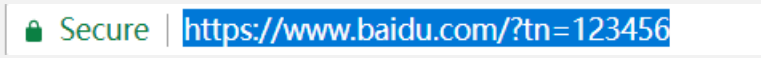
免杀与黑产流量劫持的二三事

安恒助力安全中国

汇报人：江皓秋

流量产业的劫持产业介绍

- 锁首页是什么，流量劫持能吃么



- 我不是针对谁，我是说在座的各位杀软都是垃圾！233333
- 想劫持用户该怎么办！

我不是针对谁，我是说在座的各位杀毒软件都是垃圾！

How to Bypass AV

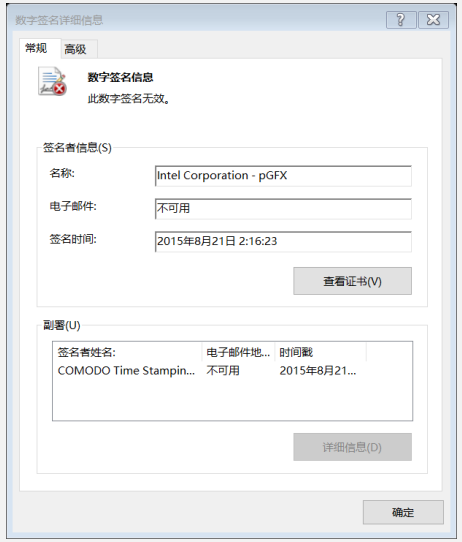
杀毒软件什么的统统都是浮云

- 杀毒软件的那些事
- 想要过掉杀毒软件的法子么总是要有的

杀软套路归套路

- 1.基于签名的检测（数字签名）
- 2.静态程序分析（特征码，导入表）
- 3.动态程序分析/沙盒分析技术(沙箱行为分析，查看调用API)
- 4.启发式分析（基于可用数据分析 PE结构 DNA相似性问题 比如字符串 pdb路径，溯源列表）
- 5.信息熵检测（PE结构区段的大小）
- 6.进程调用链条检测与内存监测
- 7.其他常见检测技术
  - 7.1 混淆检测，加壳检测，加密检测 rcx寄存器的循环

1. 基于签名的检测（数字签名）

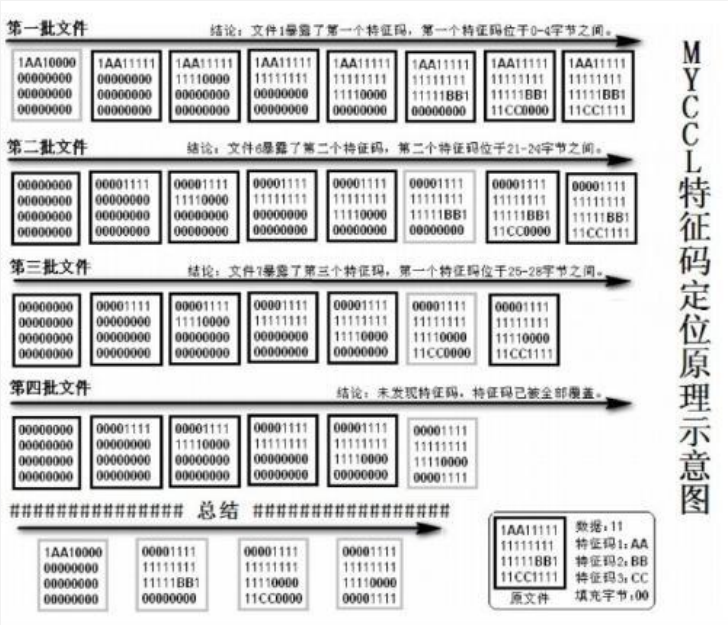


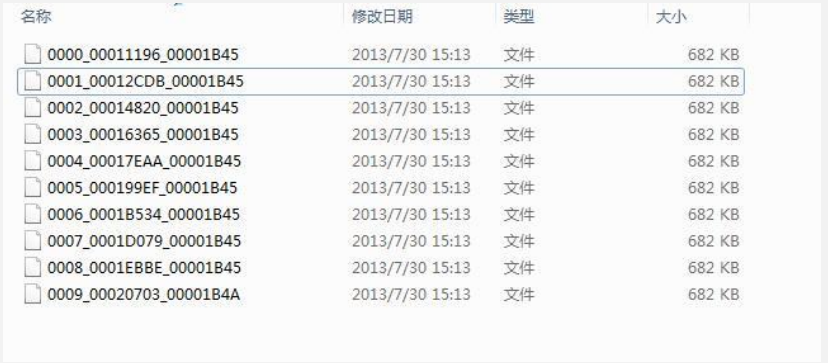
Bypass：

1、驱动人生涉嫌黑产？北信源数字签名被冒领？百度内部暗刷流量？

2、攻击windows数字证书体系？

2.静态程序分析（特征码）





- ByPass：Myccl定位特征码定位
  - 等价指令替换 add eax,1 <-> sub eax,-1
  - 花指令
  - 先加密保存成为数组到时候执行运行自己解密
  - Shellcode 寄生在外壳中
- Waring：Myccl分割模式的对抗 ecx循环计数器的对抗

