ctf web中快速反弹

转载

wssmiss 于 2019-03-25 20:22:40 发布 333 收藏 2

分类专栏： python web ctf 文章标签： ctf web


python 同时被 3 个专栏收录

11 篇文章 1 订阅

订阅专栏


web

11 篇文章 0 订阅

订阅专栏


ctf

40 篇文章 0 订阅

订阅专栏

原文：https://ciphersaw.me/2017/12/16/详解 CTF Web 中的快速反弹 POST 请求/

原文作者:Cipher Saw

0x00 前言

在 CTF Web 的基础题中，经常出现一类题型：在 HTTP 响应头获取了一段有效期很短的 key 值后，需要将经过处理后的 key 值快速 POST 给服务器，若 key 值还在有效期内，则服务器返回最终的 flag，否则继续提示“请再加快速度！！！”

如果还执着于手动地获取 key 值，复制下来对其进行处理，最后用相应的工具把 key 值 POST 给服务器，那么对不起，因为 key 值的有效期一般都在 1 秒左右，除非有单身一百年的手速，否则不要轻易尝试。显然，这类题不是通过纯手工完成的，幸好 Python 提供了简单易用、功能强大的 HTTP 第三方开源库 Requests，帮助我们轻松解决关于 HTTP 的大部分问题。

0x01 Python Requests

关于 Requests 库的详细功能请见官方文档，本文只列出解题中需要用到的部分功能。

安装并导入 requests 模块
在安装了 Python 的终端下输入以下命令安装 requests：

$ pip install requests
安装完使用以下命令导入 requests：

```
>>> import requests
```

发送 GET 请求与 POST 请求
以 Github 官网为例，对其发起 GET 请求;

```
>>> r = requests.get('https://github.com/')
```

对其发起 POST 请求：

```
>>> r = requests.post('https://github.com/')
```

查看请求头
对 Github 官网发起请求，以查看 GET 请求的请求头为例，POST 请求同理：

```
>>> r = requests.get('https://github.com/')
>>> r.request.headers
{'Connection': 'keep-alive', 'Accept-Encoding': 'gzip, deflate',...
```

查看请求头的某一属性：

```
>>> r.request.headers['Accept-Encoding']
'gzip, deflate'
```

查看响应头
对 Github 官网发起请求，以查看 GET 请求的响应头为例，POST 请求同理：

```
>>> r = requests.get('https://github.com/')
>>> r.headers
{'Status': '200 OK', 'Expect-CT': 'max-age=2592000, report-uri=...
```

查看响应头的某一属性：

```
>>> r.headers['Status']
'200 OK'
```

查看响应内容
对 Github 官网发起请求，查看服务器返回页面的内容，以查看 GET 请求的响应内容为例，POST 请求同理：

```
>>> r = requests.get('https://github.com/')
>>> r.text
u'\n\n\n\n\n\n<!DOCTYPE html>\n<html lang="en">\n  <head>\n    <meta charset="utf-8">\n...
```

传递 GET 请求参数
GET 请求参数作为查询字符串附加在 URL 末尾，可以通过 requests.get() 方法中的 params 参数完成。例如，我要构建的 URL 为 https://github.com/?username=ciphersaw&id=1，则可以通过以下代码传递 GET 请求参数：

```
>>> args = {'username': 'ciphersaw', 'id': 1}
>>> r = requests.get('https://github.com/', params = args)
>>> print(r.url)
https://github.com/?username=ciphersaw&id=1
```

其中 params 参数是 dict 类型变量。可以看到，带有请求参数的 URL 确实构造好了，不过注意，这里的 username 和 id 是为了说明问题任意构造的，传入 Github 官网后不起作用，下同。

传递 POST 请求参数
POST 请求参数以表单数据的形式传递，可以通过 requests.post() 方法中的 data 参数完成，具体代码如下：

```
>>> args = {'username': 'ciphersaw', 'id': 1}
>>> r = requests.post('https://github.com/', data = args)
```

其中 data 参数也是 dict 类型变量。由于 POST 请求参数不以明文展现，在此省略验证步骤。

传递 Cookie 参数
如果想传递自定义 Cookie 到服务器，可以使用 cookies 参数。以 POST 请求为例向 Github 官网提交自定义 Cookie（cookies 参数同样适用于 GET 请求）：

```
>>> mycookie = {'userid': '123456'}
>>> r = requests.post('https://github.com/', cookies = mycookie)
>>> r.request.headers
...'Cookie': 'userid=123456',...
```

