

# Smoke Loader – downloader with a smokescreen still alive

---

[blog.malwarebytes.com/threat-analysis/2016/08/smoke-loader-downloader-with-a-smokescreen-still-alive/](http://blog.malwarebytes.com/threat-analysis/2016/08/smoke-loader-downloader-with-a-smokescreen-still-alive/)

Malwarebytes Labs

August 5, 2016

This time we will have a look at another payload from recent RIG EK campaign. It is Smoke Loader (Dofail), a bot created several years ago – one of its early versions was advertised on the black market in 2011. Although there were some periods of time in which it was not seen for quite a while, it doesn't seem to plan retirement. The currently captured sample appears to be updated in 2015.

This small application is used to download other malware. What makes the bot interesting are various tricks that it uses for deception and self protection.

We will walk through the used techniques and compare the current sample with the older one (from 2014).

## Analyzed samples

---

Main focus of this analysis is the below sample, which is dropped by Rig EK:

The above sample downloads:

Payload:

f60ba6b9d5285b834d844450b4db11fd – (it is an IRC bot, C&C: med-global-fox[DOT]com)

Updated Smoke Loader:

During the analysis it will be compared against the old sample, first seen in September 2014

## Behavioral analysis

---

After being deployed, Smoke Loader injects itself into *explorer.exe* and deletes the original executable. We can see it making new connections from inside the *explorer* process.

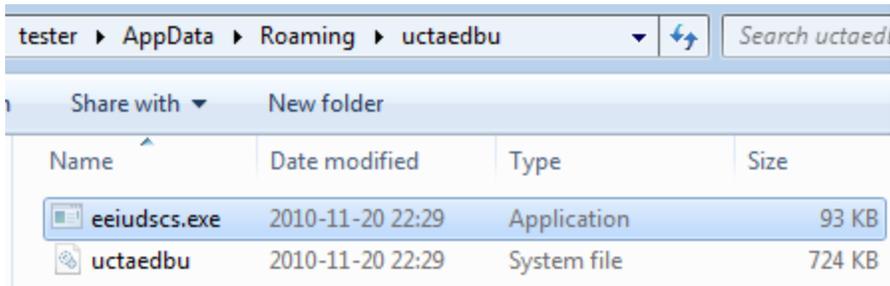
## Installation and updates

---

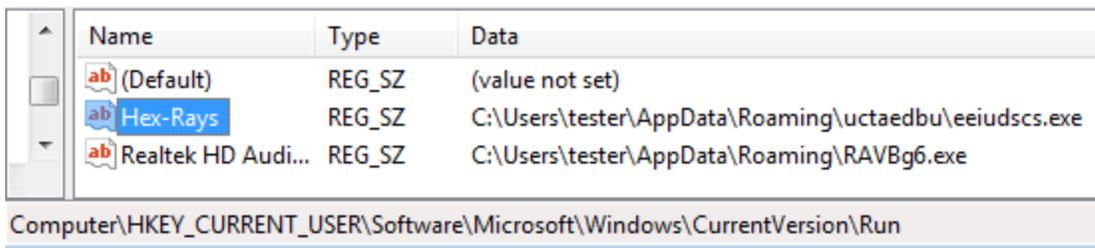
Smoke Loader not only installs its original sample but also replaces it with a fresh version, which is downloaded from the C&C – path: ***http://<CnC address>/system32.exe***. This trick makes detection more difficult – updated samples are repacked by a different crypter, may also have their set of C&Cs changed.