2. 静态程序分析（导入表）

[基本] [区段] [数据表] [导入] [导出] [资源] [重定位] [异常] [.Net]

Dll 名称	O_Thunk	TimeStamp	FowardChain	Name1	Thunk
USER32.dll	00019330	00000000	00000000	0001936E	00019158
MSVCR120D...	00019268	00000000	00000000	00019474	00019090
KERNEL32.dll	000191D8	00000000	00000000	00019736	00019000

ThunkFOA	ThunkValue	Hint	Func Name
00006730	00019360	0000024D	MessageBoxW

```
    MessageBox(NULL, _T("World"), _T("Hello"), NULL);
00CE1013 56                push        esi
00CE1014 68 10 21 CE 00    push        0CE2110h
00CE1019 68 18 21 CE 00    push        0CE2118h
00CE101E 56                push        esi
00CE101F FF 15 9C 20 CE 00 call        dword ptr ds:[0CE209Ch]
```

```
0x00CE2110  48 65 6c 6c 6f 00 00 00  Hello.
0x00CE2118  57 6f 72 6c 64 00 00 00  World.
```

Address	Length	Type	String
.rdata:00402100	00000006	C	Hello
.rdata:00402108	00000006	C	World
.rdata:0040235E	0000000B	C	USER32.dll
.rdata:004024A8	0000000D	C	MSVCR120.dll
.rdata:004025...	0000000D	C	KERNEL32.dll

2. 静态程序分析（导入表, 字符串）

```
pMyMsgBox MyMessageBox = NULL;
//1、没处理
//MessageBox(NULL, _T("World"), _T("Hello"), NULL);

//2、处理
TCHAR szTitle[] = { _T('w'), _T('o'), _T('r'), _T('l'), _T('d'), _T('\0') };
TCHAR szBuf[] = { _T('H'), _T('e'), _T('l'), _T('l'), _T('o'), _T('\0') };
TCHAR szUser32[] = { _T('U'), _T('s'), _T('e'), _T('r'), _T('3'), _T('2'), _T('.'), _T('D'), _T('L'), _T('L'), _T('\0') };
TCHAR szMsgBox[] = { _T('M'), _T('e'), _T('s'), _T('s'), _T('a'), _T('g'), _T('e'), _T('B'), _T('o'), _T('x'), _T('A'), _T('\0') };
HMODULE hMod = LoadLibrary(szUser32);
if (hMod != NULL)
{
    MyMessageBox = (pMyMsgBox)GetProcAddress(hMod, szMsgBox);
}
MyMessageBox(NULL, szBuf, szTitle, NULL);
```

Dll 名称	O_Thunk	TimeStamp	FowardChain	Name1	Thunk
KERNEL32.dll	00002290	00000000	00000000	0000234E	00002000
MSVCR120.dll	000022BC	00000000	00000000	0000249A	0000202C

```
    HMODULE hMod = LoadLibrary(szUser32);
008A1011 8D 45 E0             lea         eax,[ebp-20h]
008A1014 C7 45 EC 77 6F 72 6C mov         dword ptr [ebp-14h],6C726F77h
008A101B 50                   push        eax
008A101C 33 F6                xor         esi,esi
008A101E 66 C7 45 F0 64 00    mov         word ptr [ebp-10h],64h
008A1024 C7 45 F4 48 65 6C 6C mov         dword ptr [ebp-0Ch],6C6C6548h
008A102B 66 C7 45 F8 6F 00    mov         word ptr [ebp-8],6Fh
008A1031 C7 45 E0 55 73 65 72 mov         dword ptr [ebp-20h],72657355h
008A1038 C7 45 E4 33 32 2E 44 mov         dword ptr [ebp-1Ch],442E3233h
008A103F 66 C7 45 E8 4C 4C    mov         word ptr [ebp-18h],4C4Ch
008A1045 C6 45 EA 00          mov         byte ptr [ebp-16h],0
008A1049 C7 45 D4 4D 65 73 73 mov         dword ptr [ebp-2Ch],7373654Dh
008A1050 C7 45 D8 61 67 65 42 mov         dword ptr [ebp-28h],42656761h
008A1057 C7 45 DC 6F 78 41 00 mov         dword ptr [ebp-24h],41786Fh
008A105E FF 15 00 20 8A 00    call        dword ptr ds:[008A2000h]
```

Address	Length	Type	String
.rdata:0040234E	0000000D	C	KERNEL32.dll
.rdata:0040249A	0000000D	C	MSVCR120.dll