其中 cookies 参数也是 dict 类型变量。可以看到，POST 请求的请求头中确实包含了自定义 Cookie。

会话对象 Session()
Session 是存储在服务器上的相关用户信息，用于在有效期内保持客户端与服务器之间的状态。Session 与 Cookie 配合使用，当 Session 或 Cookie 失效时，客户端与服务器之间的状态也随之失效。

有关 Session 的原理可参见以下文章：

session的根本原理及安全性
Session原理

requests 模块中的 会话对象 Session() 能够在多次请求中保持某些参数，使得底层的 TCP 连接将被重用，提高了 HTTP 连接的性能。

Session() 的创建过程如下：

```
>>> s = requests.Session()
```

在有效期内，同一个会话对象发出的所有请求都保持着相同的 Cookie，可以看出，会话对象也可以通过 get 与 post 方法发送请求，以发送 GET 请求为例：

```
>>> r = s.get('https://github.com/')
```

题目

【Bugku CTF】 Web —— Web6
此题是上一题的升级版，除了要求快速反弹 POST 请求，还要求所有的请求必须在同一个 Session 内完成，因此会话对象 Session() 就派上用场了。相关链接如下：

题目链接：http://123.206.31.85/challenges#Web6
解题链接：http://120.24.86.145:8002/web6/

进入解题链接，直接查看源码：

```
</br>我感觉你得快点!!!<!-- OK ,now you have to post the margin what you find -->
```

发现 POST 请求参数的键值为 margin，最后看看响应头：

发现 flag 属性，其值同样是一段 base64 编码。这里就不手工解码再提交 POST 请求了，直接用上一题的 Python 脚本试试：

```
此处注意第 8 行的 base64 解码，因为经过第一次 base64 解码后，仍然还是一段 base64 编码，所以要再解码一次。**解题过程中，要自行动手查看每一次解码后的值，才能选择合适的方法去获得最终 key 值。**

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import requests
import base64
url = 'http://120.24.86.145:8002/web6/'
headers = requests.get(url).headers
key = base64.b64decode(base64.b64decode(headers['flag']).decode().split(":")[1])
post = {'margin': key}
print(requests.post(url, data = post).text)
```

结果如下，果然没那么容易得到 flag：

```
我都说了让你快点。。。</br>我感觉你得快点!!!<!-- OK ,now you have to post the margin what you find -->

[Finished in 0.4s]
```

嗯，眉头一紧，发现事情并不简单。下面看看 GET 请求与 POST 请求的请求头与响应头是否内有玄机：

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import requests
import base64
url = 'http://120.24.86.145:8002/web6/'
get_response = requests.get(url)
print('GET Request Headers:\n', get_response.request.headers, '\n')
print('GET Response Headers:\n', get_response.headers, '\n')
key = base64.b64decode(base64.b64decode(get_response.headers['flag']).decode().split(":")[1])
post = {'margin': key}
post_responese = requests.post(url, data = post)
print('POST Request Headers:\n', post_responese.request.headers, '\n')
print('POST Response Headers:\n', post_responese.headers, '\n')
```

不出所料，结果如下，原来是 GET 请求和 POST 请求的响应头都有 Set-Cookie 属性，并且值不相同，即不在同一个会话中，各自响应头中的 flag 值也不等：

```
GET Request Headers:
 {'User-Agent': 'python-requests/2.18.4', 'Accept-Encoding': 'gzip, deflate', 'Accept': '*/*', 'Connection': 'keep-alive'}

GET Response Headers:
 {'Server': 'nginx', 'Date': 'Sun, 17 Dec 2017 04:41:55 GMT', 'Content-Type': 'text/html;charset=utf-8', 'Transfer-Encoding': 'chunked', 'Connection': 'keep-alive',
 'Keep-Alive': 'timeout=60', 'Set-Cookie': 'PHPSESSID=otvnsgn9bthkunc2qgbre6bmmeoh2mnj; path=/; HttpOnly', 'Expires': 'Thu, 19 Nov 1981 08:52:00 GMT', 'Cache-Control':
 'no-store, no-cache, must-revalidate, post-check=0, pre-check=0', 'Pragma': 'no-cache', 'flag': '6LeR55qE6L+Y5LiN6ZSZ77yM57uZ5L2gZmxhZ+WQpzogT1RrNE56UTE=',
 'Content-Encoding': 'gzip'}

POST Request Headers:
 {'User-Agent': 'python-requests/2.18.4', 'Accept-Encoding': 'gzip, deflate', 'Accept': '*/*', 'Connection': 'keep-alive', 'Content-Length': '16', 'Content-Type':
 'application/x-www-form-urlencoded'}

