REVERSE-PRACTICE-BUUCTF-9

原创

P1umH0 于 2021-02-26 23:06:14 发布 39 收藏

分类专栏： Reverse-BUUCTF

本文链接：https://blog.csdn.net/weixin_45582916/article/details/114155831

Reverse-BUUCTF 专栏收录该内容

32 篇文章 3 订阅

订阅专栏

REVERSE-PRACTICE-BUUCTF-9

[ACTF新生赛2020]usualCrypt

exe程序，运行后提示输入flag，无壳，ida分析

main函数逻辑清晰，获取输入，sub_401080函数对输入进行变表base64变换，结果存储到v5，然后check，验证输入

IDA View-A | Pseudocode-A | Stack of _main | Hex View-1 | St

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    int v3; // esi
4    int result; // eax
5    int v5; // [esp+8h] [ebp-74h]
6    int v6; // [esp+Ch] [ebp-70h]
7    int v7; // [esp+10h] [ebp-6Ch]
8    __int16 v8; // [esp+14h] [ebp-68h]
9    char v9; // [esp+16h] [ebp-66h]
10   char input; // [esp+18h] [ebp-64h]
11
12   printf((int)aGiveMeYourFlag);                 // 来来来，Give me your flag:
13   scanf(aS, &input);
14   v5 = 0;
15   v6 = 0;
16   v7 = 0;
17   v8 = 0;
18   v9 = 0;
19   sub_401080((int)&input, strlen(&input), (int)&v5);// 变表base64
```

```
  v3 = 0;
  while ( *((_BYTE *)&v5 + v3) == byte_40E0E4[v3] )// check
  {
    if ( ++v3 > strlen((const char *)&v5) )
      goto LABEL_6;
  }
  printf((int)aError);
LABEL_6:
  if ( v3 - 1 == strlen(byte_40E0E4) )
    result = printf((int)aAreYouHappyYes);
  else
    result = printf((int)aAreYouHappyNo);
  return result;
}
```


sub_401080函数是很明显的base64，在函数开始部分，sub_401000函数对表进行了变换，在函数结束返回部分，sub_401030函数对变表后的base64字符串进行了英文字母的大小写转换

```
int __cdecl sub_401080(int a1, int a2, int a3)
{
  int v3; // edi
  int v4; // esi
  int v5; // edx
  int v6; // eax
  int v7; // ecx
  int v8; // esi
  int v9; // esi
  int v10; // esi
  int v11; // esi
  _BYTE *v12; // ecx
  int v13; // esi
  int v15; // [esp+18h] [ebp+8h]

  v3 = 0;
  v4 = 0;
  sub_401000();
  v5 = a2 % 3;
  v6 = a1;
  v7 = a2 - a2 % 3;
  v15 = a2 % 3;
  if ( v7 > 0 )
  {
    do
    {
      LOBYTE(v5) = *(_BYTE *)(a1 + v3);
      v3 += 3;
      v8 = v4 + 1;
      *(_BYTE *)(v8++ + a3 - 1) = base64_table[(v5 >> 2) & 0x3F];
      *(_BYTE *)(v8++ + a3 - 1) = base64_table[16 * (*(_BYTE *)(a1 + v3 - 3) & 3)
                                             + (((signed int)*(unsigned __int8 *)(a1 + v3 - 2) >> 4) & 0xF)];
      *(_BYTE *)(v8 + a3 - 1) = base64_table[4 * (*(_BYTE *)(a1 + v3 - 2) & 0xF)
                                           + (((signed int)*(unsigned __int8 *)(a1 + v3 - 1) >> 6) & 3)];
      v5 = *(_BYTE *)(a1 + v3 - 1) & 0x3F;
      v4 = v8 + 1;
```


sub_401000函数，将byte_40E0AA[6]到byte_40E0AA[14]和base64_table[6]到base64_table[14]两段交换

```
signed int sub_401000()
{
  signed int result; // eax
  char v1; // cl

  result = 6;
  do
  {
```

```
    v1 = byte_40E0AA[result];
    byte_40E0AA[result] = base64_table[result];
    base64_table[result++] = v1;
  }
  while ( result < 15 );
  return result;
}
```


sub_401030函数，对变表后的base64字符串英文字母的大小写转换，其他字符不变