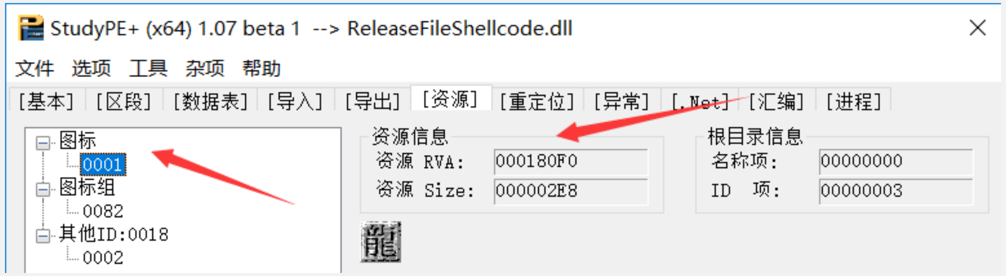
- ByPass：
  - 字符串栈化
  - 动态调用
  - Waring：过长的字符串任然存在被优化到全局idata段的可能性

2. 静态程序分析（释放PE资源）

– PE结构资源段中的存放之殇
  • 资源段中PE特征是重点（mz头，表头，函数表 相似模型）
– PE结构资源段中加密之殇
  • 资源段中加过密之后信息密度非常高，而且无法类似其他特征数据（JPG，ICO），体积过大，无法持久化免杀

```
#pragma once

static const DWORD MxEncode_key1 = 0x1580E368;
static const DWORD MxEncode_key2 = 0xADB27863;

#pragma data_seg("UPX2")

BYTE MxEncode_bin[0x41178] = {
    0x38, 0x19, 0x45, 0xA5, 0x38, 0x8F, 0x43, 0x8D, 0x20, 0x11, 0x04, 0x00, 0x68, 0xE3, 0x80, 0x15,
    0x63, 0x78, 0xB2, 0xAD, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x62, 0xAF, 0xF2, 0x7C, 0x30, 0xA9, 0x81, 0xC1,
    0x2C, 0xE6, 0x90, 0x45, 0xB6, 0x95, 0xEB, 0x1F, 0x00, 0x83, 0x0B, 0xDF, 0x66, 0x47, 0x08, 0xF7,
    0x48, 0xEE, 0xF8, 0x9F, 0x16, 0x3A, 0x60, 0x21, 0x82, 0x7D, 0x74, 0x18, 0xC3, 0xA1, 0x22, 0x76,
    0x34, 0x75, 0xAF, 0x2C, 0x32, 0x4F, 0xDD, 0xF3, 0xA4, 0x42, 0x0B, 0x21, 0xDB, 0x4E, 0xA3, 0xAE,
    0xC3, 0xA7, 0xB3, 0xB6, 0xA0, 0x8C, 0x5F, 0x66, 0xD1, 0x23, 0x03, 0x7D, 0x85, 0xE6, 0xF0, 0x5E,
    0xF4, 0x4C, 0x2E, 0x75, 0x4A, 0x77, 0x3E, 0x14, 0x11, 0x03, 0x46, 0x65, 0xA7, 0xCF, 0x8B, 0x59,
    0x09, 0xEC, 0x16, 0x2A, 0x73, 0x90, 0xFF, 0x4F, 0x60, 0x55, 0xB7, 0x6D, 0xE8, 0xCF, 0xA0, 0xD0,
    0xBF, 0x00, 0x46, 0x8A, 0xF5, 0xFF, 0xCE, 0x1D, 0x7D, 0xF8, 0xFF, 0xCA, 0x09, 0x91, 0x77, 0x85,
```

