



## Unpacking Fraudulent “Fax”: Dyreza Malware from Spam

October 19, 2015 by [hasherezade](#)

Last updated: October 16, 2016

This post describes the process of unpacking a malware delivered in a spam campaign. The described sample has been delivered on 1 October 2015 at 17:33 CEST.

E-mail content:

fax: 78569889408553675571063326069.pdf



7856988940855367557...

Fragment of message headers:

Received: from spamfilter.jpenergypartners.com (84.95.205.45.forward.012.net.il [84.95.205.45])

(using TLSv1 with cipher ECDHE-RSA-AES256-SHA (256/256 bits))

(No client certificate requested)

(...)

Received: by jpep.com with SMTP

id oSMlcE7; Thu, 01 Oct 2015 11:35:28 -0400

Received: from spamfilter.jpenergypartners.com (spamfilter.jpenergypartners.com [74.7.154.89])

by spamfilter.jpenergypartners.com with ESMTP id ixqVA32rotPPduej2c;

Thu, 01 Oct 2015 11:33:28 -0400

Message-ID: <560D52C0.D8CF643D@jpep.com>

Date: Thu, 01 Oct 2015 11:33:28 -0400

From: "fax" <sstandley@jpep.com>

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:31.0) Gecko/20100101

Thunderbird/31.7.0

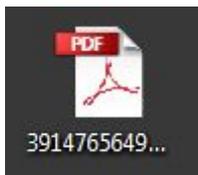
MIME-Version: 1.0

(...)

Subject:

Content-Type: multipart/mixed;

The malicious sample is shipped as a ZIP-compressed executable, pretending to be a PDF document:



After several stages of unpacking, we receive the payload belonging to the [Dyreza](#) malware family, a credential stealer.

## Elements involved

There are 4 different executables involved in a chain of execution, leading to deployment of core malicious functions:

- [841ca39312bdbb4a0010807a773961d6](#) – **391476564923647.exe** – the main file, which we receive directly after unpacking the original ZIP
- [cc5d0acba5c7e0d62dd547641d9da1a1](#) – **zatribmet.exe** – an intermediate executable, which is dropped into the %TEMP% folder

- 2 payloads, which are loaded by the RunPE technique, substituting in memory the 2 above samples. Note that the payloads' hashes (below) may vary, depending on the chosen method of dumping:
  - [9a4a171db069af2b15d6f88759b08db0](#) – **Payload #1** (Upatre – obfuscated & self modifying downloader, prepares code and injects it into *svchost.exe*)
  - [ff3d706015b7b142ee0a8f0ad7ea2911](#) – **Payload #2** (Dyreza agent)

The following focuses on retrieving these payload files.

Flow:

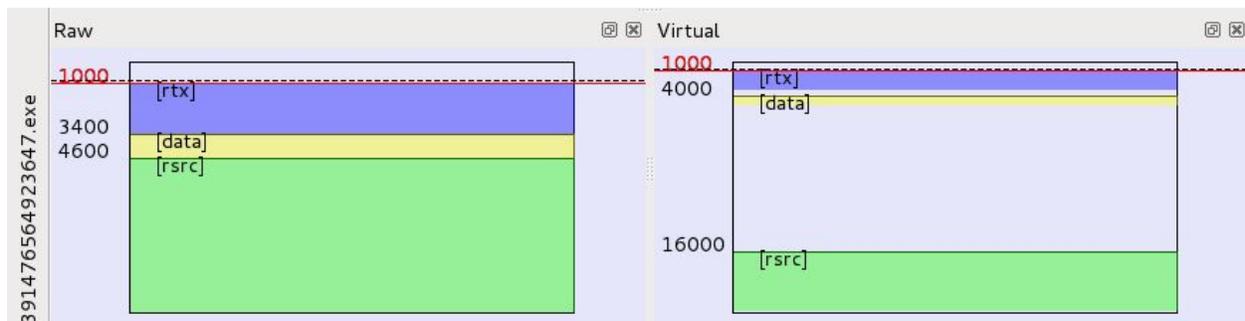
**391476564923647.exe** -> RunPE : **Payload#1** -> injects code into **svchost.exe** -> downloads: **zatribmet.exe** -> RunPE: **Payload #2**

## Unpacking

The 2 samples, **391476564923647.exe** and **zatribmet.exe**, are carriers of payloads. Both are packed by the same custom packer containing obfuscating techniques.