During the current analysis, the initial sample of Smoke Loader dropped the following one: bc305b3260557f2be7f92cbbf9f82975

Sample is saved in a hidden subfolder, located in %APPDATA%:



Smoke Loaded adds its current sample and all other downloaded executables to the Windows registry. Names of the keys are randomly chosen among the names of existing entries:



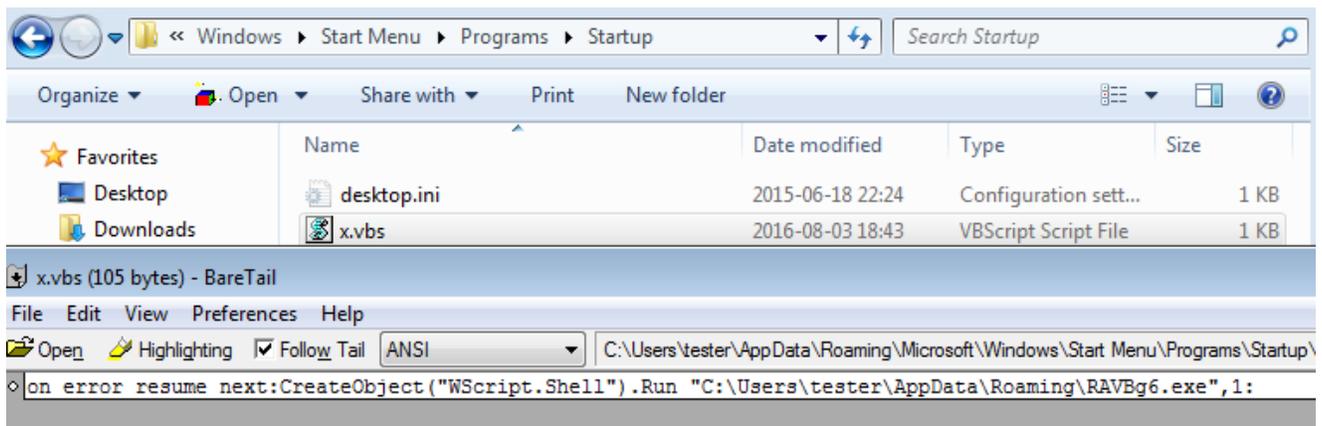
This persistence method is pretty simple (comparing i.e. with Kovter), however there are some countermeasures taken against detection of the main module. The timestamp of the dropped executable is changed, so that malware cannot be found by searching recently modified files. Access to the file is blocked – performing reading or writing operations on it is not possible.

## Loading other executables

During its presence in the system it keeps downloading additional modules – “plugins”. First, the downloaded module is saved in %TEMP% under a random name and run. Then, it is moved to %APPDATA%. Below, we can see that the payload established connection with its own separate C&C:

Process	PID	Protocol	Remote Address	Remote ...	State	Sent Packets	Sent Bytes	Rcvd Packets	Rcvd Bytes
5684.tmp.exe	3092	TCP	193.169.252.82	444	ESTABLISHED	9	269	70	11 533
Properties for 5684.tmp.exe: 3092									
Version: n/a									
Path: C:\Users\tester\AppData\Roaming\RAVBg6.exe									
End Process									
OK									
					CLOSE_WAIT			1	37
					CLOSE_WAIT	4	637	16	16 465
					LISTENING				
					LISTENING				
					LISTENING				
					LISTENING				
					LISTENING				
					LISTENING				
					LISTENING				
					LISTENING				
					LISTENING			30	660
					LISTENING				
					LISTENING				
					LISTENING				

There is also a script in Autostart for deploying the payload:



## Network communication

To make analysis of the traffic harder, along with communicating with the C&C bot generates a lot of redundant traffic, sending requests to legitimate domains.

The current sample's C&C addresses:

- [smoktruefalse.com](http://smoktruefalse.com)
- [prince-of-persia24.ru](http://prince-of-persia24.ru)

Traffic is partially encrypted.

In the examples below, we can see how the bot downloads from the C&C other executables.

1 – Updating the main bot with a new sample of Smoke Loader:



