# 红帽杯2019 Easyre wp

原创

xi@0ji233 于 2021-05-17 21:46:16 发布 82 收藏



---

title: WP:红帽杯2019 easyRe
date: 2021-05-6 1:24:00
tags:

- writeup
- binary security
- reverse analysis
  comments: true
  categories:
- ctf
- reverse

---

`pwn`题做完`re`题当然也不能少，`buu`上除了那些水题，开始做一些带点技术含量的题目了。红帽杯2019的`easyRE`上来就是`800KB`的`elf`文件，看起来就很有技术含量的样子。其实看到大文件不用怕，函数多也不用怕，因为需要分析的函数一定只有几个，如果你点进去调用了大部分你没见过的函数，还有十分复杂的`goto`关系，那么这个函数你可以直接当他不存在，这是我自己得到的一个结论，不一定对，但是可以应付大部分的题目。废话不多说进入正题：buuctf 2019红帽杯easyRE

所有二进制安全相关的题目字符串一定是切入点，因为它可读。做`pwn`题你就找`flag`，`bin/sh`，当然这是签到-难度的题目才会有的字符串。做逆向题的话，看到很长的64位或者65位大小写字母和数字和一些其它字符组成的表，那么直接考虑`base64`，然后就是一些带`flag`的字符串或者是什么`right`，`correct`，`congratulate`之类判断正误的话，那么多半也是以调用这个字符串的函数为中心去分析。好的，打开先看一下，看到有很多很多的函数，先不慌(实则慌的一批)，冷静地先摁一个`shift+F12`查看字符串。

| Address | Length | Type | String |
| --- | --- | --- | --- |
| .rodata:00000... | 000002E9 | C | Vm0wd2VHUXhTWGhpUm1SWVYwZDRWVll3Wkc5WFJsbDNXa1pPVlUxV2NIcFhhMk0xVmp... |
| .rodata:00000... | 0000000A | C | continue! |
| .rodata:00000... | 00000010 | C | You found me!!! |
| .rodata:00000... | 00000009 | C | bye bye~ |
| .rodata:00000... | 00000041 | C | ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/ |
| .rodata:00000... | 00000014 | C | ../csu/libc-start.c |
| .rodata:00000... | 00000017 | C | FATAL: kernel too old\n |
| .rodata:00000... | 00000030 | C | __ehdr_start.e_phentsize == sizeof *GL(dl_phdr) |
| .rodata:00000... | 00000028 | C | FATAL: cannot determine kernel version\n |
| .rodata:00000... | 00000027 | C | unexpected reloc type in static binary |
| .rodata:00000... | 00000013 | C | generic_start_main |
| .rodata:00000... | 0000000A | C | /dev/full |
| .rodata:00000... | 0000000A | C | /dev/null |
| .rodata:00000... | 00000035 | C | cannot set %fs base address for thread-local storage |
| .rodata:00000... | 00000029 | C | %s%s%s:%u: %s%sAssertion `%s' failed.\n%n |
| .rodata:00000... | 00000013 | C | Unexpected error.\n |
| .rodata:00000... | 0000000F | C | OUTPUT_CHARSET |
| .rodata:00000... | 00000009 | C | charset= |
| .rodata:00000... | 00000009 | C | LANGUAGE |
| .rodata:00000... | 00000006 | C | POSIX |
| .rodata:00000... | 00000012 | C | /usr/share/locale |
| .rodata:00000... | 00000009 | C | messages |
| .rodata:00000... | 00000012 | C | /usr/share/locale |
| .rodata:00000... | 0000000E | C | /locale.alias |
| .rodata:00000... | 0000000C | C | LC_MESSAGES |