Due to the fact, that unpacking of both files follows analogical steps, the first file: **391476564923647.exe** will be used as an example.

Sections in the original sample:



*First section (code) renamed to **rtx**. Section **data**, that is relatively small in the raw image, gets enlarged in the virtual image, to fit the uncompressed data. Entry point is at the beginning of the first section (offset 0x1000).*

Unpacking follows several stages:

1. In **rtx** section: unpacking code into **data** section and jumping there
2. In **data** section: loading imports, copying a piece of position-independent code (that will be referred as **shellcode**) from **data** into newly allocated memory

- In **shellcode**: opening the file of running executable and reading, piece by piece, the code to be decrypted into memory. It forms a new PE file, that is, after that loaded by a RunPE technique into the memory, substituting the currently running executable.

## Stage 1:

Entry Point of **391476564923647.exe**:

```

CPU - main thread, module 39147656
00401000 $ BF DF1D2FEF MOV EDI,EF2F1DDF
00401005 > 81C7 21221111 ADD EDI,11112221
00401008 . 8BEC MOV EBP,ESP
0040100D . 8BD5 MOV EDX,EBP
0040100F . 66:81FA 02FF CMP DX,0FF02
00401014 . ^76 EF JBE SHORT 39147656.00401005
00401016 . ^76 ED JBE SHORT 39147656.00401005
00401018 . ^76 EB JBE SHORT 39147656.00401005
0040101A . ^76 E9 JBE SHORT 39147656.00401005
0040101C . 60 PUSHAD
0040101D . 68 36010000 PUSH 136
00401022 . 68 00484000 PUSH 39147656.00404800
00401027 . FF15 68404000 CALL DWORD PTR DS:[<&kernel32.GetWindow
0040102D . E8 97010000 CALL 39147656.004011C9
00401032 $ BE 80404000 MOV ESI,<&kernel32.GetTickCount>
00401037 . FF16 CALL DWORD PTR DS:[ESI]
00401039 . BE 80404000 MOV ESI,<&kernel32.GetTickCount>
0040103E . FF16 CALL DWORD PTR DS:[ESI]
00401040 . BE 80404000 MOV ESI,<&kernel32.GetTickCount>
00401045 . FF16 CALL DWORD PTR DS:[ESI]
00401047 . BE 80404000 MOV ESI,<&kernel32.GetTickCount>
0040104C . FF16 CALL DWORD PTR DS:[ESI]
0040104E . BE 80404000 MOV ESI,<&kernel32.GetTickCount>
00401053 . FF16 CALL DWORD PTR DS:[ESI]
00401055 . BE 80404000 MOV ESI,<&kernel32.GetTickCount>
0040105A . FF16 CALL DWORD PTR DS:[ESI]
0040105C . BE 80404000 MOV ESI,<&kernel32.GetTickCount>
00401061 . FF16 CALL DWORD PTR DS:[ESI]
00401063 . BE 80404000 MOV ESI,<&kernel32.GetTickCount>
00401068 . FF16 CALL DWORD PTR DS:[ESI]
0040106A . BE 80404000 MOV ESI,<&kernel32.GetTickCount>
BufSize = 136 (310.)
Buffer = 39147656.00404800
GetWindowsDirectoryA

```

First, the **wsecedit.dll** (**C:\Windows\system32\wsecedit.dll**) is opened:

```

CPU - main thread, module 39147656
00401003 . BE 80404000 MOV ESI,<&kernel32.GetTickCount>
00401008 . FF16 CALL DWORD PTR DS:[ESI]
0040100A . BE 80404000 MOV ESI,<&kernel32.GetTickCount>
0040100F . FF16 CALL DWORD PTR DS:[ESI]
004010E1 . BE 7C404000 MOV ESI,<&kernel32.GetTickCount>
004010E6 . FF16 CALL DWORD PTR DS:[ESI]
004010E8 . BE 7C404000 MOV ESI,<&kernel32.GetTickCount>
004010ED . FF16 CALL DWORD PTR DS:[ESI]
004010EF . BB DF474000 MOV EBX,39147656.004047DF
004010F4 . 6A 00 PUSH 0
004010F6 . 6A 00 PUSH 0
004010F8 . 6A 00 PUSH 0
004010FA . 68 80000000 PUSH 80
004010FF . 6A 03 PUSH 3
00401101 . 6A 00 PUSH 0
00401103 . 6A 01 PUSH 1
00401105 . 68 00000000 PUSH 00000000
0040110A . 83C3 21 ADD EBX,21
0040110D . 53 PUSH EBX
0040110E . BE 6C404000 MOV ESI,<&kernel32.CreateFileA>
00401113 . FF16 CALL DWORD PTR DS:[ESI]
0012FF40 00401115 340. CALL to CreateFileA from 39147656.00401113
0012FF44 00404000 .Ho. FileName = "C:\Windows\system32\wsecedit.dll"
0012FF48 80000000 ...C Access = GENERIC_READ
0012FF4C 00000001 0... ShareMode = FILE_SHARE_READ
0012FF50 00000000 .... pSecurity = NULL
0012FF54 00000003 <... Mode = OPEN_EXISTING
0012FF58 00000080 C... Attributes = NORMAL
0012FF5C 00000000 .... hTemplateFile = NULL