POST Response Headers:
 {'Server': 'nginx', 'Date': 'Sun, 17 Dec 2017 04:41:55 GMT', 'Content-Type': 'text/html;charset=utf-8', 'Transfer-Encoding': 'chunked', 'Connection': 'keep-alive',
 'Keep-Alive': 'timeout=60', 'Set-Cookie': 'PHPSESSID=gmo97t24era0qpuq6hrf4ruqknsemrhu; path=/; HttpOnly', 'Expires': 'Thu, 19 Nov 1981 08:52:00 GMT', 'Cache-Control':
 'no-store, no-cache, must-revalidate, post-check=0, pre-check=0', 'Pragma': 'no-cache', 'flag': '6LeR55qE6L+Y5LiN6ZSZ77yM57uZ5L2gZmxhZ+WQpzogTVRVM016UXg=',
 'Content-Encoding': 'gzip'}

[Finished in 0.4s]
```


接下来引入会话对象 Session()，稍作修改就能保证 GET 请求与 POST 请求在同一个会话中了：

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import requests
import base64
url = 'http://120.24.86.145:8002/web6/'
s = requests.Session()
headers = s.get(url).headers
key = base64.b64decode(base64.b64decode(headers['flag']).decode().split(":")[1])
post = {"margin":key}
print(s.post(url, data = post).text)
```

与上一题代码的区别是：此处用会话对象 Session() 的 get 和 post 方法，而不是直接用 requests 模块里的，这样可以保持 GET 请求与 POST 请求在同一个会话中。将同一会话中的 key 值作为 POST 请求参数提交，最终得到 flag：

```
KEY{111dd62fcd[illegible]}
[Finished in 0.4s]
```

虽然到此即可结束，但为了验证以上两次请求真的在同一会话内，我们再次查看请求头与响应头：

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import requests
import base64
url = 'http://120.24.86.145:8002/web6/'
s = requests.Session()
get_response = s.get(url)
print('GET Request Headers:\n', get_response.request.headers, '\n')
print('GET Response Headers:\n', get_response.headers, '\n')
key = base64.b64decode(base64.b64decode(get_response.headers['flag']).decode().split(":")[1])
post = {'margin': key}
post_responese = s.post(url, data = post)
print('POST Request Headers:\n', post_responese.request.headers, '\n')
print('POST Response Headers:\n', post_responese.headers, '\n')
```

结果如下，GET 请求中响应头的 Set-Cookie 属性与 POST 请求中请求头的 Cookie 属性相同，表明两次请求确实在同一会话中。

```
GET Request Headers:
 {'User-Agent': 'python-requests/2.18.4', 'Accept-Encoding': 'gzip, deflate', 'Accept': '*/*', 'Connection': 'keep-alive'}

GET Response Headers:
 {'Server': 'nginx', 'Date': 'Sun, 17 Dec 2017 05:12:09 GMT', 'Content-Type': 'text/html;charset=utf-8', 'Transfer-Encoding': 'chunked', 'Connection': 'keep-alive',
 'Keep-Alive': 'timeout=60', 'Set-Cookie': 'PHPSESSID=1mhsrpj3cjspd6agbq6i3ptei8od051b; path=/; HttpOnly', 'Expires': 'Thu, 19 Nov 1981 08:52:00 GMT', 'Cache-Control':
 'no-store, no-cache, must-revalidate, post-check=0, pre-check=0', 'Pragma': 'no-cache', 'flag': '6LeR55qE6L+Y5LiN6ZSZ77yM57uZ5L2gZmxhZ+WQpzogTlRZek5qazU=',
 'Content-Encoding': 'gzip'}

POST Request Headers:
 {'User-Agent': 'python-requests/2.18.4', 'Accept-Encoding': 'gzip, deflate', 'Accept': '*/*', 'Connection': 'keep-alive', 'Cookie':
 'PHPSESSID=1mhsrpj3cjspd6agbq6i3ptei8od051b', 'Content-Length': '13', 'Content-Type': 'application/x-www-form-urlencoded'}

POST Response Headers:
 {'Server': 'nginx', 'Date': 'Sun, 17 Dec 2017 05:12:09 GMT', 'Content-Type': 'text/html;charset=utf-8', 'Transfer-Encoding': 'chunked', 'Connection': 'keep-alive',
 'Keep-Alive': 'timeout=60', 'Expires': 'Thu, 19 Nov 1981 08:52:00 GMT', 'Cache-Control': 'no-store, no-cache, must-revalidate, post-check=0, pre-check=0', 'Pragma':
 'no-cache', 'Content-Encoding': 'gzip'}