```
int __cdecl sub_401030(const char *a1)
{
  __int64 v1; // rax
  char v2; // al

  v1 = 0i64;
  if ( strlen(a1) != 0 )
  {
    do
    {
      v2 = a1[HIDWORD(v1)];
      if ( v2 < 97 || v2 > 122 )
      {
        if ( v2 < 65 || v2 > 90 )
          goto LABEL_9;
        LOBYTE(v1) = v2 + 32;
      }
      else
      {
        LOBYTE(v1) = v2 - 32;
      }
      a1[HIDWORD(v1)] = v1;
LABEL_9:
      LODWORD(v1) = 0;
      ++HIDWORD(v1);
    }
    while ( HIDWORD(v1) < strlen(a1) );
  }
  return v1;
}
```


写脚本即可得到flag

```
#coding:utf-8
#base="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"  原表
base=[0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4A,
  0x4B, 0x4C, 0x4D, 0x4E, 0x4F, 0x50, 0x51, 0x52, 0x53, 0x54,
  0x55, 0x56, 0x57, 0x58, 0x59, 0x5A, 0x61, 0x62, 0x63, 0x64,
  0x65, 0x66, 0x67, 0x68, 0x69, 0x6A, 0x6B, 0x6C, 0x6D, 0x6E,
  0x6F, 0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78,
  0x79, 0x7A, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
  0x38, 0x39, 0x2B, 0x2F]  #原表的ascii码表示，方便进行原表变换
#对原表进行变换
for i in range(6,15):
    base[i],base[10+i]=base[10+i],base[i]
#base_changed是变表，需要转成字符串的形式
base_changed=''.join(chr(i) for i in base)
print("Current Base:\n%s " %base_changed) #打印base_changed变表
def base64_decode(inputs): #inputs是base64字符串
    # 将字符串转化为2进制
    bin_str = []
    for i in inputs:
        if i != '=':
            x = str(bin(base_changed.index(i))).replace('0b', '')
            bin_str.append('{:0>6}'.format(x))
    # 输出的字符串
    outputs = ""
    nums = inputs.count('=')
    while bin_str:
        temp_list = bin_str[:4]
        temp_str = "".join(temp_list)
        # 补足8位字节
        if (len(temp_str) % 8 != 0):
            temp_str = temp_str[0:-1 * nums * 2]
        # 将四个6字节的二进制转换为三个字符
        for i in range(0, int(len(temp_str) / 8)):
            outputs += chr(int(temp_str[i * 8:(i + 1) * 8], 2))
        bin_str = bin_str[4:]
    print("Decoded String:\n%s " % outputs)
#enc是经变表base64以及大小写转换后的字符串
enc="zMXHz3TIgnxLxJhFAdtZn2fFk3lYCrtPC2l9"
#c将enc大小写转换回去 再解变表base64
c=""
for i in enc:
    if i.isupper():
        c+=i.lower()
    elif i.islower():
        c+=i.upper()
    else:
        c+=i
base64_decode(c)
```

运行结果

```
D:\python27-x64\python2.exe D:/Python/pycharm/pycfile/test1.py
Current Base:
ABCDEFQRSTUVWXYPGHIJKLMN0Zabcdefghijklmnopqrstuvwxyz0123456789+/
Decoded String:
flag{bAse64_h2s_a_Surprise}
```

[MRCTF2020]Transform

exe程序，运行后提示输入code，输入错误打印Wrong，无壳，ida分析

main函数逻辑清晰，获取输入，检验输入的长度，对输入的内容变换位置后存储到byte_414040，byte_414040再异或一下，最后check

