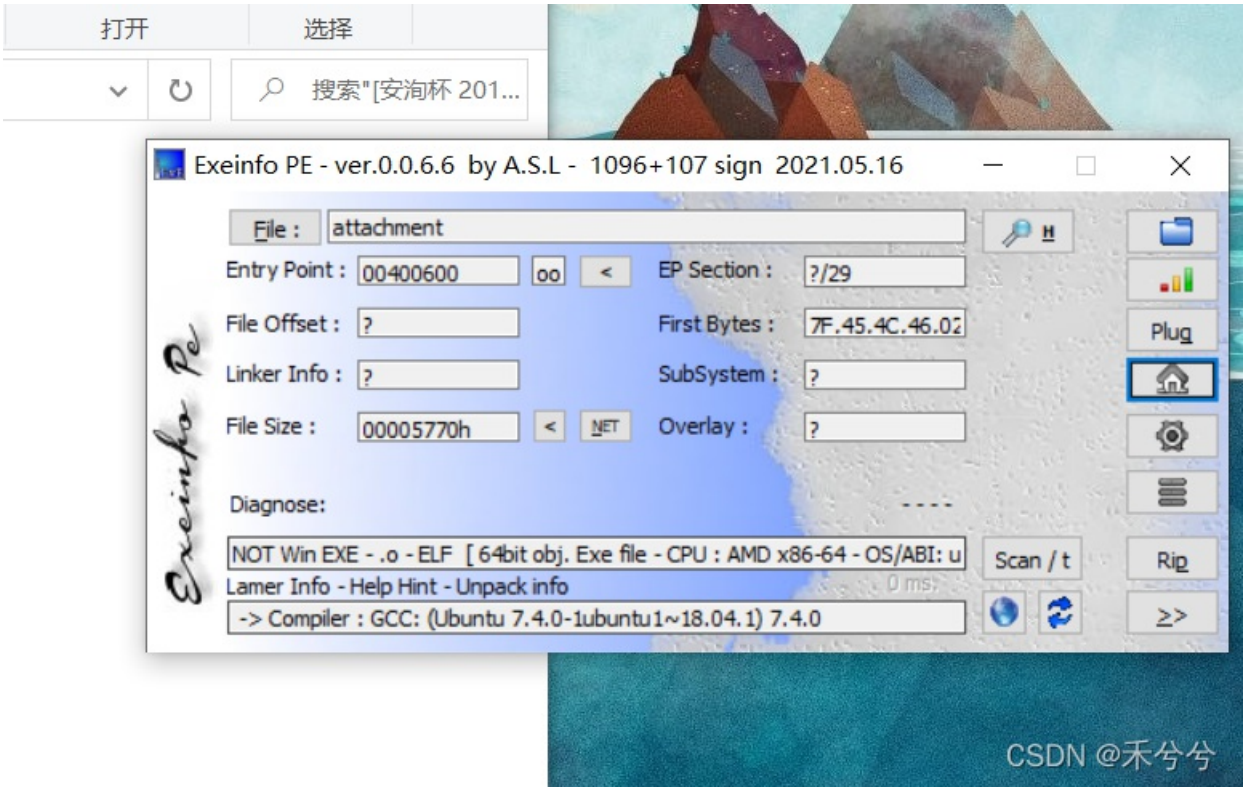
[安洵杯 2019]game writeup

原创

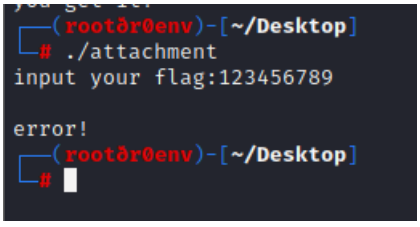
禾兮兮 于 2021-12-31 13:56:42 发布 195 收藏
文章标签： 经验分享 其他

