bytectf 2019 re 驱动逆向 DancingKeys WP

神雕大侠qsq / 2019-09-18 09:19:00 / 浏览数 3919 安全技术 CTF 顶(1) 踩(0)

---

bytectf 2019 re 驱动逆向 DancingKeys WP

## 比赛地址

https://adworld.xctf.org.cn/match/contest_challenge?event=101&hash=b1c22799-e6cf-4892-937d-c189605f5b5f.event

## 简介

本题是一个windows键盘过滤驱动程序的逆向，可以参考https://blog.csdn.net/m0_37552052/article/details/83037567

## 程序流程分析

在driver_entry驱动入口函数中

```
NTSTATUS __stdcall DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
{
 struct _DRIVER_OBJECT *v2; // rdi

 v2 = DriverObject;
 _security_init_cookie();
 return sub_140002C90(v2);                     // ■■
}
```

跟进：

```
__int64 __fastcall sub_140002C90(PDRIVER_OBJECT a1)
{
 __int64 v1; // rdx
 __int64 v2; // r8
 __int64 v3; // r9
 unsigned int i; // [rsp+20h] [rbp-18h]
 PDRIVER_OBJECT v6; // [rsp+40h] [rbp+8h]

 v6 = a1;
 sub_1400032C0();                              // windows■■■■■■
 sub_140003170();                              // CPU■■■■■■■
 sub_140002830();                              // ■■■■■■■■■■■■■■■■■■■■■■■■
 sub_1400033E0(v6);                            // ■■■■■■■■■■■■■\\Driver\\Kbdclass
 sub_1400028A0(v6, v1, v2, v3);                // ■■■■■■■\\??\\DancingKeys■■■■■■■■■\\Device\\DancingKeys
 for ( i = 0; i < 0x1B; ++i )
   v6->MajorFunction[i] = (PDRIVER_DISPATCH)sub_1400029D0; //■■MajorFunction■■■■
 v6->MajorFunction[IRP_MJ_READ] = (PDRIVER_DISPATCH)sub_140002C40;// ■■■■■■■■■
 v6->MajorFunction[IRP_MJ_PNP] = (PDRIVER_DISPATCH)sub_140002BA0;// pnp
 v6->MajorFunction[IRP_MJ_POWER] = (PDRIVER_DISPATCH)sub_140002BF0;// ■■
 v6->MajorFunction[IRP_MJ_DEVICE_CONTROL] = (PDRIVER_DISPATCH)sub_140002A20;// IO■■■■■■■■■
 v6->DriverUnload = (PDRIVER_UNLOAD)sub_140002B50;// ■■■■
 return 0i64;
}
```

比较关键的地方在于键盘输入处理例程和IO控制请求处理例程：

### 键盘输入处理例程

由于前面绑定了键盘设备，所以所有的键盘IRP请求，会走本驱动过一遍。本驱动的MajorFunction[IRP_MJ_READ]拦截键盘输入操作
跟进该处理例程至关键代码：

```
__int64 __fastcall sub_1400037A0(_DEVICE_OBJECT *a1, _IRP *a2)
{
 __int64 v2; // r9
 __int64 v4; // [rsp+20h] [rbp-38h]
 __int64 v5; // [rsp+28h] [rbp-30h]
 unsigned int i; // [rsp+30h] [rbp-28h]
```

```
 PKEYBOARD_INPUT_DATA v7; // [rsp+38h] [rbp-20h]
 ULONG_PTR v8; // [rsp+40h] [rbp-18h]
 _IRP *v9; // [rsp+68h] [rbp+10h]

 v9 = a2;
 if ( a2->IoStatus.Status >= 0 )
 {
   v7 = (PKEYBOARD_INPUT_DATA)a2->AssociatedIrp.SystemBuffer;
   v8 = a2->IoStatus.Information / 0xC;
   for ( i = 0; i < v8; ++i )
   {
     if ( !v7[i].Flags )                       // ■■■■■flag==0■■■■
     {
       input[input_count] = (last_input + 42) ^ v7[i].MakeCode;// MakeCode■■■■■■■■■■■■■■■■■■■■
       last_input = input[input_count];
       LODWORD(v4) = input[input_count];
       DbgPrintEx(77i64, 563i64, "Magic code %d: %02x\n", (unsigned int)input_count, v4, v5);
       if ( ++input_count == 16 )
       {
         DbgPrintEx(77i64, 563i64, "Magic code buffer is now full\n", v2, v4, v5);
         input_count = 0;
         last_input = 0;
       }
     }
   }
 }
 --dword_140006080;
 if ( v9->PendingReturned )
   sub_140003A20(v9);
 return (unsigned int)v9->IoStatus.Status;
}
```

