unsigned --decode --cookie method, I saw that the plaintext data of the cookie stored a 'very-auth' variable initialized to a value 'blank'.

In the given python source code for the server, I noticed that the server was checking to see if the 'very-auth' cookie was set to the value 'admin'.

The cookie was also being signed with a random value from the array of accepted cookie flavors, so I created a word list of all the cookie types.

Because there were much fewer than 100 flavors I thought it might be efficient to bruteforce the secret of the given session cookie.

After almost no delay, the flask-unsign command was able to decipher the session cookie's signed secret to be 'peanut butter'.

With this knowledge, I could now attempt to create a forged session cookie with the desired 'very-auth' value and known signed secret to see if the value would be accepted by the server to give me authenticated access.

I used the flask-unsign --sign --cookie --secret flags to forge a new session cookie with the 'very-auth' variable set to admin and the signed secret to be 'peanut butter' to mimic the known session cookie and bypass the authentication check.

The top screenshot shows a web browser at the URL `mercury.picoctf.net:53700/display`. The page title is "Most Cookies" with a "Reset" link. A large gray box displays the flag: `picoCTF{pwn_4ll_th3_cook1E5_3646b931}`.

The bottom screenshot shows the Chrome DevTools Storage tab. The left sidebar lists storage areas: Cache Storage, Cookies, Indexed DB, Local Storage, and Session Storage. The "Cookies" section is expanded, showing a table of cookies for the domain `http://mercury.picoctf.net:53700`.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
session	eyJ2ZXJ5X2F1dGgiOiJhZG1pbjU9LjZVGwDw.a0Ln5PSFfMrusw9e9DCr4PlybVg	mercury.pic...	/	Session	70	true	false	None	Mon, 13 Nov 2023 ...

The bottom status bar shows a network request: `GET http://mercury.picoctf.net:53700/favicon.ico` with a status of `[HTTP/1.1 404 NOT FOUND 0ms]`.

After pasting the cookie into my browser and refreshing the webpage, I was given the flag!