```

Its headers are read into memory:

```

00401101 . 6A 00          PUSH 0
00401103 . 6A 01          PUSH 1
00401105 . 68 00000080   PUSH 80000080
0040110A . 83C3 21       ADD EBX,21
0040110D . 53           PUSH EBX
0040110E . BE 6C404000   MOV ESI,<&kernel32.CreateFileA>
00401113 . FF16        CALL DWORD PTR DS:[ESI]
00401115 . 60          PUSHAD
00401116 . BE 7C404000   MOV ESI,<&kernel32.GetTickCount>
0040111B . FF16        CALL DWORD PTR DS:[ESI]
0040111D . BE 7C404000   MOV ESI,<&kernel32.GetTickCount>
00401122 . FF16        CALL DWORD PTR DS:[ESI]
00401124 . 61          POPAD
00401125 . 6A 00          PUSH 0
00401127 . 54          PUSH ESP
00401128 . 83EC 04       SUB ESP,4
0040112B . C70424 320100 MOV DWORD PTR SS:[ESP],132
00401132 . 83EC 04       SUB ESP,4
00401135 . 891C24       MOV DWORD PTR SS:[ESP],EBX
00401138 . 83EC 04       SUB ESP,4
0040113B . 890424       MOV DWORD PTR SS:[ESP],EAX
0040113E . FF15 50404000 CALL DWORD PTR DS:[<&kernel32._hread>]
00401144 . BE 74404000   MOV ESI,<&kernel32.GetTickCount>

0012FF48 00401144 D40. [CALL to _hread from 39147656.0040113E
0012FF4C 000000CC hFile = 000000CC (window)
0012FF50 00404800 .H0. Buffer = 39147656.00404800
0012FF54 00000132 20.. BufSize = 132 (306.)

```

First 0x132 bytes of **wsecedit.dll** :

Address	Hex dump	ASCII
00404800	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZE.♦...♦... ..
00404810	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	\$.....@.....
00404820	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00404830	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00404840	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	AV  A.+.!\$0L=!Th
00404850	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program cannot
00404860	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	be run in DOS
00404870	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode...\$.....
00404880	99 B0 FE 0C 0D 01 90 5F 0D 01 90 5F 0D 01 90 5F	000. _T0E _T0E _T0E
00404890	04 A9 13 5F CE D1 90 5F 04 A9 05 5F 08 D1 90 5F	d'e  _f0E_d'e  _e0E
004048A0	04 A9 03 5F C2 D1 90 5F 0D 01 91 5F 47 D3 90 5F	d'e  _T0E_T0L_G0E
004048B0	04 A9 02 5F DC D1 90 5F 04 A9 14 5F 89 D1 90 5F	d'e  _0E_d'e  _e0E
004048C0	04 A9 04 5F DC D1 90 5F 04 A9 01 5F DC D1 90 5F	d'e  _0E_d'e  _0E
004048D0	52 69 63 68 0D 01 90 5F 00 00 00 00 00 00 00 00	RichT0E.....
004048E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004048F0	50 45 00 00 4C 01 04 00 54 0B 5B 4A 00 00 00 00	PE..L0♦.T  J....
00404900	00 00 00 00 E0 00 02 21 0B 01 09 00 00 30 06 00	...0.0!00..0♦.
00404910	00 BE 0D 00 00 00 00 00 67 81 05 00 00 10 00 00	.z.....gu♦.♦.
00404920	00 40 06 00 00 00 C1 1F 00 10 00 00 00 02 00 00	@♦.....+♦.♦.♦.
00404930	06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	♦.....
00404940	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00404950	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00404960	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00404970	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00404980	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00404990	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004049A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004049B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004049C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004049D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004049E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
004049F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00404A00	82 42 82 42 E3 70 62 7D AE 42 82 42 BC 17 03 42	eB0B0)B)♦:B0B0 ♦0B
00404A10	82 A4 53 7D BC 42 82 42 82 58 1B 0E B2 A4 63 61	e0S)♦ B0B0♦+000Aca
00404A20	C0 01 C0 24 B8 24 80 F0 31 EF 6D F8 9B 01 4F 4E	!0 ♦\$S0C-1'm°T00N
00404A30	82 42 82 58 E3 52 85 24 92 45 50 58 88 5F 83 24	eB0XNR0\$ [EPX♦_a\$
00404A40	82 3C FE A4 A0 48 64 5E 89 5C 3D FB B2 09 C0 9F	e<#A0Hd^e^\=000. ♦0

