

蠕蟲惡意程式 Win32.Parite 深度分析與掃描程式 - TeamT5

 teamt5.org/tw/posts/how-to-detect-and-recover-from-virus-win32-parite/



2023.02.14IR Team

Share:

TeamT5 資安事件應變團隊(Incident Response Team) 在某次事件調查中，發現受害主機中的執行檔遭受一款名為 Win32.Parite 的蠕蟲惡意程式感染。受到感染的主機，會散播至其他執行檔並等待攻擊者連線；若連線成功後，攻擊者可在受感染的主機上執行各種通訊和數據收集行為，影響甚深。

我們於本文詳細分析該惡意程式，也開發掃描程式與說明解毒方式，協助企業提升資安防禦能量。我們並呼籲尚在使用 WinXP/2003/7 等32位元作業系統的組織、企業，建議可以使用掃描程式來掃內網系統。

惡意程式 Win32.Parite 基本資訊

68

/ 71

?

Community Score

68 security vendors and 1 sandbox flagged this file as malicious

1e918b08f1d0b9c58a8236d56bb8105fc91c3b79a63ef6a8510ea05961f9cd93

257.98 KB

2022-09-21 21:39:04 UTC

Size

4 months ago

EXE

nedwp.exe

peexe stealth spreader overlay

DETECTION

DETAILS

RELATIONS

BEHAVIOR

CONTENT

TELEMETRY

COMMUNITY 1

Submissions ①

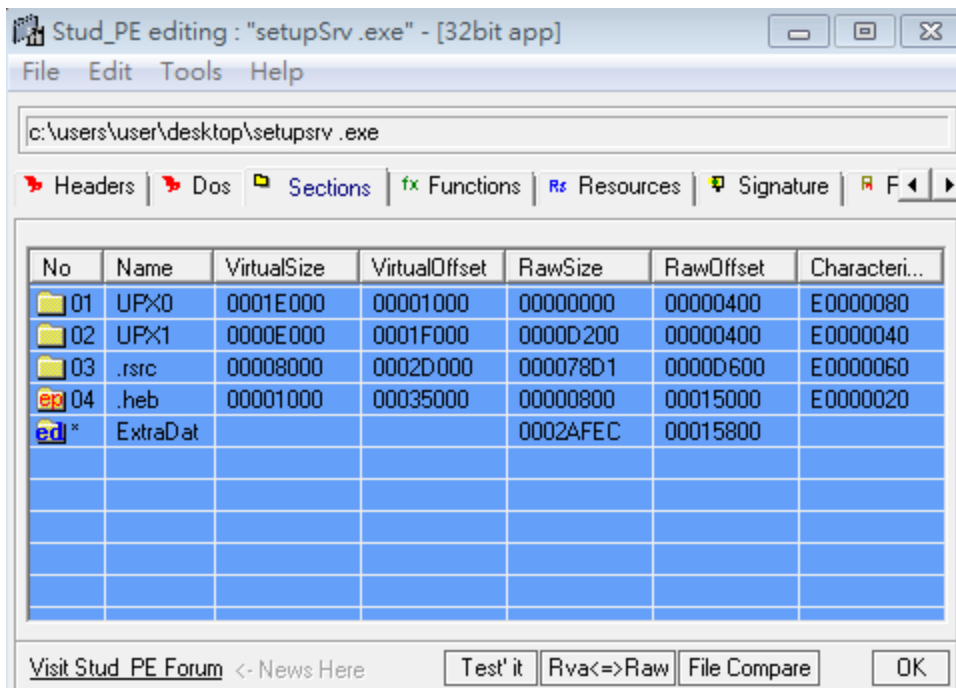
Date	Name	Source	Country
2018-01-11 00:08:29 UTC	LSASSRV.EXE	📁 c4025027 - api	US
2018-01-11 13:38:57 UTC	369362589f98130fd83a91b74531345c68d168b8	✉ 0786e468 - email	FR
2022-09-20 20:46:59 UTC	1e918b08f1d0b9c58a8236d56bb8105fc91c3b79a63ef6a8510ea05961f9cd93-dropped.bin	📁 2f654f3b - api	US

圖片來源: VirusTotal

惡意程式分析

Win32.Parite - Shellcode

TeamT5 取得惡意檔案，可以使用 Stud_PE 觀察，可以看到節區(Sections)裡面多了一個名稱為 .heb 節區，並且程式進入點(Entry Point)被指向該節區。Win32.Parite 病毒的特徵，就是會將自己插入在最後一個節區，該節區名稱是隨機的字母組合。



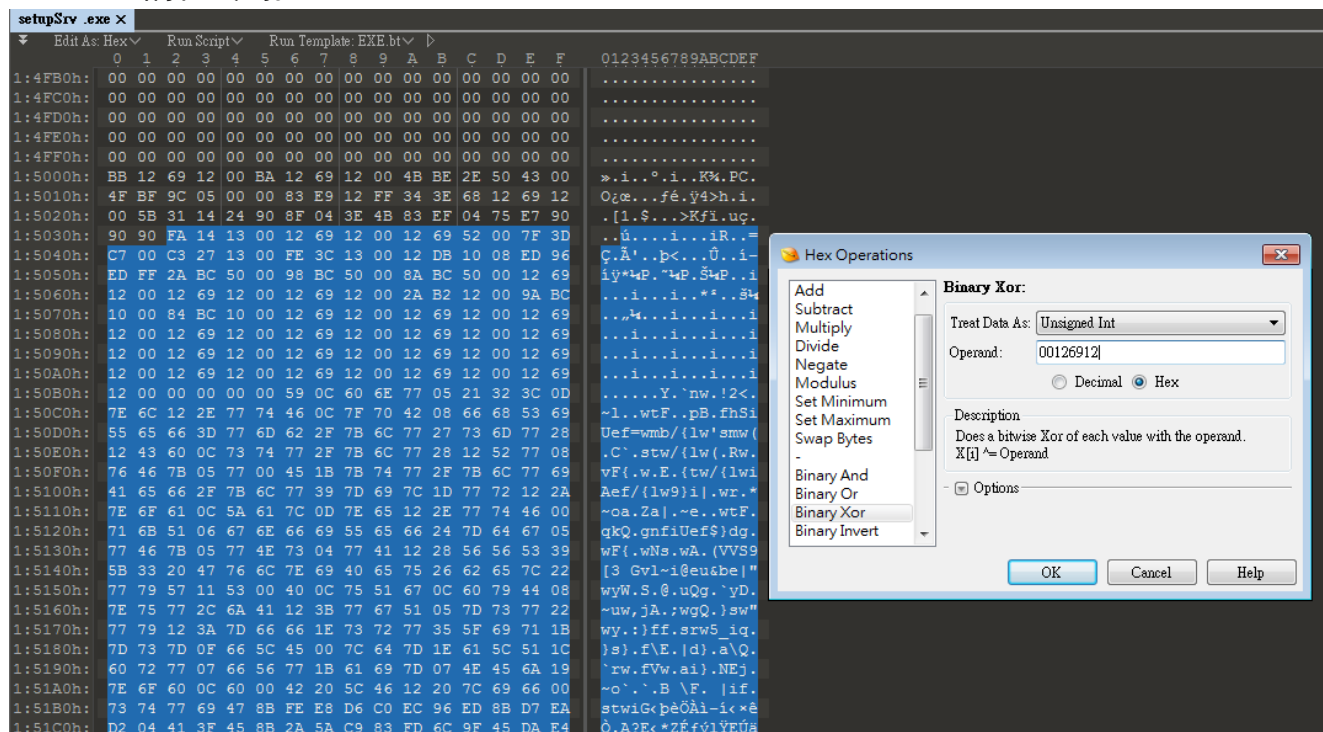
進入點是一段小小的解碼程式(Decoder)，暫存器中的edx為xor解碼的運算數值，而esi為編碼資料的位址，edi為編碼資料的大小。解碼時，運算一次為4 bytes，全部處理的範圍是0x43502E+4~0x43502E+0x59C，因為使用[push/pop] dword ptr [esi+edi]的手法，所以這裡的xor的目標位址才會是[esp]。

```

.heb_:00435000      public start
.heb_:00435000 start:
.heb_:00435000      mov     ebx, 126912h
.heb_:00435005      mov     edx, 126912h
.heb_:0043500A      dec     ebx
.heb_:0043500B      mov     esi, (offset loc_43502D+1)
.heb_:00435010      dec     edi
.heb_:00435011      mov     edi, 59Ch
.heb_:00435016      loc_435016:                                ; CODE XREF: .heb_:loc_43502D↓j
.heb_:00435016      sub     ecx, 12h
.heb_:00435019      push    dword ptr [esi+edi]
.heb_:0043501C      push    126912h
.heb_:00435021      pop     ebx
.heb_:00435022      xor     [esp], edx
.heb_:00435025      nop
.heb_:00435026      pop     dword ptr [esi+edi]
.heb_:00435029      dec     ebx
.heb_:0043502A      sub     edi, 4
.heb_:0043502D      loc_43502D:                                ; DATA XREF: .heb_:0043500B↑o
.heb_:0043502D      jnz     short loc_435016
.heb_:0043502F      nop
.heb_:00435030      nop
.heb_:00435031      nop
.heb_:00435032      cli
.heb_:00435033      adc     al, 13h
.heb_:00435035      add     [edx], dl
.heb_:00435037      imul    edx, [edx], 52691200h
.heb_:0043503D      add     [edi+3Dh], bh
.heb_:00435040      mov     dword ptr [eax], 1327C3h