– ByPass：
  • 资源数组化加密放在特殊段中
  • 使用低密度算法，不要使用压缩算法，推荐使用简单单字节异或或者变异base64

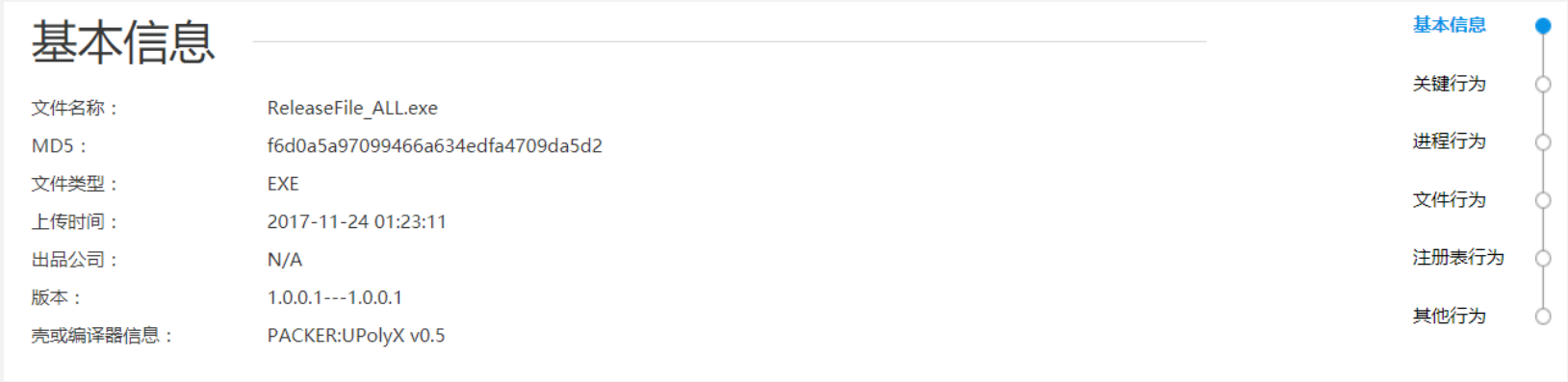



```
fprintf(fp, "#pragma once\n\n");
fprintf(fp, "static const DWORD %s_key1 = 0x%08X;\n", szParaName, k1);
fprintf(fp, "static const DWORD %s_key2 = 0x%08X;\n\n", szParaName, k2);

if (uFlag == 1)//upx1
{
    fprintf(fp, "#pragma data_seg(\"UPX1\")\n\n");
    fprintf(fp, "BYTE %s_bin[0x%X] = {\n\t", szParaName, total + sizeof(MyFileStruct));
}
else if (uFlag == 2)//upx2
{
    fprintf(fp, "#pragma data_seg(\"UPX1\")\n\n");
    fprintf(fp, "BYTE %s_bin[0x%X] = {\n\t", szParaName, total + sizeof(MyFileStruct));
}
else
{
    fprintf(fp, "static const BYTE %s_bin[0x%X] = {\n\t", szParaName, total + sizeof(MyFileStruct));
}
```

```
BYTE *pb;
int i = 0;
//输出头部分
pb = (BYTE*)pmyFileStruct;
for (; i < sizeof(MyFileStruct); i++)
{
    fprintf(fp, "0x%02X, ", *pb++);
    if ((i + 1) % 16 == 0)
        fprintf(fp, "\n\t");
}

pb = (BYTE *)pbuf;
for (; i < total + sizeof(MyFileStruct); i++)
{
    fprintf(fp, "0x%02X, ", *pb++);
    if ((i + 1) % 16 == 0)
        fprintf(fp, "\n\t");
}
```

3. 动态程序分析/沙盒分析技术

- 关键API监控
- 敏感目录访问控制
- 注册表
- 虚拟机检测
- 服务写入
- 流程记录
- 污点数据跟踪
- 尝试条件触发
- DNA关联（PE信息）
- **等等**



3.动态程序分析/沙盒分析技术

- ByPass SandBox:
- 虚拟机检测：
  - 特权指令 Vmware为真主机与虚拟机之间提供了相互沟通的通讯机制，它使用“IN”指令来读取特定端口的数据以进行两机通讯，但由于IN指令属于特权指令，在处于保护模式下的真机上执行此指令时，除非权限允许，否则将会触发类型为“EXCEPTION_PRIV_INSTRUCTION”的异常，而在虚拟机中并不会发生异常
  - 指纹特征 BIOS，USB控制器，显卡，网卡地址，注册表信息等等
- Shadowcall调用：
  - 偷取系统函数代码以此来避开函数头部分的hook，用以绕过流程记录检测
- 代码挖坑：
  - 利用虚拟机拒绝特性（你要的给不了算了一起gg吧）
    - *分配并填充大内存*
      - *一般超大内存申请，在反病毒的虚拟机中直接会拒绝由此检测虚拟机或者沙箱存在*
    - *成百上千的递增*
      - *利用检测ecx寄存器特性进行万次循环，在反病毒的虚拟机中直接会拒绝由此检测虚拟机或者沙箱存在*
  - 利用虚拟机模拟特性（要什么给什么，尽量让你跑）
    - *尝试打开系统进程*
      - 代码会尝试打开一般是拥有所有权限的4号系统进程。如果这个代码没有运行在系统MIC和会话0下，这个将会失败
      - *尝试打开一个不存在URL*
  - 利用虚拟机监测特性
    - 时间差，在虚拟机中回去hook sleep这种函数然后跳过，完全可以在之前和之后记录时间相减之后与sleep应该得时间进行对比来判断是否在虚拟机中
    - 父进程，如果父进程也是自己的话才开始启动核心代码