每天一道re

Exeinfo PE查壳，可以用Linux打开



用kali打开，得到关键语句



用ida打开，找到main函数F5反编译

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  signed int v3; // eax
  unsigned int v4; // ST38_4
  __int64 v5; // rsi
  signed int v7; // [rsp+2Ch] [rbp-54h]
  char v8; // [rsp+40h] [rbp-40h]
  int v9; // [rsp+78h] [rbp-8h]
  int v10; // [rsp+7Ch] [rbp-4h]

  v9 = 0;
  printf("input your flag:", argv, envp);
  gets(&v8);
  v10 = general_inspection((int (*)[9])sudoku);
  v7 = -1804515313;
  while ( 1 )
  {
    while ( 1 )
    {
      while ( v7 == -2071121728 )
      {
        v4 = blank_num((int (*)[9])sudoku);
        v5 = mem_alloc(v4);
        trace(sudoku, v5, v4);
        check((int (*)[9])sudoku);
        check1(&v8);
        check3(&v8);
        v9 = 0;
        v7 = -303742386;
      }
      if ( v7 != -1804515313 )
        break;
      v3 = -2071121728;
      if ( v10 )
        v3 = 664169471;
      v7 = v3;
    }
    if ( v7 == -303742386 )
      break;
    if ( v7 == 664169471 )
    {
      printf("error");
      check((int (*)[9])sudoku);
      v9 = 0;
      v7 = -303742386;
    }
  }
  return v9;
}
```

输入的flag传入到v8，直接跟踪v8，发现传入了check1和check3，查看check1，用了ollvm，OLLVM（Obfuscator-LLVM）是瑞士西北应用科技大学安全实验室于2010年6月份发起的一个项目，这个项目的目标是提供一个LLVM编译套件的开源分支，能够通过代码混淆和防篡改，增加对逆向工程的难度，提供更高的软件安全性。

来自(47条消息) OLLVM分析_chrisnotfound-CSDN博客_ollvm

其实慢慢分析不断跟踪v10的值，就很快分析出了他其实就是三层加密操作

```
result = (unsigned int)(v10 + 67245798);
if ( v10 == -67245798 )
  break;
switch ( v10 )
{                                       // 第一层加密：将flag分为两部分，前后部分置换
  case 75381312:
    v4 = a1[v12];
    a1[v12] = a1[v11];
    a1[v11] = v4;
    v10 = -291294424;
    break;
  case 1519002972:
    v2 = strlen(a1);
    v3 = 1555725255;
    if ( v11 < v2 >> 1 )                // strlen的一半
      v3 = 75381312;
    v10 = v3;
    break;
  case 1555725255:
    v12 = 0;
    v10 = -1393133668;
    break;
  case 1974939745:
    a1[v12] = (a1[v12] & '' | ~a1[v12] & '\f') - 20;// 第三层加密，对每个值进行操作 a1[v12] = (a1[v12] & 0xF3 | ~a1[v12] & 0xC) - 20;
    v10 = -831482631;
    break;
}
}
return result;
```



```
            }
            if ( v10 != -1393133668 )
              break;
            v5 = strlen(a1);
            v6 = -1988665894;
            if ( v12 < v5 )
              v6 = -1018472136;
            v10 = v6;
          }
          if ( v10 != -1018472136 )
            break;
          v7 = a1[v12];
          a1[v12] = a1[v12 + 1];            // 第二层加密：前一个和后一个置换
          a1[v12 + 1] = v7;
          v10 = -146751883;
        }
        if ( v10 != -831482631 )
          break;
        ++v12;
        v10 = -2084833488;
      }
      if ( v10 != -291294424 )
        break;
      ++v11;
      ++v12;
      v10 = 1519002972;
```


接下来就是check3，check3又套了一层check2加密操作