```
10  sub_402230();
11  printf(argc, (__int64)argv, v3, (__int64)"Give me your code:\n");
12  sub_40E5F0(argc, (__int64)argv, (__int64)input, (__int64)"%s");
13  if ( strlen(*(const char **)&argc) != 33 )    // 输入的长度为33
14  {
15    printf(argc, (__int64)argv, v4, (__int64)"Wrong!\n");
16    system(*(const char **)&argc);
17    exit(argc);
18  }
19  for ( i = 0; i <= 32; ++i )
20  {
21    byte_414040[i] = input[dword_40F040[i]];    // dword_40F040数组元素作为input的下标，结果存储到byte_414040，实际上是对input的内容变换位置
22    v4 = i;
23    byte_414040[i] ^= LOBYTE(dword_40F040[i]);  // byte_414040异或dword_40F040
24  }
25  for ( j = 0; j <= 32; ++j )
26  {
27    v4 = j;
28    if ( res[j] != byte_414040[j] )           // check
29    {
30      printf(argc, (__int64)argv, j, (__int64)"Wrong!\n");
31      system(*(const char **)&argc);
32      exit(argc);
33    }
34  }
35  printf(argc, (__int64)argv, v4, (__int64)"Right!Good Job!\n");
36  printf(argc, (__int64)argv, (__int64)input, (__int64)"Here is your flag: %s\n");
37  system(*(const char **)&argc);
38  return 0;
39}
```


写脚本即可得到flag

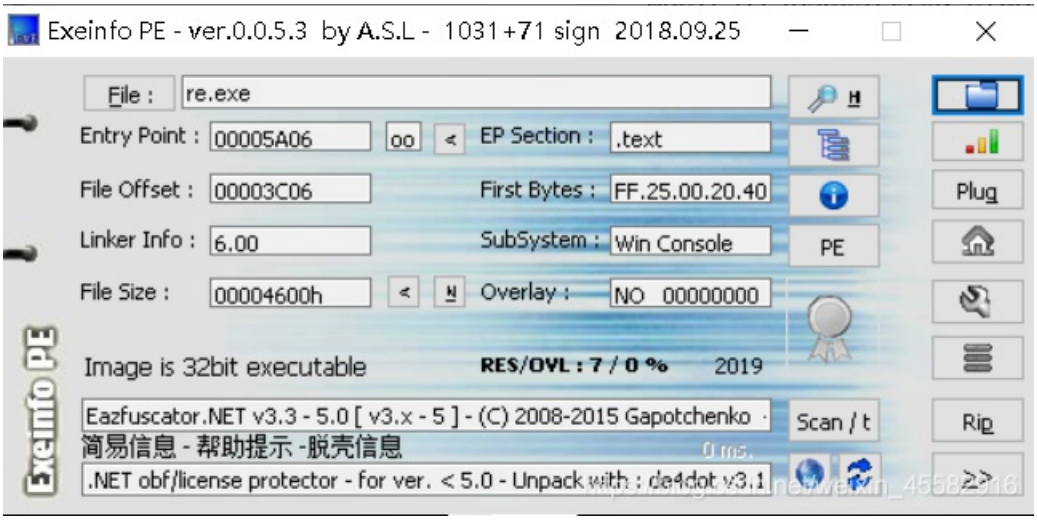
```
res=[0x67, 0x79, 0x7B, 0x7F, 0x75, 0x2B, 0x3C, 0x52, 0x53, 0x79,
  0x57, 0x5E, 0x5D, 0x42, 0x7B, 0x2D, 0x2A, 0x66, 0x42, 0x7E,
  0x4C, 0x57, 0x79, 0x41, 0x6B, 0x7E, 0x65, 0x3C, 0x5C, 0x45,
  0x6F, 0x62, 0x4D]
arr_40F040=[9,10,15,23,7,24,12,6,1,16,3,17,32,29,11,30,27,22,4,
  13,19,20,21,2,25,5,31,8,18,26, 28, 14,0]
flag=[0]*len(res)
for i in range(len(res)):
    res[i]^=arr_40F040[i]
    flag[arr_40F040[i]]=res[i]
print(''.join(chr(i) for i in flag))
```