After that, its CodeSize is retrieved and compared to some hard-coded values. (\*This is the only role that malware authors had for this DLL – it is not used anywhere else, neither in the process of unpacking).

If it doesn't satisfy the conditions, the execution is interrupted:

```

CPU - main thread, module 39147656
00401159 . 58      POP EAX
0040115A . 03C3    ADD EAX,EBX           EAX -> beginning of PE header (wsecedit.dll)
0040115C . 6A 1E   PUSH 1E
0040115E . 5E      POP ESI
0040115F . 83EE 02  SUB ESI,2
00401162 . 8B0430  MOV EAX,DWORD PTR DS:[EAX+ESI]   EAX -> CodeSize (wsecedit.dll)
00401165 . E9 00000000 JMP 39147656.0040116A
0040116A > 3D 00000900 CMP EAX,90000           CodeSize > 0x90000
0040116F . 0F87 2F000000 JB 39147656.004011A4
00401175 . BA 00520200 MOV EDX,25200           CodeSize > 0x25200
0040117A . 3BC2    CMP EAX,EDX
0040117C . 0F8F 0D000000 JG 39147656.0040118F
00401182 . 0F8F 07000000 JG 39147656.0040118F
00401188 . CC      INT3                   interrupt execution
00401189 . 0F8E 15000000 JLE 39147656.004011A4
0040118F > E8 AF200000 CALL 39147656.00403243   unpack shellcode into 'data' section
00401194 . 58      POP EAX
00401195 . BE 106F4000 MOV ESI,39147656.00406F10

DS:[0040490C]=00063000
EAX=004048F0 (39147656.004048F0), ASCII "PE"

Address  Hex dump  ASCII
004048EC 00 00 00 00 50 45 00 00 4C 01 04 00 54 DB 5B 4A  ....PE..L0..T[U
004048FC 00 00 00 00 00 00 00 00 E0 00 02 21 0E 01 09 00  .....0..!@..
00404900 00 30 06 00 00 BE 0D 00 00 00 00 67 31 05 00  .0+.z.....gu+.
0040491C 00 10 00 00 00 40 06 00 00 00 C1 1F 00 10 00 00  |.+.~.....+v|.
0040492C 00 02 00 00 06 00 00 00 00 00 00 00 00 00 00 00  @..+.....
0040493C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0040494C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....