```
__int64 __fastcall check3(char *a1)
{
  __int64 result; // rax
  signed int v2; // eax
  signed int v3; // [rsp+28h] [rbp-18h]
  int v4; // [rsp+3Ch] [rbp-4h]

  v4 = check2(a1);
  v3 = 16123822;
  while ( 1 )
  {
    while ( v3 == 16123822 )
    {
      v2 = 1478060410;
      if ( !v4 )
        v2 = 1274132590;
      v3 = v2;
    }
    result = (unsigned int)(v3 - 824643665);
    if ( v3 == 824643665 )
      break;
    if ( v3 == 1274132590 )
    {
      v3 = 824643665;
      printf("error!\n");
    }
    else if ( v3 == 1478060410 )
    {
      v3 = 824643665;
      printf("you get it!\n");
    }
  }
  return result;
}
```

根据动态调试可以知道flag不正确会返回1，正确即为零，然后check3就会输出“you get it！”

看看check3，也是和check1同样的分析方法

```
  case 430996436:
    ++v15;
    v11 = -2671583;
    break;
  case 441246003:
    v16[v15] = s[v15] - 232084296 + 232084248;// 第一层加密：将s操作给了v16
    v11 = 430996436;
    break;
  case 564268595:
    v7 = 1954800504;
    if ( v14 < 9 )
      v7 = -1658909923;
    v11 = v7;
    break;
  case 705300330:
    v5 = 1611237474;
    if ( !D0g3[9 * v15 + v14] )
      v5 = -1244045086;
    v11 = v5;
```

```
        }
        if ( v11 != -1658909923 )
          break;
        v8 = -1129833658;
        if ( D0g3[9 * v15 + v14] != sudoku[9 * v15 + v14] )// 第三层：与sodu进行比较，一致则输出v12=0
          v8 = -528396247;
        v11 = v8;
      }
      if ( v11 != -1613667829 )
        break;
      v11 = -2119125118;
    }
    if ( v11 != -1369143226 )
      break;
    v14 = 0;
    v11 = -740861019;
  }
  if ( v11 != -1244045086 )
    break;
  D0g3[9 * v15 + v14] = v16[v13++];// 第二层：D0g3[9 * v15 + v14] ==0时，将v16给了二维数组dog3
  v11 = 1611237474;
}
if ( v11 != -1129833658 )
  break;
v11 = -90011013;
}
if ( v11 != -740861019 )
  break;
```


第二层：D0g3[9 * v15 + v14] ==0时，将v16给了二维数组dog3，有点绕，我用了动态调试的方法，一开始查看dog3的值：

```
data:00000000006042FE db    0
data:00000000006042FF db    0
data:0000000000604300 public D0g3
data:0000000000604300 ; _DWORD D0g3[81]
data:0000000000604300 D0g3 dd 1, 0, 5, 3, 2, 7, 0, 0, 8, 8, 0, 9, 0, 5, 0, 0, 2, 0, 0, 7, 0, 0, 1
data:0000000000604300                                         ; DATA XREF: check2(char *)+462↑o
data:0000000000604300                                         ; check2(char *):loc_402F53↑o ...
data:0000000000604300 dd 0, 5, 0, 3, 4, 9, 0, 1, 0, 0, 3, 0, 0, 0, 1, 0, 0, 7, 0, 9, 0, 6, 7
data:0000000000604300 dd 0, 3, 2, 9, 0, 4, 8, 0, 0, 6, 0, 5, 4, 0, 8, 0, 9, 0, 0, 4, 0, 0, 1
data:0000000000604300 dd 0, 3, 0, 0, 2, 1, 0, 3, 0, 7, 0, 4
data:0000000000604300 _data ends
data:0000000000604300
```


不对F8，得到