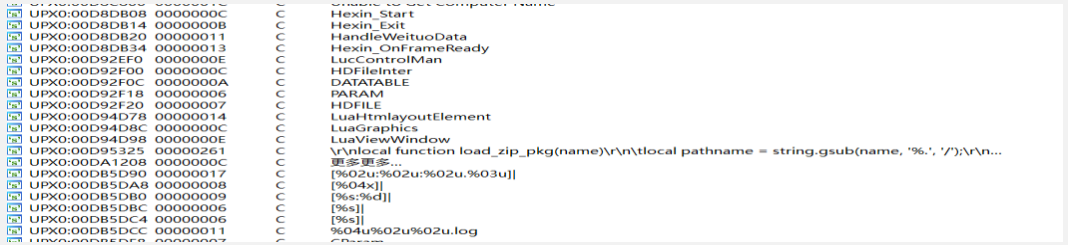
4. 启发式分析

名称	V Addr.	V Size	R Offset	R Size	标志
.text	00001000	00055892	00000400	00055A00	60000020
.rdata	00057000	0002413E	00055E00	00024200	40000040
.data	0007C000	000036B4	0007A000	00001000	C0000040
.gfids	00080000	00000344	0007B000	00000400	40000040
.tls	00081000	00000009	0007B400	00000200	C0000040
.rsrc	00082000	0002A038	0007B600	0002A200	40000040
.reloc	000AD000	00003FA0	000A5800	00004000	42000040

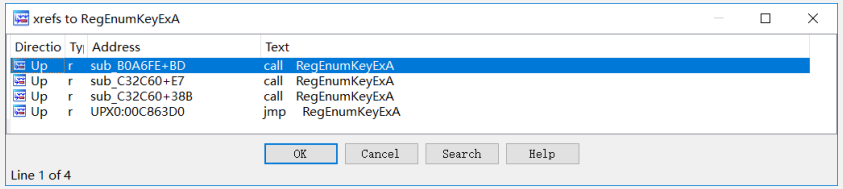
- 基于可用数据分析 综合度量值 积分制
- PE结构（区段大小，数字证书）
  - DNA相似性问题
    - pdb路径

```
0544c90h: 59 3A 5C 56 69 65 77 45 6E 67 69 6E 65 5C 56 69 ; Y:\ViewEngine\Vi
0544ca0h: 65 77 2B 2B 5C 52 65 6C 65 61 73 65 5C 50 43 48 ; ew++\Release\PCH
0544cb0h: 75 6E 74 65 72 36 34 2E 70 64 62 00 00 00 00 00 ; unter64.pdb.....
```

    - 字符串



    - API调用流程与交叉引用

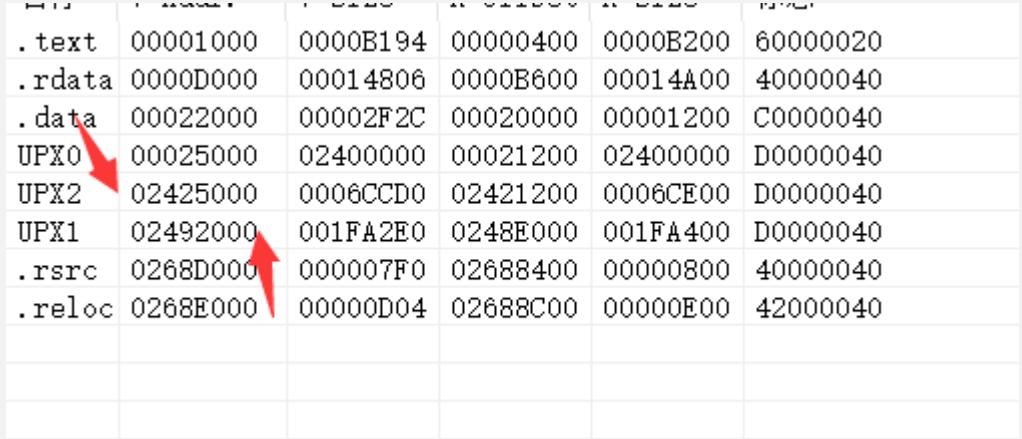


- 参考文章：http://www.docin.com/p-972425133-f3.html

5.信息熵检测（德国小红伞的最爱）

- 1、PE区段大小检查：
  - 检查pe区段信息熵是否存在异常：比如idata段与code段比例，如果idata段比率大但是代码并不多的话，很大会怀疑偷藏恶意代码在数组中，因为数组编译到idata段，然后会报告加密者

- 2、PE区段数量的检查：
  - 区段数量过多或者过少并且体积过大的话也会报存在加密者可能性



.text	00001000	0000B194	00000400	0000B200	60000020
.rdata	0000D000	00014806	0000B600	00014A00	40000040
.data	00022000	00002F2C	00020000	00001200	C0000040
UPX0	00025000	02400000	00021200	02400000	D0000040
UPX2	02425000	0006CCD0	02421200	0006CE00	D0000040
UPX1	02492000	001FA2E0	0248E000	001FA400	D0000040
.rsrc	0268D000	000007F0	02688400	00000800	40000040
.reloc	0268E000	00000D04	02688C00	00000E00	42000040