```

Otherwise, the programs proceed to unpack the data section:

```

CPU - main thread, module 39147656
00403243 83C4 04      ADD ESP,4
00403246 8BF3      MOV ESI,EBX           ESI = 0x404800
00403248 60      PUSHAD
00403249 B8 00EA0C54 MOV EAX,540CEA00
0040324E 2D 00AC0C54 SUB EAX,540CAC00
00403253 50      PUSH EAX
00403254 83C4 04      ADD ESP,4           EAX = 0x3E00
00403257 50      PUSH EAX           loop_top
00403258 56      PUSH ESI
00403259 03F0     ADD ESI,EAX
0040325B 4E      DEC ESI           39147656.00403600
0040325C 8A0E     MOV CL,BYTE PTR DS:[ESI]
0040325E 5E      POP ESI
0040325F 58      POP EAX
00403260 4B      DEC EBX
00403261 C0C1 02      ROL CL,2
00403264 80C1 F9      ADD CL,0F9
00403267 8AD0     MOV DL,AL
00403269 FECA     DEC DL
0040326B 80E2 01      AND DL,1
0040326E 32CA     XOR CL,DL
00403270 80F1 03      XOR CL,3
00403273 884C06 FF MOV BYTE PTR DS:[ESI+EAX-1],CL
00403277 2D 01000000 SUB EAX,1
0040327C 85C0     TEST EAX,EAX
0040327E 0F84 05000000 JE 39147656.00403289
00403280 E8 CBFFFFFF CALL 39147656.00403254   call loop_top
00403289 61      POPAD
0040328A ^E9 05DFFFFFF JMP 39147656.00401194
0040328F 81C6 F4DAFFFF ADD ESI,-250C           ESI = 00406F10
00403295 FFD6     CALL ESI           call unpacked code, RVA = 4A04 ('data' section)
00403297 0000     ADD BYTE PTR DS:[EAX],AL

```

... and the execution is redirected there.

**Stage 2:**

Inside the data section:

First, additional imports are loaded by their [checksums](#):

```

CPU - main thread, module 39147656
00404A04 8BEC MOV EBP,ESP Stage#2 EntryPoint
00404A06 81EC B0000000 SUB ESP,0B0
00404A0C E8 57060000 CALL 39147656.00405068
00404A11 8945 EC MOV DWORD PTR SS:[EBP-14],EAX
00404A14 E8 00000000 CALL 39147656.00404A19
00404A19 58 POP EAX
00404A1A 66:33C0 XOR AX,AX
00404A1D 8985 7CFFFFFF MOV DWORD PTR SS:[EBP-84],EAX
00404A23 8B08 MOV EBX,EAX
00404A25 8BF8 MOV EDI,EAX
00404A27 BE BEBADDDE MOV ESI,DEADBABE
00404A2C 64:FF35 30000000 PUSH DWORD PTR FS:[30]
00404A33 58 POP EAX
00404A34 8B40 0C MOV EAX,DWORD PTR DS:[EAX+C]
00404A37 8B40 0C MOV EAX,DWORD PTR DS:[EAX+C]
00404A3A 3958 18 CMP DWORD PTR DS:[EAX+18],EBX
00404A3D 74 04 JE SHORT 39147656.00404A43
00404A3F 8B00 MOV EAX,DWORD PTR DS:[EAX]
00404A41 ^EB F7 JMP SHORT 39147656.00404A3A
00404A43 8978 18 MOV DWORD PTR DS:[EAX+18],EDI
00404A46 8970 1C MOV DWORD PTR DS:[EAX+1C],ESI
00404A49 68 EEBAC01F PUSH 1FC0EAE
00404A4E FF75 EC PUSH DWORD PTR SS:[EBP-14]
00404A51 E8 45060000 CALL 39147656.0040509B Load import by checksum...
00404A56 8945 98 MOV DWORD PTR SS:[EBP-68],EAX
00404A59 68 5A6FDEA9 PUSH A9DE6F5A
00404A5E FF75 EC PUSH DWORD PTR SS:[EBP-14]
00404A61 E8 35060000 CALL 39147656.0040509B
00404A66 8945 9C MOV DWORD PTR SS:[EBP-64],EAX
00404A69 68 6A8E0820 PUSH 20088E6A
00404A6E FF75 EC PUSH DWORD PTR SS:[EBP-14]
00404A71 E8 25060000 CALL 39147656.0040509B
00404A76 8945 8C MOV DWORD PTR SS:[EBP-74],EAX
00404A79 68 62678DA4 PUSH A48D6762

