CTF wp 2018“骇极杯”全国大学生网络安全邀请赛暨第四届上海市大学生网络安全大赛线上赛 writeup

原创

whiteQ777 于 2018-11-05 10:21:10 发布 2993 收藏
分类专栏： CTF 文章标签： CTF

本文链接：https://blog.csdn.net/Thea_1207/article/details/83744517

为0x00 签到题

操作内容：

| 登陆比赛界面上去看到签到题，一般情况下签到题是没有难度的题目给定一串字符
MZWGCZ33GM2TEMRSMQZTALJUGM4WKLJUMFTGELJZGFTDILLBMJSWEYZXGNTGKMBVMN6Q

Base32 一闪而过，比的就是手速，解码拿到flag。

FLAG 值：

标志{35222d30-439e-4afb-91F4-abebc73fe05c}

0x01 easy_py

操作内容：

| 首先查看二进制文件

03f3 0d0a bebc ce5b 6300 0000 0000 0000

000f 0000 0040 0000 0073 b200 0000 7106

0064 2333 7109 0064 0000 6401 0064 0200

这里的2333很可疑，通过尝试修改成1000，进行反编译

```
 1       0 JUMP_ABSOLUTE 6   '到6'
         3 LOAD_CONST 16   '对'
         6 JUMP_ABSOLUTE 9   '到9'
         9 LOAD_CONST 0   ''
        12 LOAD_CONST 1   10
        15 LOAD_CONST 2   7
        18 LOAD_CONST 3   1
        21 LOAD_CONST 4   29
        24 LOAD_CONST 5   14
```

```
          27 LOAD_CONST          2 7
          30 LOAD_CONST          6 22
          33 LOAD_CONST          6 22
          36 LOAD_CONST          7 31
          39 LOAD_CONST          8 57
          42 LOAD_CONST          9 30
          45 LOAD_CONST         10 9
          48 LOAD_CONST         11 52

2         51 LOAD_CONST         12 27
          54 BUILD_LIST_15      15
          57 STORE_NAME          0 'cmp'

3         60 LOAD_NAME           1 'raw_input'
          63 CALL_FUNCTION_0     0

4         66 STORE_NAME          2 'flag'
          69 LOAD_CONST          0 ''
          72 STORE_NAME          3 'm'
          75 SETUP_LOOP         91 'to 169'
          78 LOAD_NAME           2 'flag'
          81 GET_ITER
          82 FOR_ITER           83 'to 168'
          85 STORE_NAME          4 'i'
          88 LOAD_NAME           5 'ord'
          91 LOAD_NAME           4 'i'
          94 CALL_FUNCTION_1     1
          97 UNARY_INVERT
          98 LOAD_CONST         13 102
         101 BINARY_AND
         102 LOAD_NAME           5 'ord'
```

```
        105  LOAD_NAME            4  'i'
        108  CALL_FUNCTION_1      1
        111  LOAD_CONST           18  -103
        114  BINARY_AND
        115  BINARY_OR
        116  STORE_NAME           4  'i'
        119  LOAD_NAME            4  'i'
        122  LOAD_NAME            0  'cmp'
        125  LOAD_NAME            3  'm'
        128  BINARY_SUBSCR
        129  COMPARE_OP           2  '=='
        132  POP_JUMP_IF_FALSE    144  'to 144'
        135  LOAD_NAME            3  'm'

  8     138  UNARY_NEGATIVE
        139  LOAD_CONST           14  -1
        142  BINARY_ADD
        143  UNARY_NEGATIVE

 10     144  STORE_NAME           3  'm'
        147  JUMP_BACK            73  'to 73'
        150  CONTINUE             73  'to 73'
        153  LOAD_CONST           15  'wrong'
        156  PRINT_ITEM
        157  PRINT_NEWLINE_CONT
        158  LOAD_NAME            6  'exit'
        161  CALL_FUNCTION_0      0
        164  POP_TOP
        165  JUMP_BACK            73  'to 73'
        168  POP_BLOCK
      169_0  COME_FROM             '75'
```

```
169  LOAD_CONST          16  'right'

172  PRINT_ITEM

173  PRINT_NEWLINE_CONT
```

Parse error at or near `LOAD_CONST' instruction at offset 3

得到的是python的字节码，通过手动模拟汇编，写出POC

```
const = [0, 10, 7, 1, 29, 14, 7, 22, 22, 31, 57, 30, 9, 52, 27]

flag = ""

for i in const:
    flag = flag + chr((~i) & 0x66 | (-103 & i))

print flag
```