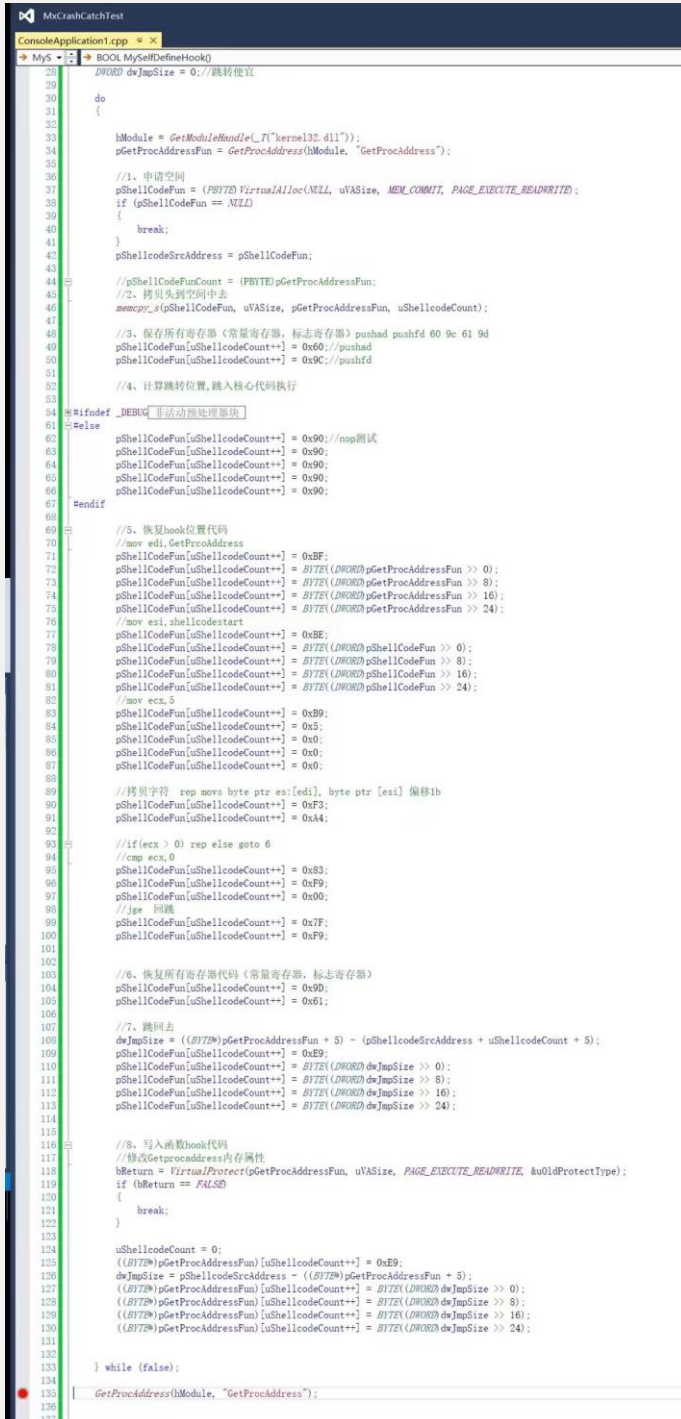
5.信息熵检测（德国小红伞的最爱）

- 3、区段属性：
  - 编译器分配的几个常用节的属性基本是固定一致的，自己新开辟的节要注意属性问题
- 4、区段内部熵值大小：
  - 区段大片留有空白典型的外壳释放器特征，因为无法确定需要释放的文件具体有多大很可能编译的时候留空白过大，过小的话也很有可能认为是释放器

- ByPass：
  - 1、化整为零多申请几个节一般可以一到三个，不要把所有需要释放的文件放在一个节里，尤其是加过密的数据，数据密度高信息熵高很容易报毒
  - 2、自己拿捏好数据存放在段中的大小，这个需要多次调试，保证尾部留有的空白是符合pe结构和编译器分配的
  - 3、申请节的属性需要控制好