```

Then, a part of the code from data section (0x694 bytes, beginning marked blue on below illustration) is copied into a newly allocated memory (for the convenience I called it **shellcode**). This step is necessary in order to have execution redirected outside the mapped image – further, the full image is cleared for the purpose of mapping the payload on its place (RunPE technique).

```

CPU - main thread, module 39147656
004048FB FF55 B4 CALL DWORD PTR SS:[EBP-4C]
004048FE 3B85 7CFFFFFF CMP EAX,DWORD PTR SS:[EBP-84] GetModuleHandle
00404904 74 03 JE SHORT 39147656.00404809
00404906 8B45 24 MOV EAX,DWORD PTR SS:[EBP+24]
00404909 8B45 EC MOV DWORD PTR SS:[EBP-14],EAX
0040490C 8B40 3C MOV EAX,DWORD PTR DS:[EAX+3C]
0040490F 8B45 EC ADD EAX,DWORD PTR SS:[EBP-14]
00404912 8B45 E8 MOV DWORD PTR SS:[EBP-18],EAX
00404915 8B40 50 MOV EAX,DWORD PTR DS:[EAX+50]
00404918 8B45 E4 MOV DWORD PTR SS:[EBP-1C],EAX
0040491B 6A 40 PUSH 40
0040491D 68 00100000 PUSH 1000
00404922 FF75 E4 PUSH DWORD PTR SS:[EBP-1C]
00404925 6A 00 PUSH 0
00404927 FF55 88 CALL DWORD PTR SS:[EBP-78] VirtualAlloc
0040492D 8B45 D8 MOV DWORD PTR SS:[EBP-28],EAX
0040492D B8 FFFFFFFF MOV EAX,-1
00404932 8B45 E0 MOV DWORD PTR SS:[EBP-20],EAX
00404935 E8 00000000 CALL 39147656.00404B3A
0040493D 58 POP EAX
0040493D 83C0 15 ADD EAX,15
0040493E 68 94060000 PUSH 694
00404943 FF75 D8 PUSH DWORD PTR SS:[EBP-28]
00404946 58 PUSH EAX
00404946 E8 78060000 CALL 39147656.004051C4 copy shellcode to allocated memory
00404946 FF55 D8 CALL DWORD PTR SS:[EBP-28] call shellcode
0040494F 8BC5 MOV EAX,EBP shellcode starts here:
00404951 83E8 48 SUB EAX,48
00404954 58 PUSH EAX
00404955 6A 40 PUSH 40
00404957 FF75 E4 PUSH DWORD PTR SS:[EBP-1C]
0040495D FF75 EC PUSH DWORD PTR SS:[EBP-14]
0040495D FF55 9C CALL DWORD PTR SS:[EBP-64]
00404960 6A 40 PUSH 40
00404962 68 00100000 PUSH 1000
00404967 68 04010000 PUSH 104
0040496C 6A 00 PUSH 0

```

Address	Hex dump	ASCII	
00540900	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0012FEB8 00404B4F Arg1 = 00404B4F
00540910	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0012FEBC 00540000 Arg2 = 00540000
00540920	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....	0012FEC0 00000694 Arg3 = 00000694
			0012FEC4 00000132

The execution is then redirected to that part of memory (outside the mapped PE sections).

**Stage 3:**

Inside the *shellcode*, the file of the current executable is opened from the disk; particular chunks are then read and decrypted. It uses imports loaded in the Stage 2.

```

CPU - main thread
00540000 8BC5      MOV EAX,EBP      Stage#3 Entry Point:
00540002 83E8 48    SUB EAX,48
00540005 50      PUSH EAX
00540006 6A 40    PUSH 40
00540008 FF75 E4   PUSH DWORD PTR SS:[EBP-1C]
0054000B FF75 EC   PUSH DWORD PTR SS:[EBP-14]
0054000E FF55 9C   CALL DWORD PTR SS:[EBP-64]      kernel32.VirtualProtect
00540011 6A 40    PUSH 40
00540013 68 00100000 PUSH 1000
00540018 68 04010000 PUSH 104
0054001D 6A 00    PUSH 0
0054001F FF55 88   CALL DWORD PTR SS:[EBP-78]      kernel32.VirtualAlloc
00540022 8BD8    MOV EBX,EAX
00540024 68 04010000 PUSH 104
00540029 53      PUSH EBX
0054002A FF75 EC   PUSH DWORD PTR SS:[EBP-14]
0054002D FF55 DC   CALL DWORD PTR SS:[EBP-24]      kernel32.GetModuleFileNameA
00540030 6A 00    PUSH 0
00540032 68 80000000 PUSH 80
00540037 6A 03    PUSH 3
00540039 6A 00    PUSH 0
0054003B 6A 01    PUSH 1
0054003D 68 00000000 PUSH 80000000
00540042 53      PUSH EBX
00540043 FF55 D0   CALL DWORD PTR SS:[EBP-30]      kernel32.CreateFileA
00540046 8945 A4    MOV DWORD PTR SS:[EBP-5C],EAX
00540049 68 00800000 PUSH 8000
0054004E 6A 00    PUSH 0
00540050 53      PUSH EBX
00540051 FF55 84   CALL DWORD PTR SS:[EBP-7C]      kernel32.VirtualFree
00540054 6A 40    PUSH 40
00540056 68 00100000 PUSH 1000
0054005B 68 44200000 PUSH 2044
00540060 6A 00    PUSH 0
00540062 FF55 88   CALL DWORD PTR SS:[EBP-78]
00540065 8945 D4    MOV DWORD PTR SS:[EBP-2C],EAX      kernel32.VirtualAlloc
00540068 6A 40    PUSH 40
0054006A 68 00100000 PUSH 1000
0054006F 6A 40    PUSH 40
00540071 6A 00    PUSH 0
00540073 FF55 88   CALL DWORD PTR SS:[EBP-78]      kernel32.VirtualAlloc