FLAG值：

flag{happy_xoR}

0x02 web1
操作内容：

打开网页什么都没有，很自然的打开控制台看一下源码。

里面写着让我们找robots.txt，所以在url后添加robots.txt

显示source.php和flag.php

我先打开flag.php检查了一下，什么都没有。然后打开source.php

在源码里写着需要post参数admin，在测试注入的时候发现1有回显，显示只有本地才能拿到flag

第一反应是X-Forwarded-For绕过，结果不行，上网找了一排，挨个试了一下，X-Clien-IP，可以绕过。

之后源码里跳出了一张img，分号比较显眼，试了一下把照片下载下来，结果显示301，这条路不行。

接着就是想这个分号，测试了一下，可以通过构造url来请求到资源，于是就想到是个ssrf漏洞。

接着构造请求访问到flag.php，可以在源码里看到注释起来的flag。

FLAG值：

flag{11d340f0-0740-4229-9ff0-c2fca56f0706}

0x03 web2
操作内容：

存在源码泄露

然后使用 vi命令恢复得到

```php
<?php
error_reporting(0);
class come {
  private $method;
  private $args;
  function __construct($method, $args) {
    $this->method = $method;
    $this->args = $args;
  }
  function __wakeup() {
    foreach ($this->args as $k => $v) {
      $this->args[$k] = $this->waf(trim($v));
    }
  }
  function waf($str) {
    $str = preg_replace("/[<>*;|?\n ]/", "", $str);
    $str = str_replace('flag', '', $str);
    return $str;
  }
  function echo ($host) {
```



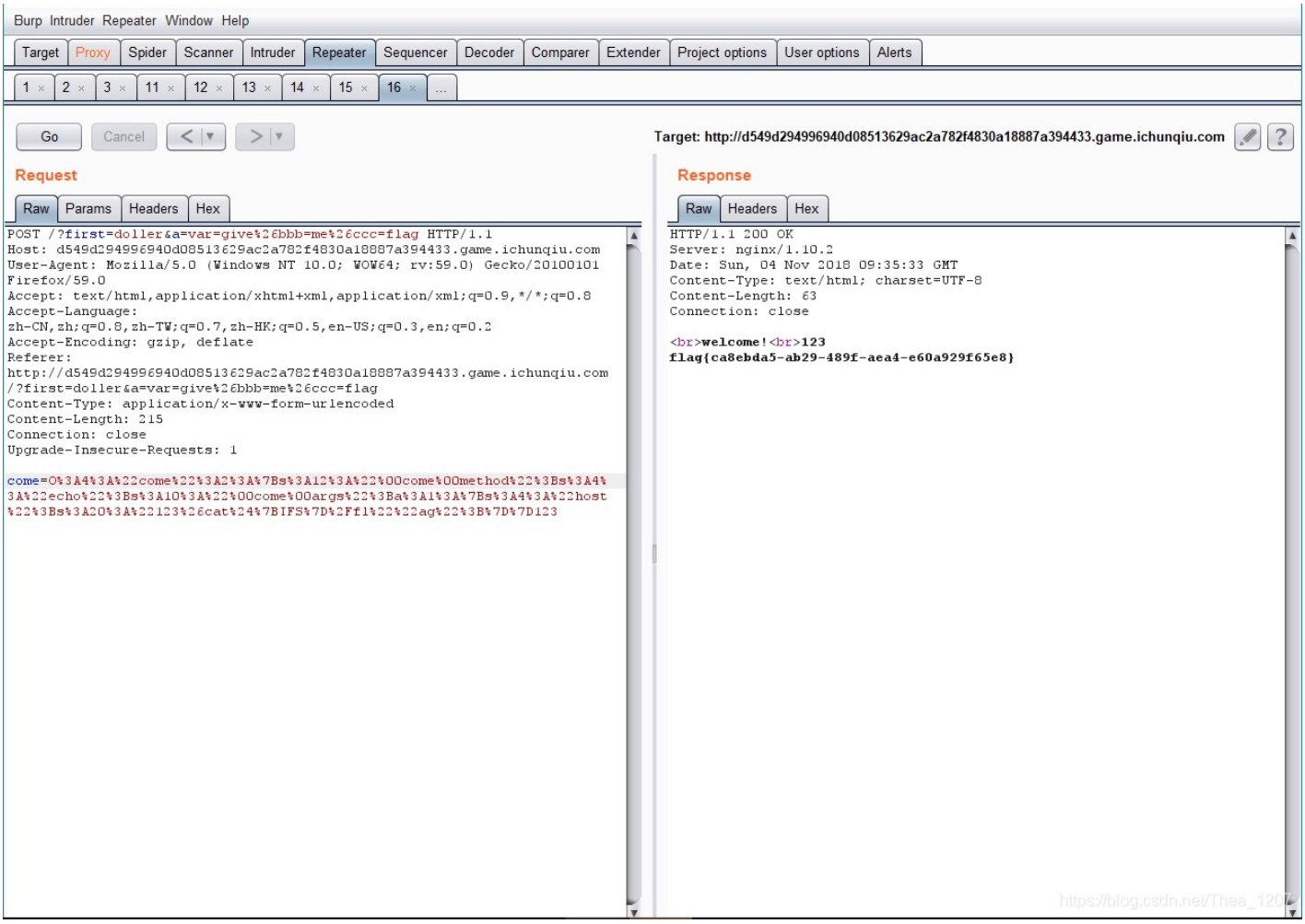
```php
$first = 'hi';
$var = 'var';
$bbb = 'bbb';
$ccc = 'ccc';
$i = 1;
foreach ($_GET as $key => $value) {
  if ($i === 1) {
    $i++;
    $$key = $value;
  } else {
    break;
  }
}
if ($first === "doller") {
  @parse_str($_GET['a']);
```

```php
if ($var === "give") {
  if ($bbb === "me") {
    if ($ccc === "flag") {
      echo "<br>welcome!<br>";
      $come = @$_POST['come'];
      unserialize($come);
    }
  } else {
    echo "<br>think about it<br>";
  }
} else {
  echo "NO";
}
} else {
  echo "Can you hack me?<br>";
}
```