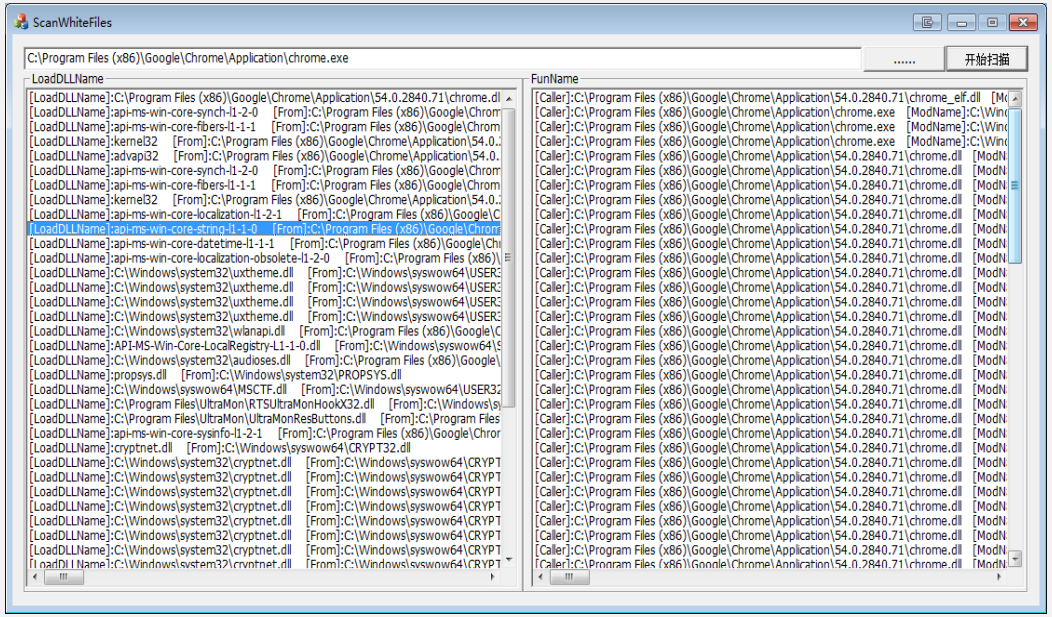
6. 进程调用链检测与内存监测

- 1、调用链检测：
  - 根据执行高危操作的进程的父进程来判断是否是用户执行（如父进程是否是Explorer.exe）
- 2、内存监测
  - 对于可疑程序启动之后检查申请内存的代码是否有可能被用于执行shellcode，如果有直接拦截
- 3、栈回溯
  - 对于已经被标记为可疑进程的程序在进行高危操作的时候一律拦截并且检查调用栈，如果调用方不可信直接拦截并且出现警告



6. 进程调用链条检测与内存监测

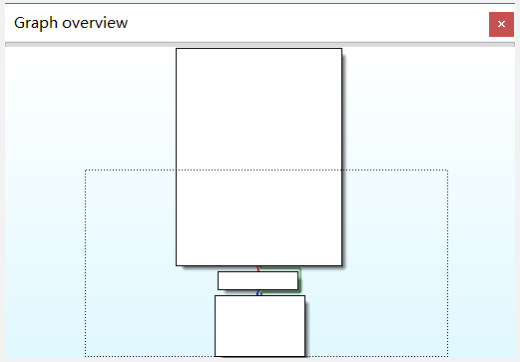
- ByPass：
  - 白加黑
    - 利用dll劫持漏洞来完成攻击（好的白加黑难找？）
  - 利用已经过白的程序的漏洞（目前最好的方案也是最难的方案）
    - 利用已经过白的一部分程序找出其存在的漏洞比如读取文件，然后想办法溢出并且控制流程之后再去执行恶意shellcode，则大功可成
  - 交保护费！！！！！！怎么交就不讲了反正某些数字的杀软并不是什么好鸟！

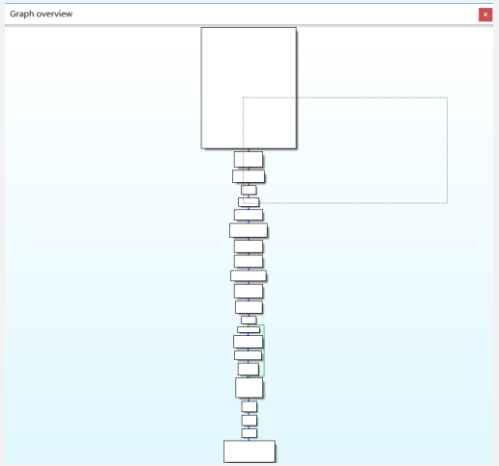


7. 其他常见检测技术

- 混淆检测：
  - 代码乱序，膨胀，虚拟化变异，特征明显vmhandle与水印
- 加壳检测：
  - OEP与区段关系，入口点特征
- 加密检测：
  - Rcx/ecx 寄存器 自动dump密文

上述代码将会加大杀毒软件查杀PE积分的值，当积分超过一定值的时候会判定为可疑进程






```
                pushf
                clc
                test    edi, 7FB45C19h
                test    eax, edx
                push    edi
                xor     di, 2F1Dh
                add     edi, edx
                push    edx
                rol     edi, 89h
                push    esi
                push    ebp
                rol     esi, 0D8h
                push    ebx
                push    ecx
                add     ebx, 4C74364h
                inc     ebp
                shl     ebp, 8Dh
                push    eax
                movzx   ebp, bp
                mov     eax, 0
                stc
                push    eax
                bts     di, cx
                sub     ebx, eax
                mov     esi, [esp+24h+arg_0]
                add     esi, 55ED0CD4h
                sal     ebp, 2Ch
                neg     esi
                ror     esi, 3
                rcl     bx, cl
                sbb     edi, 0C101BACh
                neg     esi
                neg     ebp
                add     esi, eax
                cmp     esi, 6D040B8Dh
                mov     ebp, esp
                add     edi, ecx
                lea     esp, [esp-0C0h]

loc_43BB5F:                             ; CODE XREF: .vmp0:loc_413113↑j
                                        ; .vmp0:loc_417326↑j ...
                mov     ebx, esi
                neg     ah
                mov     eax, 0
                sub     ebx, eax
                btr     ax, 70h
                shld    eax, eax, 91h

loc_43BB73:                             ; DATA XREF: sub_43BAF5:loc_43BB73↓o
                lea     edi, loc_43BB73
                mov     eax, [esi]
                cmp     dh, 5
                jmp     loc_44CC82
sub_43BAF5      endp ; sp-analysis failed
```