```

This results in the emergence of a new PE (payload) in memory:

```

CPU - main thread
00540152 E8 15030000 CALL 0054046C
00540157 68 00000000 PUSH 0000
0054015C 6A 00 PUSH 0
0054015E FF75 A8 PUSH DWORD PTR SS:[EBP-58]
00540161 FF55 84 CALL DWORD PTR SS:[EBP-7C] kernel32.VirtualFree
00540164 68 00000000 PUSH 0000
00540169 6A 00 PUSH 0
0054016B FF75 AC PUSH DWORD PTR SS:[EBP-54]
0054016E FF55 84 CALL DWORD PTR SS:[EBP-7C] kernel32.VirtualFree
00540171 68 00000000 PUSH 0000
00540176 6A 00 PUSH 0
00540178 FF75 B0 PUSH DWORD PTR SS:[EBP-50]
0054017E FF55 84 CALL DWORD PTR SS:[EBP-7C] kernel32.VirtualFree
00540181 8B45 04 MOV EAX,DWORD PTR SS:[EBP-2C]
00540184 8B45 04 MOV EAX,DWORD PTR DS:[EAX+3C]
00540187 8945 04 ADD EAX,DWORD PTR SS:[EBP-2C]
0054018A 8B40 06 MOV DWORD PTR SS:[EBP-38],EAX
0054018D 66:8945 CC MOV WORD PTR SS:[EBP-34],AX
00540191 8B5D C8 MOV EBX,DWORD PTR SS:[EBP-38]
00540194 8B5B 54 MOV EBX,DWORD PTR DS:[EBX+54]
00540197 8BC5 MOV EAX,EBP
00540199 83E8 48 SUB EAX,48
0054019C 50 PUSH EAX
0054019D 6A 40 PUSH 40
0054019F 53 PUSH EBX
005401A0 FF75 EC PUSH DWORD PTR SS:[EBP-14]
005401A6 FF55 9C CALL DWORD PTR SS:[EBP-64] kernel32.VirtualProtect
005401A7 FF75 EC PUSH DWORD PTR SS:[EBP-14]
005401AA FF75 D4 PUSH DWORD PTR SS:[EBP-2C]
005401B2 E8 F4030000 CALL 005405A6 copy headers from unpacked PE
005401B2 8B45 C8 MOV EAX,DWORD PTR SS:[EBP-38]

```

Address	Hex dump	ASCII
00560000	FF 31 9E FA 4D 5A 80 00 01 00 00 00 04 00 00 00	1x'MZC.0...♦...
00560010	FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00	..S.....@...
00560020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....
00560030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....
00560040	68 00 00 00 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C	h...a  a.- =t\$0L
00560050	CD 21 54 68 69 73 20 69 73 20 61 20 50 45 20 65	=!This is a PE e
00560060	78 65 63 75 74 61 62 6C 65 0D 0A 24 50 45 00 00	xecutable..\$PE..
00560070	4C 01 02 00 00 00 00 00 00 00 00 00 00 00 00 00	L0@.....
00560080	E0 00 02 01 0B 01 00 00 00 16 00 00 00 02 00 00	0.0000.....@
00560090	00 00 00 00 00 10 00 00 00 10 00 00 00 30 00 00	.....@.....
005600A0	00 00 40 00 00 10 00 00 00 02 00 00 05 00 01 00	.....@.....@.
005600B0	00 00 00 00 05 00 01 00 00 00 00 00 00 40 00 00	.....@.....@.
005600C0	00 02 00 00 00 00 00 00 02 00 00 00 00 00 10 00	.....@.....@.
005600D0	00 10 00 00 00 10 00 00 10 00 00 10 00 00 00 00	.....@.....@.
005600E0	10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....@.
005600F0	00 00 00 00 00 30 00 00 E0 01 00 00 00 00 00 00	.....0.0@.....
00560100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....@.
00560110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....@.
00560120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....@.
00560130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....@.
00560140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....@.
00560150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....@.
00560160	00 00 00 00 2E 74 65 78 74 00 00 00 27 14 00 00	.....text...'¶.
00560170	00 10 00 00 00 16 00 00 00 02 00 00 00 00 00 00	.....@.....@.
00560180	00 00 00 00 00 00 00 00 20 00 00 E0 2E 72 73 72	.....@.....0.rsr
00560190	63 00 00 00 E0 01 00 00 00 30 00 00 02 00 00 00	c...00...0...@.
005601A0	00 18 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....@.....@.