题目给了一段神秘代码，刚好是16字节，猜测就是这里的数据，将数据按如上算法解密发现恰好是输入的键盘码,解密代码：

```
data = [0x25,0x40,0x5a,0x86,0xb5,0xf1,0x3e,0x58,0x80,0x9b,0xdb,0x0b,0x30,0x49,0x66,0x8c]
res = []
temp = 0
for i in data:
   res.append(((temp+42)%256)^i)
   temp = i
print res
```

解密结果为按下了：tab tab b 1 4 c k b 1 n a backspace 4 r y enter，基本确定就是这样了。

## IO控制请求处理例程

当输入完成后，应用层通过DeviceIoControl使用控制码0x222404与驱动通讯，驱动根据虚拟的操作系统版本和cpu信息数据与上面的键盘码进行一系列运算，最终向用户

```
__int64 __fastcall sub_140002A20(_DEVICE_OBJECT *a1, _IRP *a2)
{
 __int64 v2; // r9
 __int64 v4; // [rsp+20h] [rbp-28h]
 _IO_STACK_LOCATION *v5; // [rsp+28h] [rbp-20h]
 _IRP *v6; // [rsp+30h] [rbp-18h]
 _DEVICE_OBJECT *v7; // [rsp+50h] [rbp+8h]
 _IRP *Irp; // [rsp+58h] [rbp+10h]

 Irp = a2;
 v7 = a1;
 sub_1400027A0();                              // ■■■nop■■
 v5 = sub_140002D80(Irp);
 HIDWORD(v4) = v5->Parameters.Read.ByteOffset.LowPart;
 if ( HIDWORD(v4) == 0x222404 )
 {
   if ( v7 == DeviceObject )
   {
     v6 = (_IRP *)Irp->AssociatedIrp.SystemBuffer;
     LODWORD(v4) = v5->Parameters.Read.Length;
     if ( v6 && (unsigned int)v4 >= 0x64 )
     {
```

```
      sub_1400030B0();                      // ■■■■■■■■cpu■■■■■■■■md5■■■md5■■8■■■■■0x20■■■■■
      sub_140002DD0(v6);                    // ■■■■■■■■■■■■■■■■■■
      Irp->IoStatus.Information = (unsigned int)v4;
      Irp->IoStatus.Status = 0;
      IofCompleteRequest(Irp, 0);
      return 0i64;
    }
    DbgPrintEx(77i64, 563i64, "Invalid Output Buffer\n", v2, v4, v5);
  }
  else
  {
    DbgPrintEx(77i64, 563i64, "Wrong device!\n", v2, v4, v5);
  }
 }
 else
 {
   DbgPrintEx(77i64, 563i64, "Wrong device control code!\n", v2, v4, v5);
 }
 Irp->IoStatus.Information = 0i64;
 Irp->IoStatus.Status = 0;
 IofCompleteRequest(Irp, 0);
 return 0i64;
}
```

## 开始getflag

到这里有两种思路：
1.把上面的加密代码抄下来，然后提取出需要的数据，然后计算出flag
2.动态调试，输入上面解密到的16个按键，编写应用程序通过DeviceIoControl使用控制码0x222404与驱动通讯获取flag

这里我选择了第二种（主要是那部分复杂的加密没看明白）

具体的流程是这样的：
1.配置windbg+win7虚拟机（这个是64位驱动）+ida双机调试环境
2.在入口处设置断点，使用驱动加载工具，ida中成功断下。
3.在windows版本号判断时，修改windows信息为0xDEADBEEF
4.CPU硬件信息判断时，替换获取到信息为FakeIntel（记得本来后面多出来的部分要用\x00填充掉）
5.nop掉反调试线程
6.编写应用程序通过DeviceIoControl使用控制码0x222404与驱动通讯获取flag

```
#include <windows.h>
#include <stdio.h>

void getflag()
{
   DWORD  z = 0;
   char buffer[0x64] = {0};
   HANDLE LINK;
   //“■■”■■■■■■■
   LINK = CreateFileW(L"\\\\.\\DancingKeys",0,FILE_SHARE_READ|FILE_SHARE_WRITE,NULL,OPEN_EXISTING,0,NULL);
   DeviceIoControl(LINK, 0x222404,buffer,0x64,buffer,0x64,&z,(LPOVERLAPPED)NULL);
   printf("%s\n", buffer);
   //■■■■■■■■
   CloseHandle(LINK);
}


int main(int argc, char *argv[])
{
   getflag();
   Sleep(100000);
   return 0;
}
```

这里在驱动的IoControl处理例程中还有一处调试检测，记得要过掉（下断点，修改标志位绕过即可）
PS：如果不想这么麻烦，有些地方可以静态patch掉（patch驱动程序需要修复pe文件头的校验和，并使用签名工具进行签名）

应用层程序的输出即为flag

点击收藏 | 0 关注 | 1

上一篇：渗透过程中可能要用到的Kali工具小总结 下一篇：区块链之智能合约入门

1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板