## Payload traffic

Smoke Loader deploys the downloaded sample, so after some time we can see traffic generated by the payload (connecting to *med-global-fox.com*). By its characteristics, we can conclude that this time the “plugin” is an IRC bot:

```
Stream Content
PING :med-global-fox.com
:irc.TestIRC.net NOTICE AUTH :*** Looking up your hostname...
:irc.TestIRC.net NOTICE AUTH :*** Found your hostname
CAP LS
USER 8603 0 * :[PL|x32|7|55918]
NICK [PL|x32|7|55918]
CAP REQ :multi-prefix
CAP END
JOIN #mybots
:irc.TestIRC.net 451 PING :You have not registered
:irc.TestIRC.net 451 CAP :You have not registered
:irc.TestIRC.net 001 [PL|x32|7|55918] :Welcome to the TestIRC IRC Network [PL|x32|7|55918]!
8603@user-46-112-71-214.play-internet.pl
:irc.TestIRC.net 002 [PL|x32|7|55918] :Your host is irc.TestIRC.net, running version Unreal3.2.8.1
:irc.TestIRC.net 003 [PL|x32|7|55918] :This server was created Mon Mar 14 2016 at 12:39:54 EDT
:irc.TestIRC.net 004 [PL|x32|7|55918] irc.TestIRC.net Unreal3.2.8.1 iowghraASORTVSxNCWqBzvdHtGp
lyhopsmtikrRcaq0ALQbSeIKVfMCuzNTGj
:irc.TestIRC.net 005 [PL|x32|7|55918] UHNAMES NAMESX SAFELIST HCN MAXCHANNELS=30 CHANLIMIT=#:30
MAXLIST=b:60,e:60,I:60 NICKLEN=30 CHANNELLEN=32 TOPICLEN=307 KICKLEN=307 AWAYLEN=307 MAXTARGETS=20 :are supported
by this server
:irc.TestIRC.net 005 [PL|x32|7|55918] WALLCHOPS WATCH=128 WATCHOPTS=A SILENCE=15 MODES=12 CHANTYPES=# PREFIX=
(qaohv)~&@%+ CHANMODES=beI,kfL,lj,psmtirRcOaQKVcuzNSMTG NETWORK=TestIRC CASEMAPPING=ascii EXTBAN=~.,cqnr
ELIST=MNUCT STATUSMSG=~&@%+ :are supported by this server
:irc.TestIRC.net 005 [PL|x32|7|55918] EXCEPTS INVEX CMDS=KNOCK,MAP,DCCALLOW,USERIP :are supported by this server
:irc.TestIRC.net 251 [PL|x32|7|55918] :There are 1 users and 178 invisible on 1 servers
:irc.TestIRC.net 254 [PL|x32|7|55918] 2 :channels formed
:irc.TestIRC.net 255 [PL|x32|7|55918] :I have 179 clients and 0 servers
:irc.TestIRC.net 265 [PL|x32|7|55918] :Current Local Users: 179 Max: 1017
:irc.TestIRC.net 266 [PL|x32|7|55918] :Current Global Users: 179 Max: 1017
:irc.TestIRC.net 422 [PL|x32|7|55918] :MOTD File is missing
:[PL|x32|7|55918] MODE [PL|x32|7|55918] :+ix
:irc.TestIRC.net 421 [PL|x32|7|55918] CAP :Unknown command
:irc.TestIRC.net 421 [PL|x32|7|55918] CAP :Unknown command
:[PL|x32|7|55918]!8603@Test-64CFB810.play-internet.pl JOIN :#mybots
:irc.TestIRC.net 353 [PL|x32|7|55918] = #mybots :[PL|x32|7|55918] [IN|x64|8-1|41001] [RU|x32|XP|81950] [lb|x64|7|
60038] [BR|x32|7|86013] [US|x32|7|39150] [US|x64|8-1|44633] [US|x64|7|48445] [MX|x64|7|37511] [US|x32|7|9886] [BE|
x32|XP|31183] [ES|x32|7|79289] [BR|x32|7|56517] [IQ|x32|7|53947] [ES|x32|XP|93419] [ID|x64|8-1|42604] [co|x32|7|
88395] [TH|x32|7|92690] [th|x32|7|71484] [US|x64|7|71686] [VE|x32|XP|51849] [eg|x64|7|70176] [US|x32|7|28022] [US|
x32|7|3332]
:irc.TestIRC.net 353 [PL|x32|7|55918] = #mybots :[BR|x32|7|85737] [AR|x64|7|42720] [CO|x32|7|34308] [sa|x64|7|200]
[TH|x32|7|10828] [US|x32|7|65578] [US|x32|XP|60053] [RU|x64|7|82146] [CZ|x32|XP|27161] [BR|x64|7|21291] [US|x64|7|
```

## Inside

Like most of the malware, Smoke Loader is distributed packed by some crypter that provides the first layer of defense against detection.

