T0fV404 #000002

RE

signin

虽然就做了一个,还是写一下wp吧.

主函数:

```
int __fastcall main_0(int argc, const char **argv, const char **envp)
{
  char *v3; // rdi
  __int64 i; // rcx
  __int64 v5; // rdx
  __int64 v6; // rcx
  __int64 v7; // r8
  char v9; // [rsp+20h] [rbp+0h] BYREF
  char Str1[6]; // [rsp+30h] [rbp+10h] BYREF
  char Source[82]; // [rsp+36h] [rbp+16h] BYREF
  char Destination[264]; // [rsp+88h] [rbp+68h] BYREF

  v3 = &v9;
  for ( i = 42i64; i; --i )
  {
    *(_DWORD *)v3 = -858993460;
    v3 += 4;
  }
  j___CheckForDebuggerJustMyCode(&unk_7FF67C4370A3, argv, envp);
  memset(Str1, 0i64, 64i64);
  printf("password:");
  scanf("%44s", Str1);
  if ( (unsigned int)undebug(v6, v5, v7) && (unsigned int)crc32() )
  {
    if ( !j_strncmp(Str1, "hgame{", 6ui64)
      && (j_strncpy(Destination, Source, 0x24ui64), (unsigned int)xtea(Destination)) )
    {
      j_puts("right");
    }
    else
    {
      j_puts("wrong");
    }
  }
  else
  {
    j_puts("error\n");
  }
  return 0;
}
```

首先是反调试:

```
__int64 __fastcall sub_7FF67C37F730(__int64 a1, __int64 a2, __int64 a3)
{
  __int64 result; // rax
  HANDLE CurrentThread; // rax
  LPCONTEXT lpContext; // [rsp+28h] [rbp+8h]

  j___CheckForDebuggerJustMyCode(&unk_7FF67C4370A3, a2, a3);
  lpContext = (LPCONTEXT)VirtualAlloc(0i64, 0x4D0ui64, 0x1000u, 4u);
  if ( lpContext )
  {
    sub_7FF67C3734FE(lpContext, 1232i64);
    lpContext->ContextFlags = 1048592;
    CurrentThread = GetCurrentThread();
    if ( GetThreadContext(CurrentThread, lpContext) )
    {
      qword_7FF67C42B880[0] = lpContext->Dr0;
      qword_7FF67C42B880[1] = lpContext->Dr1;
      qword_7FF67C42B880[2] = lpContext->Dr2;
      qword_7FF67C42B880[3] = lpContext->Dr3;
      if ( qword_7FF67C42B880[0]
        || qword_7FF67C42B880[1]
        || qword_7FF67C42B880[2]
        || (result = 24i64, qword_7FF67C42B880[3]) )
      {
        j_puts("Debug error.");
        j_exit(0);
      }
    }
    else
    {
      return 0i64;
    }
  }
  else
  {
    j_puts("VirtualAlloc failed.");
    return 0i64;
  }
  return result;
}
```

搜了一下,dr寄存器是控制硬件断点的地址和状态的,所以打硬件断点会被gank.这里是通过上下文直接获取dr寄存器内容

接下来是crc32校验:

```
__int64 __fastcall sub_7FF67C378670(__int64 a1, __int64 a2, __int64 a3)
{
  char *v4; // [rsp+48h] [rbp+28h]
  int i; // [rsp+64h] [rbp+44h]

  j___CheckForDebuggerJustMyCode(&unk_7FF67C4370A3, a2, a3);
  v4 = (char *)j_j_j__malloc_base(0x10000ui64);
  memset(v4, 0i64, 0x10000i64);
  memcpy(v4, main, 0x10000i64);
  sub_7FF67C3711D6();
  for ( i = 0; i < 4; ++i )
    dword_7FF67C42B2A0[i] = sub_7FF67C371AB4(&v4[0x4000 * i], 0x4000i64);
  return 1i64;
}

__int64 __fastcall sub_7FF67C378760(__int64 a1, unsigned __int64 a2, __int64 a3)
{
  unsigned int v4; // [rsp+24h] [rbp+4h]
  unsigned __int64 i; // [rsp+48h] [rbp+28h]

  j___CheckForDebuggerJustMyCode(&unk_7FF67C4370A3, a2, a3);
  v4 = -1;
  for ( i = 0i64; i < a2; ++i )
    v4 = dword_7FF67C42D1A0[(unsigned __int8)(*(_BYTE *)(i + a1) ^ v4)] ^ (v4 >> 8);
  return ~v4;
}
```