一眼就可以看道有一个经典的`base64`表，还有一句话`you found me`(你找到我了)，像极了`flag`跟我们的对话，那么话不多说，点进去找到这个函数看看到底在哪里发出了这样的感叹。

```
__int64 sub_4009C6()
{
  __int64 result; // rax
  int i; // [rsp+Ch] [rbp-114h]
  __int64 v2; // [rsp+10h] [rbp-110h]
  __int64 v3; // [rsp+18h] [rbp-108h]
  __int64 v4; // [rsp+20h] [rbp-100h]
  __int64 v5; // [rsp+28h] [rbp-F8h]
  __int64 v6; // [rsp+30h] [rbp-F0h]
  __int64 v7; // [rsp+38h] [rbp-E8h]
  __int64 v8; // [rsp+40h] [rbp-E0h]
  __int64 v9; // [rsp+48h] [rbp-D8h]
  __int64 v10; // [rsp+50h] [rbp-D0h]
  __int64 v11; // [rsp+58h] [rbp-C8h]
  char v12[13]; // [rsp+60h] [rbp-C0h] BYREF
  char v13[4]; // [rsp+6Dh] [rbp-B3h] BYREF
  char v14[19]; // [rsp+71h] [rbp-AFh] BYREF
  char v15[32]; // [rsp+90h] [rbp-90h] BYREF
  int v16; // [rsp+B0h] [rbp-70h]
  char v17; // [rsp+B4h] [rbp-6Ch]
  char v18[72]; // [rsp+C0h] [rbp-60h] BYREF
  unsigned __int64 v19; // [rsp+108h] [rbp-18h]

  v19 = __readfsqword(0x28u);
  qmemcpy(v12, "Iodl>Qnb(ocy", 12);
  v12[12] = 127;
  qmemcpy(v13, "y.i", 3);
  v13[3] = 127;
  qmemcpy(v14, "d`3w}wek9{iy=~yL@EC", sizeof(v14));
  memset(v15, 0, sizeof(v15));
  v16 = 0;
  v17 = 0;
```

```
  sub_4406E0(0LL, v15, 37LL);
  v17 = 0;
  if ( sub_424BA0(v15) == 36 )
  {
    for ( i = 0; i < (unsigned __int64)sub_424BA0(v15); ++i )
    {
      if ( (unsigned __int8)(v15[i] ^ i) != v12[i] )
      {
        result = 4294967294LL;
        goto LABEL_13;
      }
    }
    sub_410CC0("continue!");
    memset(v18, 0, 0x40uLL);
    v18[64] = 0;
    sub_4406E0(0LL, v18, 64LL);
    v18[39] = 0;
    if ( sub_424BA0(v18) == 39 )
    {
      v2 = sub_400E44(v18);
      v3 = sub_400E44(v2);
      v4 = sub_400E44(v3);
      v5 = sub_400E44(v4);
      v6 = sub_400E44(v5);
      v7 = sub_400E44(v6);
      v8 = sub_400E44(v7);
      v9 = sub_400E44(v8);
      v10 = sub_400E44(v9);
      v11 = sub_400E44(v10);
      if ( !(unsigned int)sub_400360(v11, off_6CC090) )
      {
        sub_410CC0("You found me!!!");
        sub_410CC0("bye bye~");
      }
      result = 0LL;
    }
    else
    {
      result = 4294967293LL;
    }
  }
  else
  {
    result = 0xFFFFFFFFLL;
  }
LABEL_13:
  if ( __readfsqword(0x28u) != v19 )
    sub_444020();
  return result;
}
```

找到这个函数之后就是以这个为主去分析了，由于这个函数特别不友好，我们得去看看这个函数调用的其它那么多sub函数有没有我们认识的熟悉的函数。其实第一点很容易可以看出`sub_410CC00`非常像输出一句话，也就是`puts`函数，当然把它理解为`printf`也没有关系，不影响。根据代码段

```
v18[39] = 0;
if ( sub_424BA0(v18) == 39 )
```