```

在理解了上述的組語後，可以使用010 Editor來做XOR的運算。解開後可以發現是一段 shellcode 的程式碼。



setupSrv.exe* X																
Edit As: Hex Run Script Run Template: EXE.bt																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0123456789ABCDEF															
1:4FE0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1:4FF0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1:5000h:	BB	12	69	12	00	BA	12	69	12	00	4B	BE	2E	50	43	00
1:5010h:	4F	BF	9C	05	00	00	83	E9	12	FF	34	3E	68	12	69	12
1:5020h:	00	5B	31	14	24	90	8F	04	3E	4B	83	EF	04	75	E7	90
1:5030h:	90	90	E8	7D	01	00	00	00	00	00	00	00	40	00	6D	54
1:5040h:	D5	00	D1	4E	01	00	EC	55	01	00	00	B2	02	08	FF	FF
1:5050h:	FF	FF	38	D5	42	00	8A	D5	42	00	98	D5	42	00	00	00
1:5060h:	00	00	00	00	00	00	00	00	00	00	38	DB	00	00	88	D5
1:5070h:	02	00	96	D5	02	00	00	00	00	00	00	00	00	00	00	00
1:5080h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1:5090h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1:50A0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1:50B0h:	00	00	12	69	12	00	4B	65	72	6E	65	6C	33	32	2E	64
1:50C0h:	6C	6C	00	47	65	74	54	65	6D	70	50	61	74	68	41	00
1:50D0h:	47	65	74	54	65	6D	70	46	69	6C	65	4E	61	6D	65	41
1:50E0h:	00	43	72	65	61	74	65	46	69	6C	65	41	00	52	65	61
1:50F0h:	64	46	69	6C	65	00	57	72	69	74	65	46	69	6C	65	00
1:5100h:	53	65	74	46	69	6C	65	50	6F	69	6E	74	65	72	00	43
1:5110h:	6C	6F	73	65	48	61	6E	64	6C	65	00	47	65	74	54	69
1:5120h:	63	6B	43	6F	75	6E	74	00	47	65	74	4D	6F	64	75	6C
1:5130h:	65	46	69	6C	65	4E	61	6D	65	41	00	41	44	56	41	50
1:5140h:	49	33	32	2E	64	6C	6C	00	52	65	67	4F	70	65	6E	4B

有趣的是此蠕蟲惡意程式也有變形的功能，觀察三個被感染的檔案，從中發現到shellcode的指令都不一樣，但目的都是相同的，都是要取得解碼的運算數值、編碼資料的位址與編碼資料的大小。

IDA View-A	IDA View-A	IDA View-A
<pre> .xun_:00529000 public start .xun_:00529000 start: .xun_:00529000 dec esi .xun_:00529001 mov eax, 59Ch .xun_:00529001 sub ebx, 0FFFFFFE2h .xun_:00529006 push esi .xun_:00529009 push (offset loc_529021+1) .xun_:0052900E pop esi .xun_:0052900F push 2791E2h .xun_:00529014 pop ecx .xun_:00529015 ; CODE XREF: .xun_:00529015 loc_529015 .xun_:00529015 sub edx, 0FFFFFFE2h .xun_:00529018 xor [eax+esi], ecx .xun_:00529018 sub eax, 2 .xun_:0052901E sub eax, 2 .xun_:00529021 ; DATA XREF: .xun_:00529021 loc_529021 .xun_:00529021 jnz short loc_529015 .xun_:00529023 nop .xun_:00529024 nop .xun_:00529025 nop .xun_:00529026 or ch, ah </pre>	<pre> .xun_:00414000 assume es:nothing, ss:nothing, d .xun_:00414000 .xun_:00414000 start: .xun_:00414000 push 2791E2h .xun_:00414000 pop esi .xun_:00414000 push 59Ch .xun_:00414000 pop edx .xun_:0041400C mov edi, 2791E2h .xun_:00414011 mov edi, (offset loc_414023+ .xun_:00414016 inc ecx .xun_:00414017 push 2791E2h .xun_:00414017 pop esi .xun_:0041401D loc_41401D: .xun_:0041401E dec ecx .xun_:0041401E xor [edx+edi], esi .xun_:00414021 inc eax .xun_:00414022 dec edx .xun_:00414023 loc_414023: .xun_:00414023 sub edx, 3 .xun_:00414026 jnz short loc_41401D </pre>	<pre> .wtq_:004B2000 public start .wtq_:004B2000 start: .wtq_:004B2000 mov edx, 279102h .wtq_:004B2000 push 279102h .wtq_:004B2005 pop esi .wtq_:004B200A sub edx, 0FFFFFFD2h .wtq_:004B200E push offset loc_4B202C .wtq_:004B2013 pop edi .wtq_:004B2014 sub edx, 0FFFFFFD2h .wtq_:004B2017 mov ebx, 59Ch .wtq_:004B201C loc_4B201C: .wtq_:004B201C sub edx, 0FFFFFFD2h .wtq_:004B201F push dword ptr [ebx+edi] .wtq_:004B2022 xor [esp], esi .wtq_:004B2025 pop dword ptr [ebx+edi] .wtq_:004B2028 inc eax .wtq_:004B2029 sub ebx, 4 .wtq_:004B202C loc_4B202C: .wtq_:004B202C jnz short loc_4B201C .wtq_:004B202E nop </pre>