8.一个不是那么优雅的案例

- Shellcode自动化生成：
  - 位置无关
    - 字符串使用栈化
  - Payload需要自己解析外部引用
    - PEB中获取加载模块链条（fs寄存器）
    - 扫描模块的eat中名称与自身计算的hash值是否匹配
    - 匹配则根据eat中对应关系获取地址
  - PayLoad需要保存调用栈的状态用于返回（编译器做好了）
- 字符串加密
  - 全部使用栈化代码，让字符串不保存在静态常量区而是保存在栈中
- PE加密数组化
  - 将PE数据藏在数组中放入特殊构造的区段
- 白加黑利用
  - 利用扫描白加黑劫持工具配合IDA，制作劫持dll
- 参考：
- http://www.freebuf.com/articles/system/27122.html

开发角度谈谈那些年我们锁过的首页

黑产锁首页的探讨

正面怼锁首页你需要学会的各种姿势

- 驱动层锁首页
- 流量劫持锁首页

1、简单玩法(注入浏览器HOOK getcommandlineW)

简介

- Getcommandline是一个非ring0函数，直接读取peb->ProcessParameters->CommandLine.*Buffer*
- 函数不需要进入ring0，这是浏览器获取参数的最后一步，直接hook修改返回值非常方便稳定

优缺点

- 优点：
  - 技术实现简单
  - Hook方案成熟高效稳定，不容易BSOD
  - 不会出现驱动锁首页字节长度的覆盖问题
- 缺点：
  - 容易被人干死，比如直接卸载非法hook模块
  - 针对注入可以一刀切拦截掉所有注入手法（比如VirtualProtect ）

2、进阶玩法(利用*PsSetCreateProcessNotifyRoutine*)

简介

- 注册创建进程callback函数
- 判断进程是否为想要的进程
- 通过callback函数带来的信息附加到进程中获取PEB的CommandLine.*Buffer*
- *直接强制修改该字段，修改参数*
- *保护进程VirtualProtect函数防止被注入hook，注入必来此函数*

技术难点

- *32/64bit判断（解析PE寻找标志位）*
- *进程名获取问题（超长问题）*
- *两份PEB表的解析与摘取（进程中的两份PEB）*
- *多浏览器问题（检查父进程）*

优缺点

- 优点：
  - 稳定，不需要hook注入
- 缺点or问题：
  - 有无参数，参数过长问题，覆盖or溢出？？？
  - 竞品对抗问题，
    - 无法加载驱动或者直接加载时候入口代码被直ret
    - *PsSetCreateProcessNotifyRoutine的链表被摘掉或者满了无法再挂载回调（自保护）*
    - *自己并非最后一个PsSetCreateProcessNotifyRoutine链表中的函数，修改可能并不生效*
  - 浏览器参数校验问题
    - 基本无解（ring3 PatchCheck）
- 解决思路探讨：
  - Ring0-ring3 重启解决参数过长问题/短url跳转
  - 保证自己是最后一个链表中的函数，永远让自己处于最后修改cmdline的地位

3、高级玩法(WFP OR NDIS小端口驱动)

简介

- 方案一：直接NDIS小端口过滤驱动拦截出口数据包->数据重定向到ring3代理服务器上->ssl解密->修改数据包->重新封包->回发到驱动层->出去
- 方案二：内核实现完成端口模型，将ssl解密框架移植到内核使用ssl解密，封包动作都在内核完成，市场上已经有至少五款高度稳定的成品出现

优缺点

- 优点：
  - 完美，无痕迹（所有数据包自己解析构造666）
  - 收益非常高所有推广类页面都可以有计费号实现持续化流量劫持
- 缺点：
  - 难度太高，时间代价大

逆向角度谈谈那些年我们锁过的首页

从后面怼锁首页姿势探讨

1、逆向浏览器配置文件

简介

- 直接逆向浏览器配置文件
- 方案一：裸奔型
  - chrome
- 方案二：复合校验型（文档）
  - Opera，360
- 方案三：强加密型（sqlite3数据库）
  - Maxthon 搜狗浏览器
- 方案四：驱动保护型（文档）
  - 金山猎豹浏览器（minifilter）

优缺点

- 优点：
  - 稳定，不受任何劫持影响，不存在对抗问题
  - 可以一并实现收藏夹,新建首页项目的流量变现
- 缺点：
  - 难度高部分使用强加密，逆向很头疼

锁首页引发的思考

- 互联网入口的重要性（流量为王）
- 流量为王的天下，道德沦丧究竟到了什么地步（黑产，流量劫持）
- 理性的看待现在中国GFW体制下的互联网（GFW下独特的中国互联网文化）