函数参数为字符指针，返回一个数值，是不是像极了strlen函数？

`sub_4406E0(0LL, v15, 37LL);`这个函数，第一个参数0，第二个参数，字符指针，第三个参数一个整型变量，其实也不难判断出是`read`函数，这个靠自己积累多了，看道这种形式就知道是这个函数。这么讲有些人可能懵懵的，那我这么说，`sub_4105A6("%d%d",&v1,&v2)`是什么函数？你会很快看出这就是一个`scanf`，除了`scanf`还有哪个函数写得出这种形式啊？喏，道理一样的。那么我们把那些函数重命名回去看看整体观感好了不少。

```
__int64 sub_4009C6()
{
  __int64 result; // rax
  int i; // [rsp+Ch] [rbp-114h]
  __int64 v2; // [rsp+10h] [rbp-110h]
  __int64 v3; // [rsp+18h] [rbp-108h]
  __int64 v4; // [rsp+20h] [rbp-100h]
  __int64 v5; // [rsp+28h] [rbp-F8h]
  __int64 v6; // [rsp+30h] [rbp-F0h]
  __int64 v7; // [rsp+38h] [rbp-E8h]
  __int64 v8; // [rsp+40h] [rbp-E0h]
  __int64 v9; // [rsp+48h] [rbp-D8h]
  __int64 v10; // [rsp+50h] [rbp-D0h]
  __int64 v11; // [rsp+58h] [rbp-C8h]
  char v12[13]; // [rsp+60h] [rbp-C0h] BYREF
  char v13[4]; // [rsp+6Dh] [rbp-B3h] BYREF
  char v14[19]; // [rsp+71h] [rbp-AFh] BYREF
  char v15[32]; // [rsp+90h] [rbp-90h] BYREF
  int v16; // [rsp+B0h] [rbp-70h]
  char v17; // [rsp+B4h] [rbp-6Ch]
  char v18[72]; // [rsp+C0h] [rbp-60h] BYREF
  unsigned __int64 v19; // [rsp+108h] [rbp-18h]

  v19 = __readfsqword(0x28u);
  qmemcpy(v12, "Iodl>Qnb(ocy", 12);
  v12[12] = 127;
  qmemcpy(v13, "y.i", 3);
  v13[3] = 127;
  qmemcpy(v14, "d`3w}wek9{iy=~yL@EC", sizeof(v14));
  memset(v15, 0, sizeof(v15));
  v16 = 0;
  v17 = 0;
  read(0LL, v15, 37LL);
  v17 = 0;
  if ( strlen(v15) == 36 )
  {
    for ( i = 0; i < (unsigned __int64)strlen(v15); ++i )
    {
      if ( (unsigned __int8)(v15[i] ^ i) != v12[i] )
      {
        result = 4294967294LL;
        goto LABEL_13;
      }
    }
    puts("continue!");
    memset(v18, 0, 0x40uLL);
    v18[64] = 0;
    read(0LL, v18, 64LL);
    v18[39] = 0;
    if ( strlen(v18) == 39 )
    {
      v2 = sub_400E44(v18);
```

```
      v3 = sub_400E44(v2);
      v4 = sub_400E44(v3);
      v5 = sub_400E44(v4);
      v6 = sub_400E44(v5);
      v7 = sub_400E44(v6);
      v8 = sub_400E44(v7);
      v9 = sub_400E44(v8);
      v10 = sub_400E44(v9);
      v11 = sub_400E44(v10);
      if ( !(unsigned int)sub_400360(v11, off_6CC090) )
      {
        puts("You found me!!!");
        puts("bye bye~");
      }
      result = 0LL;
    }
    else
    {
      result = 4294967293LL;
    }
  }
  else
  {
    result = 0xFFFFFFFFLL;
  }
LABEL_13:
  if ( __readfsqword(0x28u) != v19 )
    sub_444020();
  return result;
}
```

你会突然自言自语，”这两个是一个函数？“，不要慌，小场面，小场面。那么这么之后就看看主要的内容，首先一个`for`循环映入眼帘，发现我们输入的`v15[i]^i`如果不等于`v12[i]`那么就会直接跳到最后一行结束程序，那么我们肯定要看看不让它跳转的输入语句是个什么。

```
  char v12[13]; // [rsp+60h] [rbp-C0h] BYREF
  char v13[4]; // [rsp+6Dh] [rbp-B3h] BYREF
  char v14[19]; // [rsp+71h] [rbp-AFh] BYREF
  char v15[32]; // [rsp+90h] [rbp-90h] BYREF
```

这个事实上是按照栈的顺序排列的，也就是说`v12`在靠近栈顶的地方，`v15`在靠近栈底的地方。

那么`v15`看着没有`37`的大小，把它溢出一下用用别人的不就好了嘛，别人反正在我用的时候它也不用，然后那些`v12,v13,v14`都可以看成首尾相连的，因为它们的栈地址本来就挨着的。然后那些都是有赋初值的，那么我们写一个脚本跑一下看看这串字符是什么。

```cpp
#include<bits/stdc++.h>
using namespace std;
char res[100];
int main(){
    string s=(string)"Iodl>Qnb(ocy\x7f"+(string)"y.i\x7f"+(string)"d`3w}wek9{iy=~yL@EC" ;
    for(int i=0;i<s.length();i++)
        s[i]^=i;
    cout<<s;
}
```

```
C:\Users\ZX\Desktop\C++\1.exe
Info:The first four chars are `flag`
--------------------------------
Process exited after 0.01787 seconds with return value 0
请按任意键继续. . .
```

它告诉你了一串信息：前四个字符是`flag`你花了这么久时间破解出来的一串字符，它肯定有用，先留着，后面分析肯定要用。

然后第二个引入眼帘的就是10个一模一样的函数了，点进去发现它引用了base64的那张表，而且很明显看见3,4之类在base64加密解密很常见的数字，那么不用细看了，肯定base64解密。然后看看发现读入了40个字符，然后后面还有一个函数`!(unsigned int)sub_400360(v11, off_6CC090)`稍微再熟悉点就会发现肯定是`strcmp`函数，常用格式嘛`if(!strcmp)`，那么就是它解密了10次变成长度39的字符串，这里真的不用考虑这个函数是加密的情况，总不可能加密了，这个长度才40，算一算也很清楚。那么如果是解密的话，原字符串长度大概在710长度左右，公式：`40*((4/3)**10)`算得的。这么长的字符串，刚刚那个第一个就是，大概长度就是710，很符合我们的需求，写exp对它10次base64解密得到一个网址，然后你就发现自己被骗了，那里没有`flag`，然后就会发现这里没什么地方可以分析了。