解碼後，繼續分析shellcode程式碼，解開後的第一個組語的指令是一個call，下面緊接著是一個Parite自訂義的結構，裡面比較有趣的是原程式的進入點(OEP)是被編碼保護起來的。

```

.heb_:00435031      nop
.heb_:00435032      call     sub_4351B4
.heb_:00435032 ; -----
.heb_:00435037      db 0
.heb_:00435038      db 0
.heb_:00435039      db 0
.heb_:0043503A      dd 400000h      ; ImageBase
.heb_:0043503E      dd 0D5546Dh      ; OEP(RVA) Encoded
.heb_:00435042      dd 14ED1h      ; Length from last section end to pe end
.heb_:00435046      dd 155ECh      ; DLL File Offset
.heb_:0043504A      dd 802B200h      ; DLL File Size, MSB=08 is flags
.heb_:0043504E      dd 0FFFFFFFh
.heb_:00435052      dd offset LoadLibraryA ; Pointer IAT LoadLibrary & GetProcAddress
.heb_:00435056      dd 42D58Ah      ; Kernel32.dll Import Name #1
.heb_:0043505A      dd 42D598h      ; Kernel32.dll Import Name #2
.heb_:0043505E      dd 0          ; Kernel32.dll Import Table Offset
.heb_:00435062      dd 0          ; Kernel32.dll Import Table #1
.heb_:00435066      dd 0          ; Kernel32.dll Import Table #2
.heb_:0043506A      dd 0DB38h      ; Kernel32.dll Import Table Offset
.heb_:0043506E      dd 2D588h      ; Kernel32.dll Import Table #1
.heb_:00435072      dd 2D596h      ; Kernel32.dll Import Table #2
.heb_:00435076      dd 0          ; Kernel32.dll Import Table Offset
.heb_:0043507A      dd 0          ; Kernel32.dll Import Table #1
.heb_:0043507E      dd 0          ; Kernel32.dll Import Table #2
.heb_:00435082      dd 0
.heb_:00435086      dd 0
.heb_:0043508A      dd 0          ; Original Section Data Start
.heb_:0043508E      dd 0
.heb_:00435092      dd 0
.heb_:00435096      dd 0
.heb_:0043509A      dd 0
.heb_:0043509E      dd 0
.heb_:004350A2      dd 0
.heb_:004350A6      dd 0
.heb_:004350AA      dd 0
.heb_:004350AE      dd 0          ; Original Section Data End
.heb_:004350B2      dd 126912h      ; DLL xor Key
.heb_:004350B6      aKernel32Dll_0 db 'Kernel32.dll',0
.heb_:004350C3      aGettemppatha db 'GetTempPathA',0

```

Shellcode的主程式的功能僅有幾項：

- GetDeclareAPI：取得要時用的API位址存於Stack
- RegQuery_PINF：查詢PINF機碼是否存在，內容存放病毒主體路徑
- Call_DLL_Initiate：呼叫主程式DLL中的Initiate，裡面包含解開EXE原始進入點
- Drop_DLL_File：如果PINF不存在時，會丟出病毒主體(DLL)及建立PINF登入檔機碼數值

```

char sub_4351B4()
{
    char v0; // bl
    char *v1; // edi
    char *v2; // esi
    char result; // al
    unsigned __int16 v4; // [esp+Ch] [ebp-140h]
    char v5; // [esp+Eh] [ebp-13Eh]
    char v6; // [esp+114h] [ebp-38h]
    char *retaddr; // [esp+150h] [ebp+4h]

    v0 = 0;
    v1 = retaddr - 5;
    v2 = retaddr + 127; // kernel32.dll
    result = GetDeclareAPI_435254((int (__stdcall **)(int))&v6, (int)(retaddr - 5), (int)v2);
    if ( result )
    {
        result = RegQuery_PINF_435360((int)&v4, (int)&v6, (int)v2);
        if ( result && v4 >= 9u ) // infected
        {
            result = Call_DLL_Initiate_4353C8(&v5, &v6, v1, v2);
            v0 = result;
        }
        if ( !v0 )
        { // not yet infected
            result = Drop_DLL_File_435400(&v5, &v6, v1);
            if ( result )
                result = Call_DLL_Initiate_4353C8(&v5, &v6, v1, v2);
        }
        retaddr = (char *)(((DWORD *)v1 + 2) + (((DWORD *)v1 + 3)));
    }
    return result;
}

```

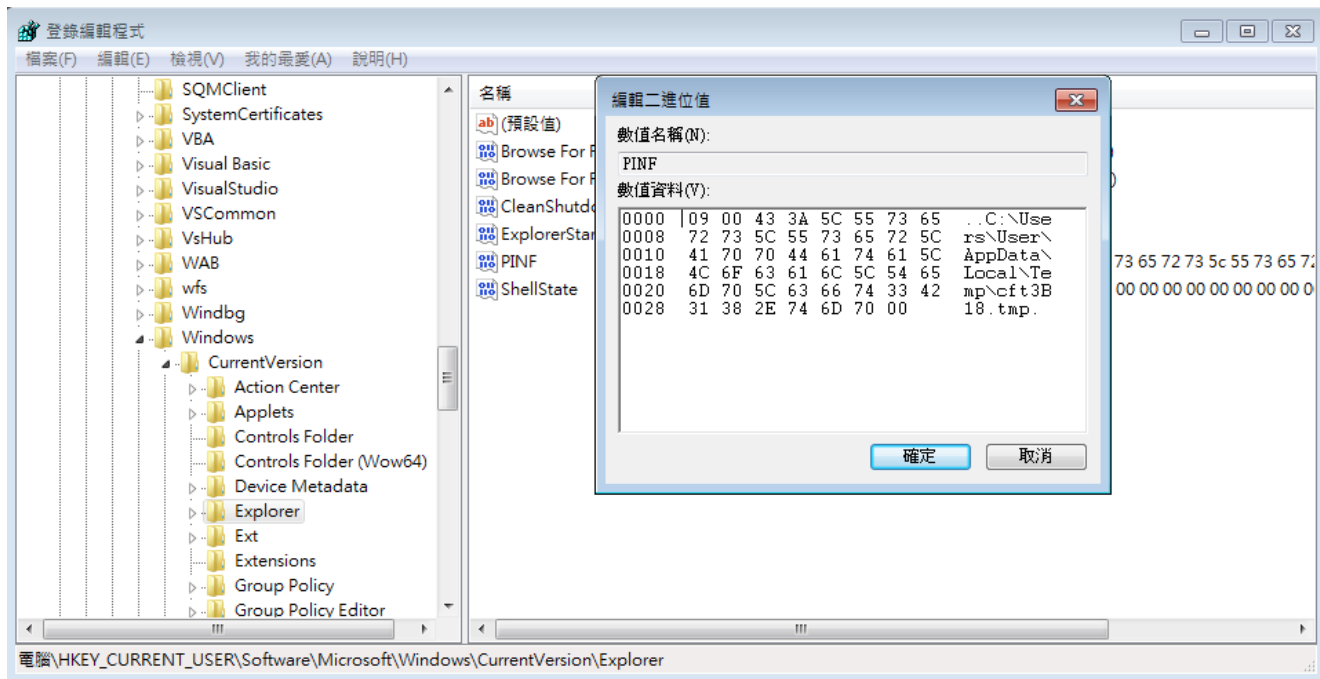
Shellcode透過GetDeclareAPI(sub_435254)取得一些必要的Windows API後，接著透過RegQuery_PINF(sub_435360)查詢位於HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer的PINF鍵值。

```

.heb_:00435360 RegQuery_PINF_435360 proc near          ; CODE XREF: sub_4351B4+39↑p
.heb_:00435360
.heb_:00435360 var_8          = dword ptr -8
.heb_:00435360 var_4          = dword ptr -4
.heb_:00435360 arg_0          = dword ptr  8
.heb_:00435360 arg_4          = dword ptr  0Ch
.heb_:00435360 arg_8          = dword ptr  10h
.heb_:00435360
.heb_:00435360      push    ebp
.heb_:00435361      mov     ebp, esp
.heb_:00435363      add     esp, 0FFFFFFF8h
.heb_:00435366      lea     eax, [ebp+var_4]
.heb_:00435369      push    ebx
.heb_:0043536A      push    esi
.heb_:0043536B      push    edi
.heb_:0043536C      xor     ebx, ebx
.heb_:0043536E      mov     edi, [ebp+arg_8]
.heb_:00435371      mov     esi, [ebp+arg_4]
.heb_:00435374      mov     [ebp+var_8], 106h
.heb_:0043537B      push    eax
.heb_:0043537C      push    20019h
.heb_:00435381      lea     edx, [edi+0BDh] ; Software\Microsoft\Windows\CurrentVersion\Explorer
.heb_:00435387      push    0
.heb_:00435389      push    edx
.heb_:0043538A      push    80000001h          ; HKEY_CURRENT_USER
.heb_:0043538F      call    dword ptr [esi+2Ch] ; RegOpenKeyExA
.heb_:00435392      test    eax, eax
.heb_:00435394      jnz     short loc_4353BC
.heb_:00435396      lea     ecx, [ebp+var_8]
.heb_:00435399      add     edi, 0F0h
.heb_:0043539F      push    ecx
.heb_:004353A0      push    [ebp+arg_0]
.heb_:004353A3      push    0
.heb_:004353A5      push    0
.heb_:004353A7      push    edi          ; PINF
.heb_:004353A8      push    [ebp+var_4]
.heb_:004353AB      call    dword ptr [esi+30h] ; RegQueryValueExA
.heb_:004353AE      test    eax, eax
.heb_:004353B0      setz    bl
.heb_:004353B3      and     ebx, 1
.heb_:004353B6      push    [ebp+var_4]
.heb_:004353B9      call    dword ptr [esi+34h] ; RegCloseKey