是没魔改过的crc校验(findcrypto可查),逻辑是检查代码有无改变.

加密逻辑

```
__int64 __fastcall sub_7FF67C378820(__int64 a1, __int64 a2, __int64 a3)
{
  int i; // [rsp+24h] [rbp+4h]

  j___CheckForDebuggerJustMyCode(&unk_7FF67C4370A3, a2, a3);
  enc(a1, dword_7FF67C42B2A0, qword_7FF67C42B880);
  for ( i = 0; i < 36; ++i )
  {
    if ( *(unsigned __int8 *)(a1 + i) != (unsigned __int8)a0[i] )
      return 0i64;
  }
  return 1i64;
}
```

其中a1是`hgame{`右边的输入,`dword_7FF67C42B2A0`是上下文获得的dr寄存器内容,肯定要为0,`qword_7FF67C42B880`为crc32校验结果,肯定是能动源码的.

而软件断点会加入`int3`中断导致代码变动,故而我们只能打硬件断点并且修改`dword_7FF67C42B2A0`的内容为0.

加密是tea类函数,估计是xtea,解密脚本如下:

```
#include <stdio.h>

int main()
{
    unsigned char data[] =
        {
            0x23, 0xEA, 0x50, 0x30, 0x00, 0x4C, 0x51, 0x47, 0xEE, 0x9C,
            0x76, 0x2B, 0xD5, 0xE6, 0x94, 0x17, 0xED, 0x2B, 0xE4, 0xB3,
            0xCB, 0x36, 0xD5, 0x61, 0xC0, 0xC2, 0xA0, 0x7C, 0xFE, 0x67,
            0xD7, 0x5E, 0xAF, 0xE0, 0x79, 0xC5, 0x00};
    unsigned int *enc = (unsigned int *)data;
    unsigned char crc_data[] =
```

```c
        {0xB5, 0x5F, 0xA2, 0x97, 0xBA, 0x6D, 0x75, 0xE1, 0x4A, 0x46,
         0x43, 0xA1, 0x4F, 0x28, 0x8F, 0x5A};

    unsigned int *CRC = (unsigned int *)crc_data;
    unsigned int key[4] = {0};

    unsigned int j = 0;
    unsigned int last;
    unsigned int cur_enc;
    unsigned int nex;
    for (int i = 1; i <= 11; i++)
    {
        j += key[i % 4];
    }

    for (int i = 1; i <= 11; i++)
    {
        unsigned int jj = (j >> 2) & 3;
        for (int k = 8; k >= 0; k--)
        {
            if (k == 8)
            {
                last = enc[k - 1];
                cur_enc = enc[k];
                enc[k] = cur_enc - (((last ^ CRC[jj ^ k & 3]) + (*enc ^ j)) ^ (((16 * last) ^ (*enc >> 3)) + ((4 * *enc) ^ (last >> 5))));
            }
            else
            {
                if (k == 0)
                    last = enc[8];
                else
                    last = enc[k - 1];
                cur_enc = enc[k];
                nex = enc[k + 1];
                enc[k] = cur_enc - (((last ^ CRC[jj ^ k & 3]) + (nex ^ j)) ^ (((16 * last) ^ (nex >> 3)) + ((4 * nex) ^ (last >> 5))));
            }
        }
        j -= key[i % 4];
    }
    printf("hgame{%s", enc);
```

```
}

```

flag为`hgame{3fe4722c-1dbf-43b7-8659-c1c4a0e42e4d}`