[Finished in 0.4s]
```


既然只需要保持两次请求中 Cookie 属性相同，那能不能构造 Cookie 属性通过普通的 get 与 post 方法完成呢？答案是可以的。请见如下代码：

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import requests
import base64
url = 'http://120.24.86.145:8002/web6/'
headers = requests.get(url).headers
key = base64.b64decode(base64.b64decode(headers['flag']).decode().split(":")[1])
post = {"margin": key}
PHPSESSID = headers["Set-Cookie"].split(";")[0].split("=")[1]
cookie = {"PHPSESSID": PHPSESSID}
print(requests.post(url, data = post, cookies = cookie).text)
```

Line 10：获得 GET 请求响应头中 Set-Cookie 属性的 PHPSESSID 值，该语句如何构造请自行分析 Set-Cookie 属性字符串值的结构；
Line 11：构造 POST 请求中 cookies 参数的 dict 类型变量；
Line 12：提交带有 data 参数与 cookies 参数的 POST 请求，最终打印响应页面的内容。
毫无疑问，以上代码的结果也是最终的 flag。

【Bugku CTF】Web —— 秋名山老司机
前面两题均是对响应头中与flag相关的属性做解码处理，然后快速反弹一个 POST 请求得到 flag 值。而本题要求计算响应内容中的表达式，将结果用 POST 请求反弹回服务器换取 flag 值。实际上换汤不换药，依旧用 Python 写个脚本即可解决。

题目链接：http://123.206.31.85/challenges#秋名山老司机
解题链接：http://120.24.86.145:8002/qiumingshan/

打开解题连接，老规矩先看源码：

```
<head>
<title>下面的表达式的值是秋名山的车速</title>
<meta charset="UTF-8">
</head>
<p>亲请在2s内计算老司机的车速是多少</p>
<div>1007975461*2058338112+1264576414*458012155*1308030357-905097817+2111144050-1522933939-1959682295-1361290434+81796428=?;</div>
<style>
div,p{
text-align: center;
margin: 0 auto;
}
</style>
```

题意很明确，要求在 2 秒内计算给出表达式的值…呃，然后呢？刷新页面再看看，噢噢，然后再将计算结果用 POST 请求反弹回服务器，请求参数的 key 值为 **value**：

```
Give me value post about 839245726+1486315050-1811716604+2144233750*1680643035*821662606*1213893054-335401769-1033365068*1295689483*773937094=?
```

从页面内容中截取表达式，可以用 string 自带的 split() 函数，但必须先要知道表达式两边的字符串，以其作为分隔符；也可以用正则表达式，仅需知道表达式本身的特征即可。此处用正则表达式更佳。先放上题解脚本，再来慢慢解析：

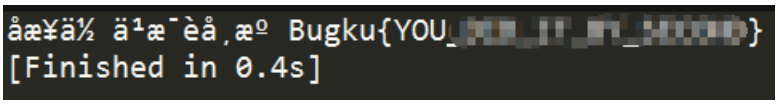
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import requests
import re
url = 'http://120.24.86.145:8002/qiumingshan/'
s = requests.Session()
source = s.get(url)
expression = re.search(r'(\d+[+\-*])+(\d+)', source.text).group()
result = eval(expression)
post = {'value': result}
print(s.post(url, data = post).text)
```

有关 requests 的部分此处不细讲，唯一要注意的是，与上一篇 writeup 一样，要利用会话对象 Session()，否则提交结果的时候，重新生成了一个新的表达式，结果自然错误。

Line 9：是利用正则表达式截取响应内容中的算术表达式。首先引入 re 模块，其次用 search() 匹配算术表达式，匹配成功后用 group() 返回算术表达式的字符串。（想掌握正则表达式，还是要多看、多想、多练，毕竟应用场合非常之广）

search() 的第一个参数是匹配的正则表达式，第二个参数是要匹配的字符串。其中 \d+代表一个或多个数字；*[+-] 匹配一个加号，或一个减号，或一个乘号，注意减号在中括号内是特殊字符，要用反斜杠转义；(\d+[+-])*+代表一个或多个由数字与运算符组成的匹配组；最后再加上剩下的一个数字 (\d+)。

Line 11：在获得算术表达式的字符串后，直接利用 Python 的内建方法 eval() 来计算出结果，简单、暴力、快捷。
执行完上述脚本，就有一定的概率可以获得 flag 了：



原始链接

作者：Cipher Saw