Then, this new payload is mapped onto memory in place of the current executable. After finishing the substitution operation, *shellcode* jumps to the first section (offset 0x1000), which is now filled with a completely new code:

```

CPU - main thread
004502EF 6A 00 PUSH 0
004502F1 FF75 04 PUSH DWORD PTR SS:[EBP-2C]
004502F4 FF55 84 CALL DWORD PTR SS:[EBP-7C]
004502F7 8BE5 MOV ESP,EBP
004502F9 61 POPAD
004502FA 68 00800000 PUSH 8000
004502FF 6A 00 PUSH 0
00450301 FF7424 C0 PUSH DWORD PTR SS:[ESP-40]
00450305 6A 00 PUSH 0
00450307 50 PUSH EAX
00450308 8B8424 70FFFFFF MOV EAX,DWORD PTR SS:[ESP-90]
0045030F 05 00100000 ADD EAX,1000
00450314 894424 04 MOV DWORD PTR SS:[ESP+4],EAX
00450318 58 POP EAX
00450319 83EC 08 SUB ESP,8
0045031C FF6424 08 JMP DWORD PTR SS:[ESP+8] 39147656.<ModuleEntryPoint>
Stack SS:[0012FF84]=00401000 (39147656.<ModuleEntryPoint>)

```

Address	Hex dump	Disassembly	Comment
00401000	81EC 140C0000	SUB ESP,0C14	
00401006	8BEC	MOV EBP,ESP	
00401008	5A	POP EDX	
00401009	58	POP EAX	
0040100A	59	POP ECX	
0040100B	E8 24000000	CALL 39147656.00401034	
00401010	D7	XLAT BYTE PTR DS:[EBX+AL]	
00401011	0E	PUSH CS	
00401012	5E	POP EAX	

## Payloads

Payloads are not intended to be a topic of this post, but let's take just a brief look.

Payload#1 is an obfuscated downloader – Upatre. It communicates with C&C over SSL and downloads a second malware (**zatribmet.exe**). After that it deletes itself.

Payload #2, which comes from **zatribmet.exe**, is a persistent Dyreza agent. It starts from its anti-VM technique by checking the number of processors of the machine it is executed on:

```

00294D20 55 PUSH EBP
00294D21 8BEC MOV EBP,ESP
00294D23 83EC 2C SUB ESP,2C
00294D26 56 PUSH ESI
00294D27 50 PUSH EAX
00294D28 64:A1 30000000 MOV EAX,DWORD PTR FS:[30]
00294D2E 8945 FC MOV DWORD PTR SS:[EBP-4],EAX
00294D31 58 POP EAX
00294D32 8B45 FC MOV EAX,DWORD PTR SS:[EBP-4]
00294D35 8378 64 02 CMP DWORD PTR DS:[EAX+64],2
00294D39 0F82 81000000 JB z_payloa.00294DC0
00294D3F E8 0CF4FFFF CALL z_payloa.00294150
00294D44 8BF0 MOV ESI,EAX

```

When the above condition is satisfied, it then copy itself into 2 locations: **%APPDATA%** and **C:\Windows** (under pseudo-random names, following the pattern **[a-zA-Z]{15}.exe** (e.g. **HdlQpVwEebXreDK.exe**). After this, it creates a task in the **scheduler** for persistence and communicates with a C&C server, which is randomly picked from a predefined pool, over SSL.

See also: [A Technical Look At Dyreza](#)