```

在被感染的機器中可以看到PINF的數值內容是\x09\x00，這地方的9為病毒版本，以\x00分隔，後再接DLL路徑。



若該鍵值存在並且前2 bytes的數值大於等於9，shellcode會透過 Call_DLL_Initate(sub_4353C8)載入病毒主體的DLL模組，並使用GetProcAddress取得Initiate函數位址，傳入000435032的組態位址後並且跳入執行。

```
.heb_:004353C8 Call_DLL_Initate_4353C8 proc near          ; CODE XREF: sub_4351B4+59↑p
.heb_:004353C8                                         ; sub_4351B4+86↑p
.heb_:004353C8
.heb_:004353C8 arg_0 = dword ptr 8
.heb_:004353C8 arg_4 = dword ptr 0Ch
.heb_:004353C8 arg_8 = dword ptr 10h
.heb_:004353C8 arg_C = dword ptr 14h
.heb_:004353C8
.heb_:004353C8 push ebp
.heb_:004353C9 mov ebp, esp
.heb_:004353CB push ebx
.heb_:004353CC mov ebx, [ebp+arg_4]
.heb_:004353CF push [ebp+arg_0] ; PINF=\x09\x00C:\Users\User\AppData\Local\Temp\cft3B18.tmp
.heb_:004353D2 call dword ptr [ebx] ; LoadLibraryA
.heb_:004353D4 test eax, eax
.heb_:004353D6 jz short loc_4353F7
.heb_:004353D8 mov edx, [ebp+arg_C]
.heb_:004353DB add edx, 0F5h
.heb_:004353E1 push edx ; Initiate
.heb_:004353E2 push eax
.heb_:004353E3 call dword ptr [ebx+4] ; GetProcAddress
.heb_:004353E6 test eax, eax
.heb_:004353E8 jz short loc_4353F7
.heb_:004353EA push [ebp+arg_8] ; 00435032
.heb_:004353ED call eax ; Call export "Initate" from dll module
.heb_:004353EF test al, al
.heb_:004353F1 jz short loc_4353F7
.heb_:004353F3 mov al, 1
.heb_:004353F5 jmp short loc_4353F9
.heb_:004353F7 ; -----
.heb_:004353F7 loc_4353F7: ; CODE XREF: Call_DLL_Initate_4353C8+E↑j
.heb_:004353F7 ; Call_DLL_Initate_4353C8+20↑j ...
.heb_:004353F7 xor eax, eax
.heb_:004353F9 loc_4353F9: ; CODE XREF: Call_DLL_Initate_4353C8+2D↑j
.heb_:004353F9 pop ebx
.heb_:004353FA pop ebp
.heb_:004353FB retn 10h
```

若該鍵值不存在，shellcode則會透過Drop_DLL_File(sub_435400)將位在後方的病毒主體(DLL)解碼後丟出來，接著執行Call_DLL_Initate(sub_4353C8)載入病毒主體的DLL模組。

```
.heb_:00435400 Drop_DLL_File_435400 proc near          ; CODE XREF: sub_4351B4+70↑p
.heb_:00435400
.heb_:00435400 var_2914      = byte ptr -2914h
.heb_:00435400 var_114      = byte ptr -114h
.heb_:00435400 var_10       = byte ptr -10h
.heb_:00435400 var_C        = dword ptr -0Ch
.heb_:00435400 var_8        = dword ptr -8
.heb_:00435400 var_1        = byte ptr -1
.heb_:00435400 arg_0        = dword ptr 8
.heb_:00435400 arg_4        = dword ptr 0Ch
.heb_:00435400 arg_8        = dword ptr 10h
.heb_:00435400
.heb_:00435400      push    ebp
.heb_:00435401      mov     ebp, esp
.heb_:00435403      push    eax
.heb_:00435404      mov     eax, 2
.heb_:00435409
.heb_:00435409 loc_435409:                          ; CODE XREF: Drop_DLL_File_435400+11↓j
.heb_:00435409      add     esp, 0FFFFFF04h
.heb_:0043540F      push    eax
.heb_:00435410      dec     eax
.heb_:00435411      jnz     short loc_435409
.heb_:00435413      mov     eax, [ebp-4]
.heb_:00435416      add     esp, 0FFFFFF6F0h
.heb_:0043541C      push    ebx
.heb_:0043541D      push    esi
.heb_:0043541E      push    edi
.heb_:0043541F      mov     ebx, [ebp+arg_4]
.heb_:00435422      mov     esi, [ebp+arg_0]
.heb_:00435425      push    104h
.heb_:0043542A      lea     eax, [ebp+var_114] ; Self EXE
.heb_:00435430      push    eax
.heb_:00435431      push    0
.heb_:00435433      call    dword ptr [ebx+28h] ; GetModuleFileNameA
.heb_:00435436      push    0
.heb_:00435438      push    1
.heb_:0043543A      push    3
.heb_:0043543C      push    0
.heb_:0043543E      push    1
.heb_:00435440      lea     edx, [ebp+var_114]
.heb_:00435446      push    80000000h
.heb_:0043544B      push    edx
.heb_:0043544C      call    dword ptr [ebx+10h] ; CreateFileA
.heb_:0043544F      mov     edi, eax
.heb_:00435451      cmp     edi, 0FFFFFFFFh
.heb_:00435454      jz      loc_43558F
.heb_:0043545A      lea     eax, [ebp+var_114]
.heb_:00435460      push    eax
.heb_:00435461      push    104h
.heb_:00435466      call    dword ptr [ebx+8] ; GetTempPathA
.heb_:00435469      call    dword ptr [ebx+24h] ; GetTickCount
```