```
0000000604300 ; _DWORD D0g3[81]
0000000604300 D0g3 dd 1, 0FFFFFFF6h, 5, 3, 2, 7, 0FFFFFFF5h, 0FFFFFFF0h, 8, 8, 0FFFFFFF7h
0000000604300                                         ; DATA XREF: check2(char *)+462↑o
0000000604300                                         ; check2(char *):loc_402F53↑o ...
0000000604300 dd 9, 0FFFFFFFAh, 5, 0, 0, 2, 0, 0, 7, 0, 0, 1, 0, 5, 0, 3, 4, 9, 0, 1
0000000604300 dd 0, 0, 3, 0, 0, 0, 1, 0, 0, 7, 0, 9, 0, 6, 7, 0, 3, 2, 9, 0, 4, 8, 0
0000000604300 dd 0, 6, 0, 5, 4, 0, 8, 0, 9, 0, 0, 4, 0, 0, 1, 0, 3, 0, 0, 2, 1, 0, 3
0000000604300 dd 0, 7, 0, 4
0000000604300 _data ends
0000000604300
```

```
stack]:00007FFC7FD574E0 db 0F6h
stack]:00007FFC7FD574E1 db 0FFh
stack]:00007FFC7FD574E2 db 0FFh
stack]:00007FFC7FD574E3 db 0FFh
stack]:00007FFC7FD574E4 db 0F5h
stack]:00007FFC7FD574E5 db 0FFh
stack]:00007FFC7FD574E6 db 0FFh
stack]:00007FFC7FD574E7 db 0FFh
stack]:00007FFC7FD574E8 db 0F0h
stack]:00007FFC7FD574E9 db 0FFh
stack]:00007FFC7FD574EA db 0FFh
stack]:00007FFC7FD574EB db 0FFh
stack]:00007FFC7FD574EC db 0F7h
stack]:00007FFC7FD574ED db 0FFh
stack]:00007FFC7FD574EE db 0FFh
stack]:00007FFC7FD574EF db 0FFh
stack]:00007FFC7FD574F0 db 0FAh
stack]:00007FFC7FD574F1 db 0FFh
stack]:00007FFC7FD574F2 db 0FFh
stack]:00007FFC7FD574F3 db 0FFh
stack]:00007FFC7FD574F4 db 0F9h
stack]:00007FFC7FD574F5 db 0FFh
```


再去与v16的值进行对比，发现是v16的值给了dog3中零的位置

然后就密码核对用到的sudoku，查看main的函数发现有对sudoku进行操作，直接动态调试得到结果

最后的脚本为：

```
#include<stdio.h>
int main()
{
 int a[81]={1,4,5,3,2,7,6,9,8,8,3,9,6,5,4,1,2,7,6,7,2,8,1,9,5,4,3,4,9,6,1,8,5,3,7,2,2,1,8,4,7,3,9,5,6,7,5,3
 int P[81]={1,0,5,3,2,7,0,0,8,8,0,9,0,5,0,0,2,0,0,7,0,0,1,0,5,0,3,4,9,0,1,0,0,3,0,0,0,1,0,0,7,0,9,0,6,7,0,3
 int R[40];//输入的flag
 int o;
 int L=0;
 int temp;

 for(int i=0;i<81;i++)
    {
     if(P[i]==0)
     {R[L]=a[i]+232084296-232084248;L++;} //得到flag的密码表
    }
  for (int i = 0; i <40; i++) { temp = R[i] + 20; temp = temp & 0xf3 | ~temp & 0xc; R[i] = temp; }
  //前后两个置换
   for(int i=0;i<40;i=i+2)
   {o = R[i]; R[i] = R[i+1]; R[i+1] = o;}
  //前后两个部分置换
  int g=40>>1;
  for(int i=0;i<40>>1;i++,g++)
  {o = R[i]; R[i] = R[g]; R[g] = o;}
  for(int i=0;i<40;i++)
  {printf("%c",R[i]);}

}
```

```
└─# ./attachment
input your flag:KDEEIFGKIJ@AFGEJAEF@FDKADFGIJFA@FDE@JG@J
you get it!
```


反思，在check2第二层加密中，没有想到用动态分析，导致静态分析错误，又浪费了时间。