然后你可以看到刚刚那一串很长的字符串上面还有一串字符串。

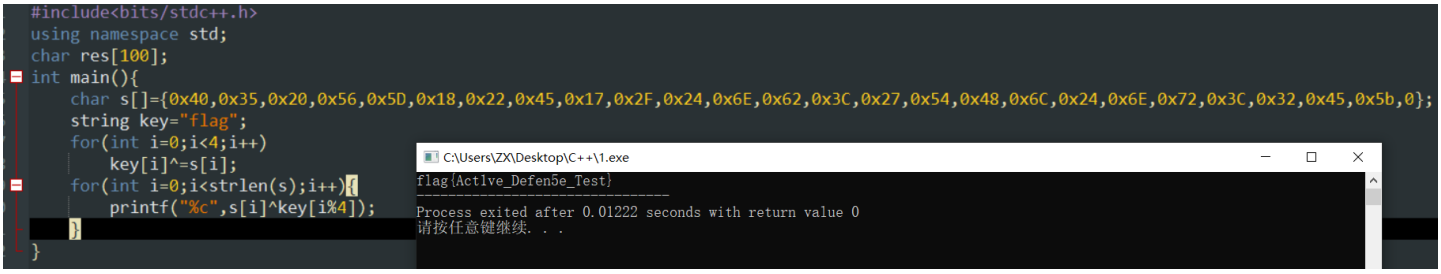
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-OVjxMy8f-1621259160378)(https://i.loli.net/2021/05/06/TdMyJQlqXgPu1WF.png)]

一路跟踪它发现来到了另一个函数`sub_400D35`可以很明显看到`102`,`103`它们分别是`f`,`g`的ascii码值，并且刚好下标在0和3，那么回想前面给的信息，直接断定这个字符串是`flag`。

```
00006CC0A0 byte_6CC0A0     db 40h                  ; DATA XREF:
00006CC0A0                                         ; sub_400D35
00006CC0A1                 db  35h ; 5
00006CC0A2                 db  20h
00006CC0A3 byte_6CC0A3     db 56h                  ; DATA XREF:
00006CC0A4                 db  5Dh ; ]
00006CC0A5                 db  18h
00006CC0A6                 db  22h ; "
00006CC0A7                 db  45h ; E
00006CC0A8                 db  17h
00006CC0A9                 db  2Fh ; /
00006CC0AA                 db  24h ; $
00006CC0AB                 db  6Eh ; n
00006CC0AC                 db  62h ; b
00006CC0AD                 db  3Ch ; <
00006CC0AE                 db  27h ; '
00006CC0AF                 db  54h ; T
00006CC0B0                 db  48h ; H
00006CC0B1                 db  6Ch ; l
00006CC0B2                 db  24h ; $
00006CC0B3                 db  6Eh ; n
00006CC0B4                 db  72h ; r
00006CC0B5                 db  3Ch ; <
00006CC0B6                 db  32h ; 2
00006CC0B7                 db  45h ; E
00006CC0B8                 db  5Bh ; [
```

然后后面对这个字符串动手动脚的，那么找到了长度为25的一个字符串，根据它的加密规则：每四位轮换`v2[4]`对它异或运算。甭管`v2`是啥，它和这个字符开头异或肯定是`flag`这个是铁的道理，因为前面破解出了一个提示。

那么咱们写脚本解一下这串字符。直接得到`flag`




exp:

```
#include<bits/stdc++.h>
using namespace std;
int main(){
 char s[]={0x40,0x35,0x20,0x56,0x5D,0x18,0x22,0x45,0x17,0x2F,0x24,0x6E,0x62,0x3C,0x27,0x54,0x48,0x6C,0x24,0x6E,0
x72,0x3C,0x32,0x45,0x5b,0};
 string key="flag";
 for(int i=0;i<4;i++)
  key[i]^=s[i];
 for(int i=0;i<strlen(s);i++){
  printf("%c",s[i]^key[i%4]);
 }
}
```

```
20,0x56,0x5D,0x18,0x22,0x45,0x17,0x2F,0x24,0x6E,0x62,0x3C,0x27,0x54,0x48,0x6C,0x24,0x6E,0x72,0x3C,0x32,0x45,0x5b,0};
string key=“flag”;
for(int i=0;i<4;i++)
key[i]^=s[i];
for(int i=0;i<strlen(s);i++){
printf("%c",s[i]^key[i%4]);
}
}
```

如果作者有哪里说的不好，还请多多海涵。