而DLL的偏移位址(Offset)及檔案大小(Size)，還有編碼的XOR key，都可以在組態中看到。

```
.heb_:00435031      nop
.heb_:00435032      call     sub_4351B4
.heb_:00435032 ; -----
.heb_:00435037      db 0
.heb_:00435038      db 0
.heb_:00435039      db 0
.heb_:0043503A      dd 400000h      ; ImageBase
.heb_:0043503E      dd 0D5546Dh     ; OEP(RVA) Encoded
.heb_:00435042      dd 14ED1h      ; Length from last section end to pe end
.heb_:00435046      dd 155ECh      ; DLL File Offset
.heb_:0043504A      dd 802B200h     ; DLL File Size, MSB=08 is flags
.heb_:0043504E      dd 0FFFFFFFh
.heb_:00435052      dd offset LoadLibraryA ; Pointer IAT LoadLibrary & GetProcAddress
.heb_:00435056      dd 42D58Ah      ; Kernel32.dll Import Name #1
.heb_:0043505A      dd 42D598h      ; Kernel32.dll Import Name #2
.heb_:0043505E      dd 0           ; Kernel32.dll Import Table Offset
.heb_:00435062      dd 0           ; Kernel32.dll Import Table #1
.heb_:00435066      dd 0           ; Kernel32.dll Import Table #2
.heb_:0043506A      dd 0DB38h      ; Kernel32.dll Import Table Offset
.heb_:0043506E      dd 2D588h      ; Kernel32.dll Import Table #1
.heb_:00435072      dd 2D596h      ; Kernel32.dll Import Table #2
.heb_:00435076      dd 0           ; Kernel32.dll Import Table Offset
.heb_:0043507A      dd 0           ; Kernel32.dll Import Table #1
.heb_:0043507E      dd 0           ; Kernel32.dll Import Table #2
.heb_:00435082      dd 0
.heb_:00435086      dd 0
.heb_:0043508A      dd 0           ; Original Section Data Start
.heb_:0043508E      dd 0
.heb_:00435092      dd 0
.heb_:00435096      dd 0
.heb_:0043509A      dd 0
.heb_:0043509E      dd 0
.heb_:004350A2      dd 0
.heb_:004350A6      dd 0
.heb_:004350AA      dd 0
.heb_:004350AE      dd 0           ; Original Section Data End
.heb_:004350B2      dd 126912h     ; DLL xor Key
.heb_:004350B6      aKernel32Dll_0 db 'Kernel32.dll',0
.heb_:004350C3      aGettemppatha db 'GetTempPathA',0
```

在Drop_DLL_File(sub_435400)函數裡面會呼叫Decoder_Content(sub_4355A0)函數，其內容是使用xor dword進行編解碼。

```

.heb_:004355A0 Decoder_Content_4355A0 proc near          ; CODE XREF: Drop_DLL_File_435400+112↑p
.heb_:004355A0                                         ; Drop_DLL_File_435400+160↑p
.heb_:004355A0
.heb_:004355A0 arg_0          = dword ptr 8
.heb_:004355A0 arg_4          = dword ptr 0Ch
.heb_:004355A0 arg_8          = dword ptr 10h
.heb_:004355A0
.heb_:004355A0      push     ebp
.heb_:004355A1      mov      ebp, esp
.heb_:004355A3      push     ebx
.heb_:004355A4      mov      ebx, [ebp+arg_0]
.heb_:004355A7      mov      edx, [ebp+arg_4]
.heb_:004355AA      mov      ecx, [ebp+arg_8]
.heb_:004355AD      test     ecx, ecx
.heb_:004355AF      jns      short loc_4355B4
.heb_:004355B1      add      ecx, 3
.heb_:004355B4
.heb_:004355B4 loc_4355B4:          ; CODE XREF: Decoder_Content_4355A0+F↑j
.heb_:004355B4      sar      ecx, 2
.heb_:004355B7      xor      eax, eax
.heb_:004355B9      cmp      ecx, eax
.heb_:004355BB      jle      short loc_4355C5
.heb_:004355BD
.heb_:004355BD loc_4355BD:          ; CODE XREF: Decoder_Content_4355A0+23↓j
.heb_:004355BD      xor      [edx+eax*4], ebx
.heb_:004355C0      inc      eax
.heb_:004355C1      cmp      ecx, eax
.heb_:004355C3      jg      short loc_4355BD
.heb_:004355C5
.heb_:004355C5 loc_4355C5:          ; CODE XREF: Decoder_Content_4355A0+1B↑j
.heb_:004355C5      pop      ebx
.heb_:004355C6      pop      ebp
.heb_:004355C7      retn     0Ch

```

Win32.Parite - 病毒主體

解碼出來的DLL使用UPX加殼，病毒主體模組本身是使用Delphi所撰寫，所導出的Initiate函數第23行有一個解碼原始進入點(OEP)的函數。

```

1 char __cdecl Initiate(_DWORD **a1)
2 {
3     char result; // a1
4     unsigned __int16 v2; // [esp+Ch] [ebp-13Ch]
5     Registry::TRegistry *v3; // [esp+114h] [ebp-34h]
6     HANDLE hObject; // [esp+118h] [ebp-30h]
7     __int16 v5; // [esp+12Ch] [ebp-1Ch]
8     __int16 v6; // [esp+12Eh] [ebp-1Ah]
9     int *v7; // [esp+140h] [ebp-8h]
10    int v8; // [esp+144h] [ebp-4h]
11
12    __InitExceptBlockLDT();
13    v5 = 8;
14    if ( a1 )
15    {
16        if ( a1[19] )
17            *a1[19] = sub_401F7C;
18        if ( a1[20] )
19            *a1[20] = sub_401FC0;
20        if ( a1[21] )
21            *a1[21] = sub_40200C;
22    }
23    Decode_OEP_4022BC(a1, 0);
24    v3 = (Registry::TRegistry *)Registry::TRegistry::TRegistry((Registry::TRegistry *)dword_40C94C);
25    hObject = OpenMutexA(0x1F0001u, 0, "Resident");
26    v5 = 32;
27    if ( !hObject || (result = sub_4019A8(v3, &v2, 262)) != 0 && v2 < 9u && v2 >= 2u )
28        result = (unsigned int)SetWindowsHookExA(4, AttachHook, 0, 0);
29    v5 = 20;
30    if ( hObject )
31        result = CloseHandle(hObject);

```

Decode_OEP(sub_4022BC)函數中的程式碼，主要是取得組態區塊中的43504D、43506E的二個陣列位址，經由sub_40232C函數進行運算，產生出16 bytes的數值，拆成4組DWORD值，與被編碼的OEP進行四次的XOR運算，最後的結果為原始OEP，有了OEP就能順利的修復被感染的PE執行檔。

```

1 char *__fastcall Decode_OEP_4022BC(int a1, char a2)
2 {
3     int v2; // esi
4     char *result; // eax
5     signed int v4; // ebx
6     signed int v5; // ebx
7     char v6; // [esp+Ch] [ebp-1Ch]
8     int v7; // [esp+1Ch] [ebp-Ch]
9     char *v8; // [esp+20h] [ebp-8h]
10    int v9; // [esp+24h] [ebp-4h]
11
12    v2 = a1; // 00435032
13    v7 = 0;
14    result = (char *)(a1 + 0x1B); // 00435032+1B=0043504D
15    if ( a2 )
16    {
17        *result = 8;
18    }
19    else if ( *result != 8 )
20    {
21        return result;
22    }
23    v9 = v2 + 0x3C; // 00435032+3C=0043506E
24    v8 = &v6;
25    v4 = 0;
26    do
27    {
28        sub_40232C(*(_BYTE *)(v2 + 0x30 + v4), (int)&v6, &v7); // 00435032+30=00435062
29        sub_40232C(*(_BYTE *)(v9 + v4++), (int)&v6, &v7);
30    }
31    while ( v4 < 8 );
32    v5 = 0;
33    do
34    {
35        result = v8;
36        *(_DWORD *)(v2 + 12) ^= *(_DWORD *)&v8[4 * v5++]; // 00435032+0C = OEP
37    }
38    while ( v5 < 4 );
39    return result;
40 }