```
test1
D:\python27-x64\python2.exe D:/Python/pycharm/pycfile/test1.py
MRCTF{Tr4nsp0slti0N_Clph3r_1s_3z}
```


[V&N2020 公开赛]CSRe

exe程序，运行出错

查壳，发现是.NET，而且加了de4dot壳



脱壳后用dnSpy打开，找到主逻辑函数

method_0函数是将两个传入的参数顺序异或的结果返回

smethod_0函数是对传入的参数进行sha1散列的结果返回

main函数，先获取输入的str，头部加个“3”，尾部加个“9”，拼接后的字符串进行sha1散列，与已知的值比较，验证输入的str是否正确

再获取输入的text，头部加“re”，拼接后的字符串进行sha1散列，散列的结果与已知的值进行异或，结果为全0，说明两个传入的参数完全相同，验证输入的text是否正确

flag即为flag{str+text}

```
using System;
using System.Security.Cryptography;
using System.Text;

// Token: 0x02000006 RID: 6
internal sealed class Class3
{
 // Token: 0x0600000D RID: 13 RVA: 0x000022C8 File Offset: 0x000004C8
 public string method_0(string string_0, string string_1)
 {
  string text = string.Empty;
  char[] array = string_0.ToCharArray();
  char[] array2 = string_1.ToCharArray();
  int num = (array.Length < array2.Length) ? array.Length : array2.Length;
  for (int i = 0; i < num; i++)
  {
   text += (int)(array[i] ^ array2[i]);
  }
  return text;
 }

 // Token: 0x0600000E RID: 14 RVA: 0x0000231C File Offset: 0x0000051C
```

```
public static string smethod_0(string string_0)
{
 byte[] bytes = Encoding.UTF8.GetBytes(string_0);
 byte[] array = SHA1.Create().ComputeHash(bytes);
 StringBuilder stringBuilder = new StringBuilder();
 foreach (byte b in array)
 {
  stringBuilder.Append(b.ToString("X2"));
 }
 return stringBuilder.ToString();
}

// Token: 0x0600000F RID: 15 RVA: 0x00002374 File Offset: 0x00000574
private static void Main(string[] args)
{
 if (!Class1.smethod_1())
 {
  return;
 }
 bool flag = true;
 Class3 @class = new Class3();
 string str = Console.ReadLine();
 if (Class3.smethod_0("3" + str + "9") != "B498BFA2498E21325D1178417BEA459EB2CD28F8")
 {
  flag = false;
 }
 string text = Console.ReadLine();
 string string_ = Class3.smethod_0("re" + text);
 string text2 = @class.method_0(string_, "63143B6F8007B98C53CA2149822777B3566F9241");
 for (int i = 0; i < text2.Length; i++)
 {
  if (text2[i] != '0')
  {
   flag = false;
  }
 }
 if (flag)
 {
  Console.WriteLine("flag{" + str + text + "}");
 }
}
}
```

用在线网站解第一段sha1，可知str为“1415”

用在线网站解第二段sha1，可知text为“turn”

[WUSTCTF2020]level1

elf文件，无壳，ida分析

main函数逻辑清晰，从文件读取flag，下标从1开始，下标为奇数，flag的内容移位，下标为偶数，flag的内容乘以下标

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  FILE *stream; // ST08_8
  signed int i; // [rsp+4h] [rbp-2Ch]
  char flag[24]; // [rsp+10h] [rbp-20h]
  unsigned __int64 v7; // [rsp+28h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  stream = fopen("flag", "r");
  fread(flag, 1uLL, 0x14uLL, stream);
  fclose(stream);
  for ( i = 1; i <= 19; ++i )
  {
    if ( i & 1 )                                    // i是奇数
      printf("%ld\n", (unsigned int)(flag[i] << i));
    else                                            // i是偶数
      printf("%ld\n", (unsigned int)(i * flag[i]));
  }
  return 0;
}
```


写脚本即可得到flag

```
arr=[0,198,232,816,200,1536,300,6144,984,51200,570,92160,
    1200,565248,756,1474560,800,6291456,1782,65536000]
flag="w"
for i in range(1,len(arr)):
    if i%2==1:
        flag+=chr(arr[i]>>i)
    else:
        flag+=chr(arr[i]//i)
print(flag)
```

```
test1
  D:\python27-x64\python2.exe D:/Python/pycharm/pycfile/test1.py
  wctf2020{d9-dE6-20c}
```