可得知是php反序列化 通过变量替换绕过

get 传参部分：first=doller&a=var=give%26bbb=me%26ccc=flag

get部分有parsestr函数来完成变量覆盖的操作



然后post部分

come=O%3A4%3A%22come%22%3A2%3A%7Bs%3A12%3A%22%00come%00method%22%3Bs%3A4%3A%

FLAG值：

flag{ca8ebda5-ab29-489f-aead-e60a929f65e8 }

0x04 cpp

操作内容：

| 导入IDA进行分析

在0x4011a函数中发现它首先将flag进行了（（flag [i] >> 6）| 4 * flag [i]）^ i

处理

```
      i < (unsigned __int64)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(buff);
      ++i )
{
  v1 = (_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](buff, i);
  v2 = 4 * *(char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](buff, i);
  *v1 = ((*(_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](buff, i) >> 6) | v2) ^ i;
}
v3 = (const char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(buff);
if ( strcmp(&s1, v3) == 0 )
{
  half_4012CE(buff);
  exit(0);
}
```

然后在401332函数中又进行了如下操作

flag [j] =（flag [j-1] | flag [j]）＆～（flag [j-1]＆flag [j]）

循环三次，等价于

flag [i] = flag [i] ^ flag [i-1]

然后在401332函数中又进行了如下操作

flag [j] =（flag [j-1] | flag [j]）＆～（flag [j-1]＆flag [j]）

循环三次，等价于

flag [i] = flag [i] ^ flag [i-1]

```
key1 = [-103, -80, -121, -98, 112, -24, 65, 68, 5, 4, 139, 154, 116, 188, 85, 88, 181, 97, 142, 5

for x in range(0,3):

  for i in range(len(key1)-1 , 0,-1):
    key1[i]=key1[i]^key1[i-1]

for x in range(0,len(key1)):
  key1[x] = keyq[x]^x

for i in range(0,len(key1)):
  for c in string.printable:
    if((((ord(c) >> 6) | (4 * ord(c)) ^ i) == key1[i]):
      key1[i]=ord(c)

print key1
```


可得到标志

FLAG 值：

标志{} W0w_y0u_m4st3r_C_p1us_p1us

0x05 What's_it

操作内容：

| 第一步：破解暴力的代码金钥得到爆破出来的查询查询结果的英文：ozulmt

```
for x1 in range(97.123):
    for x2 in range(97.123):
        for x3 in range(97.123):
            for x4 in range(97.123):
                for x5 in range(97.123):
                    for x6 in range(97.123):
                        s = hashlib.md5(bytes([x1,x2,x3,x4,x5,x6,])).hexdigest()
                        v15 = 0
                        v14 = 0
                        for i in range(32):
                            if s[i] == '0':
                                v15 = v15 + 1
                                v14 = v14 + i
                        if 10*v15 +v14 == 403:
                            print bytes([x1,x2,x3,x4,x5,x6,])
```


第二步：检查函数的算法检查输入格式，再剪裁输出，我们利用伪随机数生成一串字符，将它与剪裁得到的输入内容进行比较。将生成后的字符串dump下来，整理格式，便得到标志

```
memset(
      (void *)((unsigned int)&v3[4]) &0xFFFFFFFC,
      0,
      4*(()unsigned int)&v3-((unsigned int)&v3[4]&&0xFFFFFFFC]+33 & &0xFFFFFFFC) >>2));
printfs 1
scanfs(input)
checkht(input)
for (i=0;i<=3,i++):
    v3[i]=ASCII[rands(16)]
v8 = 0
for (j = 0; ;j++):
    v1= strlens(input)
    if (v1<=k):
        break
    if ( input[k] == v3[k]):
        ++v8
    else:
        v8 +=100
if (v8==32):
    v8 = 2
else:
    v8=3
return printf v8
```


FLAG 值：

标志{a197b847-7092-53a4-7c41-bc7d6d52e69d}