```

接著使用SetWindowsHookExA設置全局鉤子，將DLL注入所有處理程序中。

```
1 char __cdecl Initiate(_DWORD **a1)
2 {
3     char result; // a1
4     unsigned __int16 v2; // [esp+Ch] [ebp-13Ch]
5     Registry::TRegistry *v3; // [esp+114h] [ebp-34h]
6     HANDLE hObject; // [esp+118h] [ebp-30h]
7     __int16 v5; // [esp+12Ch] [ebp-1Ch]
8     __int16 v6; // [esp+12Eh] [ebp-1Ah]
9     int *v7; // [esp+140h] [ebp-8h]
10    int v8; // [esp+144h] [ebp-4h]
11
12    __InitExceptBlockLDT();
13    v5 = 8;
14    if ( a1 )
15    {
16        if ( a1[19] )
17            *a1[19] = sub_401F7C;
18        if ( a1[20] )
19            *a1[20] = sub_401FC0;
20        if ( a1[21] )
21            *a1[21] = sub_40200C;
22    }
23    Decode_OEP_4022BC((int)a1, 0);
24    v3 = (Registry::TRegistry *)Registry::TRegistry::TRegistry((Registry::TRegistry *)dword_40C94C);
25    hObject = OpenMutexA(0x1F0001u, 0, "Resident");
26    v5 = 32;
27    if ( !hObject || (result = sub_4019A8(v3, &v2, 262)) != 0 && v2 < 9u && v2 >= 2u )
28        result = (unsigned int)SetWindowsHookExA(WH_CALLWNDPROC, AttachHook, 0, 0);
29    v5 = 20;
30    if ( hObject )
31        result = CloseHandle(hObject);
```

當被注入的處理程序為explorer.exe時，Win32.Parite則會開始執行蠕蟲的行為。

```

27 else if ( a2 == 1 && GetModuleFileNameA(0, &Filename, 0x104u) )
28 {
29     v7 = 20;
30     sub_452DF8(&v13);
31     ++v8;
32     sub_452E9C(&lpLibFileName, &v13);
33     --v8;
34     sub_452E6C(&v13, 2);
35     v7 = 32;
36     v12 = 0;
37     ++v8;
38     unknown_libname_186(0, &v12);
39     lpString2 = 0;
40     ++v8;
41     Sysutils::ExtractFileName(v12);
42     if ( lpString2 )
43         v3 = lpString2;
44     else
45         v3 = &byte_45447E;
46     v4 = lstrcmpiA("explorer.exe", v3) == 0;
47     --v8;
48     sub_452E6C(&lpString2, 2);
49     --v8;
50     sub_452E6C(&v12, 2);
51     if ( v4 )
52         Action_401C0C(); // worm main func
53 }
54 return 1;
55 }

```

Win32.Parite - 後門功能

在病毒主體DLL中的程式碼，可以看到一段有興趣的程式碼，Win32.Parite支援後門功能，後門利用UDP協議來發送指令，被控制端會監聽UDP 30167通訊埠等待接收指令，被控制端回報之結果會回傳到控制端的UDP 30168通訊埠，需要上傳資料時，則需要在控制端上監聽TCP 30169通訊埠，等待被控制端主動連線，此時就能上傳資料內容。

```

1 int __usercall TFuncModule_UdpOnDataReceived@<eax>(int a1@<eax>, int a2@<edx>, int a3@<ecx>, int a4, ...)
2 {
3     int v4; // esi
4     int v5; // ebx
5     int result; // eax
6     char *handle; // [esp+Ch] [ebp-2Ch]
7     int v8; // [esp+10h] [ebp-28h]
8     __int16 v9; // [esp+24h] [ebp-14h]
9     __int16 v10; // [esp+26h] [ebp-12h]
10    int v11; // [esp+30h] [ebp-8h]
11    va_list va; // [esp+44h] [ebp+Ch]
12
13    va_start(va, a4);
14    v4 = a3;
15    v5 = a1;
16    __InitExceptBlockLDT();
17    System::AnsiString::AnsiString((char *)va, (char *)va);
18    v11 = 2;
19    handle = (char *)operator new[](v4);
20    v9 = 20;
21    Nmudp::TNMUDP::ReadBuffer(*(Nmudp::TNMUDP **)(v5 + 80), handle, v4, &v8);
22    switch ( *handle )
23    {
24        case 'e':
25            sub_405FF0(v8);
26            break;
27        case 'y':
28            sub_4061B0(v8);
29            break;
30        case 'z':
31            sub_406404(v8);
32            break;
33    }

```

回報主機資訊

控制端發出e的字元到被控制端的UDP 30167通訊埠。

No.	Time	Source	Destination	Length	Protocol	Info
	1 2020-12-...	192.168.111.1	192.168.111.140	43	UDP	57811 → 30167 Len=1
<						
> Frame 1: 43 bytes on wire (344 bits), 43 bytes captured (344 bits) on interface > Ethernet II, Src: VMware_c0:00:01 (00:50:56:c0:00:01), Dst: VMware_fd:56:5f (00 > Internet Protocol Version 4, Src: 192.168.111.1, Dst: 192.168.111.140 > User Datagram Protocol, Src Port: 57811, Dst Port: 30167 > Data (1 byte)						
Data: 65						
[Length: 1]						
<						
0000	00 0c 29 fd 56 5f 00 50	56 c0 00 01 08 00 45 00	..).V_.P V.....E.			
0010	00 1d 25 e8 00 00 80 11	b5 09 c0 a8 6f 01 c0 a8	..%. o...			
0020	6f 8c e1 d3 75 d7 00 09	e3 51 65	o...u... .Qe			

被控制端會回報主機資訊到控制端的UDP 30168通訊埠。

No.	Time	Source	Destination	Length	Protocol	Info
4	2020-12-...	192.168.111.140	192.168.111.1	207	UDP	30167 → 30168 Len=165
<						
Frame 4: 207 bytes on wire (1656 bits), 207 bytes captured (1656 bits) on interface						
Ethernet II, Src: VMware_fd:56:5f (00:0c:29:fd:56:5f), Dst: VMware_c0:00:01 (00:						
Internet Protocol Version 4, Src: 192.168.111.140, Dst: 192.168.111.1						
User Datagram Protocol, Src Port: 30167, Dst Port: 30168						
Data (165 bytes)						
Data: 669d0000000500000001000000280a00000200000053657276696365205061636b203300						
[Length: 165]						
<						
0010	00 c1 02 4a 00 00 80 11	d8 03 c0 a8 6f 8c c0 a8	...	J		...o...
0020	6f 01 75 d7 75 d8 00 ad	9f c6 66 9d 00 00 00 05	o	u	u...	...f....
0030	00 00 00 01 00 00 00 28	0a 00 00 02 00 00 00 53		(		...S
0040	65 72 76 69 63 65 20 50	61 63 6b 20 33 00 7c 58	ervice	P	ack	3... X
0050	18 14 00 2f 33 9c 71 46	33 9c 71 0c fc 3c 02 10	.../3	qF	3	q...<...
0060	2c 14 00 5c b5 d6 02 53	5b c0 02 58 a6 c1 02 62	,...	\	...S	[...X...b

執行指令

控制端送出y的字元，接著DWORD值，1是顯示在畫面中，0是隱藏在背景中執行，隨後跟著要下達的指令(可接參數)，最後以\x00結尾。

No.	Time	Source	Destination	Length	Protocol	Info
23	2020-12-...	192.168.111.1	192.168.111.140	88	UDP	64844 → 30167 Len=46
<						
Frame 23: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on interface						
Ethernet II, Src: VMware_c0:00:01 (00:50:56:c0:00:01), Dst: VMware_fd:56:5f (00:						
Internet Protocol Version 4, Src: 192.168.111.1, Dst: 192.168.111.140						
User Datagram Protocol, Src Port: 64844, Dst Port: 30167						
Data (46 bytes)						
Data: 7901000000633a5c77696e646f77735c73797374656d33325c6e6f74657061642e657865						
[Length: 46]						
<						
0000	00 0c 29 fd 56 5f 00 50	56 c0 00 01 08 00 45 00	..)	V	_P V	E
0010	00 4a 25 ee 00 00 80 11	b4 d6 c0 a8 6f 01 c0 a8	J%		...	...o...
0020	6f 8c fd 4c 75 d7 00 36	2f c3 79 01 00 00 00 63	o	Lu	6 /	y....c
0030	3a 5c 77 69 6e 64 6f 77	73 5c 73 79 73 74 65 6d	:	\	w	indow s\system
0040	33 32 5c 6e 6f 74 65 70	61 64 2e 65 78 65 20 61	32\n	notep	ad.exe	a
0050	2e 74 78 74 00 00 00 00		.	txt		

當指令成功被執行後，被控制端會回傳g字元，後面接上Done的字串。

No.	Time	Source	Destination	Length	Protocol	Info
24	2020-12-...	192.168.111.140	192.168.111.1	48	UDP	30167 → 30168 Len=6
<						
Frame 24: 48 bytes on wire (384 bits), 48 bytes captured (384 bits) on interface						
Ethernet II, Src: VMware_fd:56:5f (00:0c:29:fd:56:5f), Dst: VMware_c0:00:01 (0						
Internet Protocol Version 4, Src: 192.168.111.140, Dst: 192.168.111.1						
User Datagram Protocol, Src Port: 30167, Dst Port: 30168						
Data (6 bytes)						
Data: 67446f6e6500						
[Length: 6]						
<						
0000	00 50 56 c0 00 01 00 0c	29 fd 56 5f 08 00 45 00	.PV.....).V...E.			
0010	00 22 02 6e 00 00 80 11	d8 7e c0 a8 6f 8c c0 a8	."n.....~...o...			
0020	6f 01 75 d7 75 d8 00 0e	78 90 67 44 6f 6e 65 00	o.u.u...x.gDone.			

上傳檔案

控制端發送出z的字元，接著檔案名稱，最後以\x00結尾，該檔案會被存放在%WINDIR%資料夾下。

No.	Time	Source	Destination	Length	Protocol	Info
1	2020-12-...	192.168.111.1	192.168.111.140	52	UDP	54629 → 30167 Len=10
<						
Frame 1: 52 bytes on wire (416 bits), 52 bytes captured (416 bits) on interface						
Ethernet II, Src: VMware_c0:00:01 (00:50:56:c0:00:01), Dst: VMware_fd:56:5f (00						
Internet Protocol Version 4, Src: 192.168.111.1, Dst: 192.168.111.140						
User Datagram Protocol, Src Port: 54629, Dst Port: 30167						
Data (10 bytes)						
Data: 7a746573742e74787400						
[Length: 10]						
<						
0000	00 0c 29 fd 56 5f 00 50	56 c0 00 01 08 00 45 00	..).V.P V.....E.			
0010	00 26 26 39 00 00 80 11	b4 af c0 a8 6f 01 c0 a8	.&&9.....o...			
0020	6f 8c d5 65 75 d7 00 12	18 1f 7a 74 65 73 74 2e	o..eu... ..ztest.			
0030	74 78 74 00		txt.			

控制端監聽TCP 30169通訊埠，等待被控制端連線成功後之後便可以傳送的資料內容，結束連線後，資料將被寫入到檔案裡面。

No.	Time	Source	Destination	Length	Protocol	Info
5	2020-12-...	192.168.111.1	192.168.111.140	58	TCP	30169 → 1133 [PSH, ACK] Seq=
> Frame 5: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface \Device > Ethernet II, Src: VMware_c0:00:01 (00:50:56:c0:00:01), Dst: VMware_fd:56:5f (00:0c:29: > Internet Protocol Version 4, Src: 192.168.111.1, Dst: 192.168.111.140 > Transmission Control Protocol, Src Port: 30169, Dst Port: 1133, Seq: 1, Ack: 1, Len: 4 Source Port: 30169 Destination Port: 1133 [Stream index: 0]						
0000	00 0c 29 fd 56 5f 00 50	56 c0 00 01 08 00 45 00	..).V_P V.....E.			
0010	00 2c 26 3b 40 00 80 06	74 b2 c0 a8 6f 01 c0 a8	.,&@... t...o...			
0020	6f 8c 75 d9 04 6d 2a 88	b6 88 6b 25 7e fc 50 18	o·u·m*· ··k%~·P·			
0030	fa f0 27 a6 00 00	74 65 73 74	..'...te st			

修復被感染的檔案

被感染的檔案差異

這裡分析一下被感染後的執行檔有什麼變化：

- Section數多1
- 原始進入點(OEP)被指向最後一個病毒所在的Section
- 載入記憶體映像檔(Image)變大
- 修改了.rdata的屬性
- 新增了新的病毒節區

Startup HTran.exe X

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF	
0000h:	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....yy..
0010h:	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	@.....
0020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	è.....
0030h:	00	00	00	00	00	00	00	00	00	00	00	00	E8	00	00	00	!.....
0040h:	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	!.....LiTh
0050h:	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
0060h:	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS
0070h:	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....\$.....
0080h:	80	C4	A4	AA	C4	A5	CA	F9	C4	A5	CA	F9	C4	A5	CA	F9	€A"ÀÈÙÀÈÙÀÈÙÀÈÙ
0090h:	79	EA	5C	F9	C7	A5	CA	F9	CD	DD	5F	F9	D6	A5	CA	F9	vèÙCVÈÙY ùòVÈÙ

Template Results - EXE.bt

Name	Value
struct IMAGE_SECTION_DATA Section[1]	.rdata
UCHAR Data[11264]	
struct IMAGE_SECTION_DATA Section[2]	.data
struct IMAGE_IMPORT_DESCRIPTOR ImportDescriptor[0]	WS2_32.dll
struct IMAGE_IMPORT_DESCRIPTOR ImportDescriptor[1]	kernel32.dll

HTran.exe- X

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF	
0000h:	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ.....yy..
0010h:	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	@.....
0020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	è.....
0030h:	00	00	00	00	00	00	00	00	00	00	00	00	E8	00	00	00	!.....
0040h:	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	!.....LiTh
0050h:	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno

Compare

Result	Address A	Size A	Address B	Size B	
Match	0h	EEh	0h	EEh	
Difference	EEh	1h	EEh	1h	NumberOfSection
Match	EFh	21h	EFh	21h	
Difference	110h	3h	110h	3h	AddressOfEntryPoint
Match	113h	26h	113h	26h	
Difference	139h	1h	139h	1h	SizeOfImage
Match	13Ah	F5h	13Ah	F5h	
Difference	22Fh	1h	22Fh	1h	.rdata CHARACTERISTICS IMAGE_SNC_TYPE_NO_PAD
Match	230h	28h	230h	28h	
Difference	258h	14h	258h	28h	SectionHeader Name: .xur
Match	26Ch	180h	280h	180h	

檔案比對中，還有最後被修改的二個地方，分別為C600、E9F8。

Result	Address A	Size A	Address B	Size B
Match	230h	28h	230h	28h
Difference	258h	14h	258h	28h
Match	26Ch	180h	280h	180h
Only in A	3ECh	14h		
Match	400h	C200h	400h	C200h
Difference	C600h	6h	C600h	6h
Match	C606h	23F2h	C606h	23F2h
Difference	E9F8h	6h	E9F8h	6h
Match	E9FEh	1A02h	E9FEh	1A02h
Only in B			10400h	64FE2h

可以看到這二個地方，其實是Import Tables中kernel32.dll的前二個函數，感染後的檔案因為沒寫自己取API的Shellcode，因此靠著在導入表中插入LoadLibraryA，GetProcAddress來確保Shellcode中能正常使用此二個函數取得要用的API，在執行Shellcode時會將其還原。

21	ord:23 rva2iat: 0000E16C	21	ord:23 rva2iat: 0000E16C
22	ord:3 rva2iat: 0000E170	22	ord:3 rva2iat: 0000E170
23		23	
24	KERNEL32.dll	24	KERNEL32.dll
25	SetLastError ord:1139 rva2iat: 0000E000	25	LoadLibraryA ord:0 rva2iat: 0000E000
26	ReadFile ord:960 rva2iat: 0000E004	26	GetProcAddress ord:0 rva2iat: 0000E004
27	GetProcessHeap ord:586 rva2iat: 0000E008	27	GetProcessHeap ord:586 rva2iat: 0000E008
28	SetEndOfFile ord:1107 rva2iat: 0000E00C	28	SetEndOfFile ord:1107 rva2iat: 0000E00C

解毒程式

在了解被感染的檔案有什麼改變之後，我們便可以撰寫解毒程式來修復被感染的檔案。解毒程式基本上做了以下的事情：

1. 檢查進入點是否落在最後一個節區，且節區名稱為 .
2. 讀取xor解碼的運算數值、編碼資料的位址與編碼資料的大小
3. 解碼並讀取Parite的組態
4. 解碼原始進入點(oep)
5. NumberOfSections減1
6. SizeOfImage減最後一個節區的虛擬大小
7. 刪除最後一個節區頭
8. 修改.rdata節區的屬性
9. 修復導入表

使用方法

Usage: fix.exe <file or directory>

```
C:\Windows\system32\cmd.exe
G:\Users\User\Desktop>fix.exe
Usage: fix.exe <file or directory>
```

測試

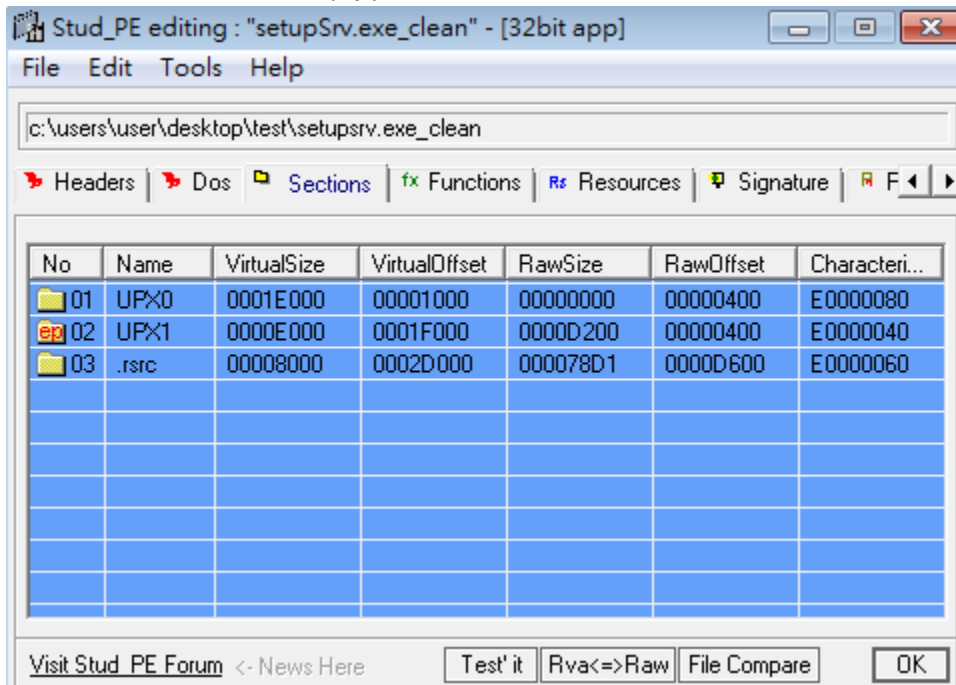
> fix.exe setupSrv.exe

```

C:\Users\User\Desktop>fix.exe test\setupSrv.exe
2023/02/10 11:11:28 Try to fix: test\setupSrv.exe
2023/02/10 11:11:28 Error parsing the import table. AddressOfData overlaps with THUNK_DATA for THUNK at:
RVA 0x4
2023/02/10 11:11:28 data at RVA can't be fetched. Corrupt header?
2023/02/10 11:11:28 Error parsing the import table. AddressOfData overlaps with THUNK_DATA for THUNK at:
RVA 0x4
2023/02/10 11:11:28 data at RVA can't be fetched. Corrupt header?
2023/02/10 11:11:28 Error parsing the import table. AddressOfData overlaps with THUNK_DATA for THUNK at:
RVA 0x4
2023/02/10 11:11:28 data at RVA can't be fetched. Corrupt header?
2023/02/10 11:11:28 Fixed: test\setupSrv.exe_clean
2023/02/10 11:11:28 main, execution time 62.5ms

```

修復好的檔案會在同樣路徑，檔案名稱以`_clean`結尾。查看修復的檔案，可以看到最後一個被感染的節區與進入點(ep)都已經修正。



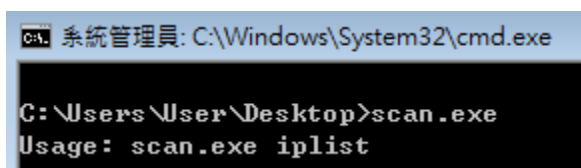
檢查主機是否受害

我們了解當控制端發送出的字元到被控制端的UDP 30167通訊埠後，被控制端會回報主機資訊到控制端的UDP 30168通訊埠。因此我們可以利用這點開發一個掃描程式，對內網主機傳送e的字元並在發送端監聽UDP 30168通訊埠等待回傳資訊。

掃描程式

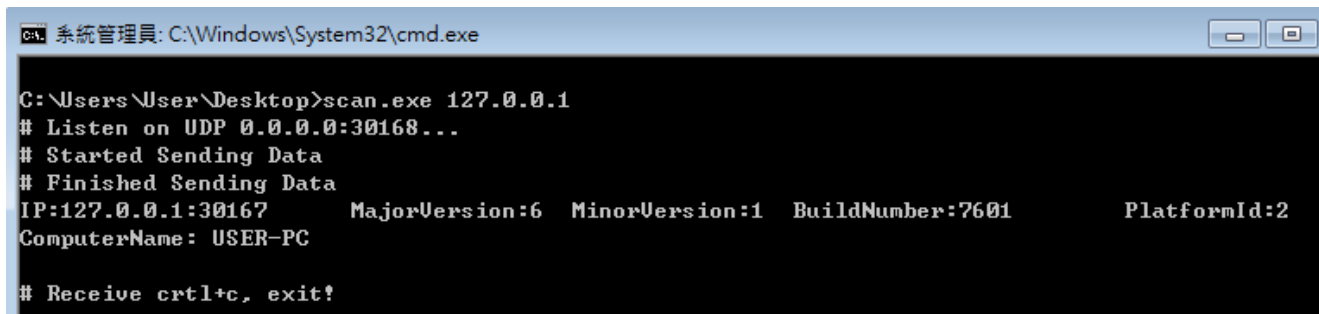
使用方法

Usage: scan.exe iplist



測試

```
> scan.exe 127.0.0.1
> scan.exe 192.168.0.0/16
```



```
系統管理員: C:\Windows\System32\cmd.exe

C:\Users\User\Desktop>scan.exe 127.0.0.1
# Listen on UDP 0.0.0.0:30168...
# Started Sending Data
# Finished Sending Data
IP:127.0.0.1:30167    MajorVersion:6    MinorVersion:1    BuildNumber:7601    PlatformId:2
ComputerName: USER-PC

# Receive ctrl+c, exit!
```

掃描程式工具下載點

[原始碼壓縮檔連結](#)

IoC

file name	md5	compiler-stamp (UTC)	type
setupSrv.exe	c5e7e7007b7ea6a3361fea7d4b029859	2009-12-04 13:35:59 UTC	EXE
cft3B18.tmp	fe763c2d71419352141c77c310e600d2	2001-10-11 12:01:58 UTC	DLL

TeamT5 杜浦數位安全團隊長期研究與解析駭客族群入侵行為，具備第一線國內外資安事件處理的豐富經驗。當資安事件發生，我們可快速完成受駭現場與歷史軌跡的綜整分析，發掘攻擊者的入侵過程與手法，提供事件處理的深入分析及建議，協助您復原與改善資安環境。

若您有資安事件處理、應變的需求，歡迎填寫諮詢表單: <https://teamt5.org/tw/contact-us/>

參考資料

[BlackBerry Cylance vs. Parite Polymorphic File Infector](#)

Share:

Related Post