After removing the crypter layer, we can see the main Smoke Loader executable. However, more unpacking needs to be done in order to reach the malicious core. For the sake of convenience, I will refer to the code section of the unpacked sample as **Stage#1**. Its execution starts in the Entry Point of the main executable and its role is to provide additional obfuscation. It also serves as a loader for the most important piece: **Stage#2** – this is a DLL, unpacked to a dynamically allocated memory and run from there.

## Stage#1

Interesting feature of this bot is that often its executables have one section only and no imports. Below you can see the visualization of sections layout (Entry Point is marked red):

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num.
▾ .text	200	3800	1000	363E	E0000020	0	0	0
>	3A00	^	463E	^	rwX			

Code at Entry Point is obfuscated and difficult to follow. It contains many redundant jumps, sometimes an address of a next jump is calculated on the fly – that’s why tools for static analysis cannot resolve them. Also, to make analysis more difficult, the code modifies itself during execution.

The initial routine decrypts selected parts of the code section using XOR with a hardcoded value:

```

0040133E
0040133E loc_40133E:
0040133E push    0CBDAC235h
00401343 mov     edx, 0CB9AD292h
00401348 xor     [esp+enc_code], edx
0040134B pop     esi
0040134C jmp     short loc_401355

```

And then it it calls it:

```

004013B6
004013B6 loc_4013B6:
004013B6 call   [esp+enc_code]
004013B9 jmp     short loc_4013EA

```

This is not the only way Smoke Loader modifies itself. In the unpacked part, we can see some more tricks. This code uses many tiny jumps followed by XOR and LODS instructions to modify and displace code after every few steps of execution. In between, junk instructions have been added to make it less readable:

0040137A	DB 4B	CHAR 'K'
0040137B	DD sample2_.00401378	
0040137F	DB 79	CHAR 'y'
00401380	XOR EAX,EDX	
00401382	JMP SHORT sample2_.0040138E	
00401384	JMP SHORT sample2_.00401380	
00401386	DB 8D	
00401387	DB 84	
00401388	DB 4B	CHAR 'K'
00401389	DD sample2_.00401386	
0040138D	DB 7A	CHAR 'z'
0040138E	STOS DWORD PTR ES:[EDI]	
0040138F	LOOPO SHORT sample2_.00401375	
00401391	POP ECX	
00401392	AND ECX,0x3	
00401395	JE SHORT sample2_.004013B3	
00401397	LODS BYTE PTR DS:[ESI]	
00401398	JMP SHORT sample2_.004013A6	
0040139A	DB 8D	
0040139B	DB 84	
0040139C	DB 4B	CHAR 'K'
0040139D	DD sample2_.0040139A	
004013A1	DB 79	CHAR 'y'
004013A2	XOR AL,DL	
004013A4	JMP SHORT sample2_.004013B0	
004013A6	JMP SHORT sample2_.004013A2	
004013A8	DB 8D	
004013A9	DB 84	
004013AA	DB 4B	CHAR 'K'
004013AB	DD sample2_.004013A8	
004013AF	DB 7A	CHAR 'z'
004013B0	STOS BYTE PTR ES:[EDI]	
004013B1	LOOPO SHORT sample2_.00401397	
004013B3	JMP SHORT sample2_.004013B6	
004013B5	DB E8	
004013B6	CALL DWORD PTR SS:[ESP]	
004013B9	JMP SHORT sample2_.004013EA	

The bot loads all the necessary imports by its own. To achieve this goal, it deploys a variant of a popular method: searching function handles in the loaded modules by calculating checksum of their names and comparing them with hardcoded values. First, a handle to the loaded module is fetched with the help of Process Environment Block (PEB)\*:

```
MOV ESI, FS:[30] ; copy to ESI handle to PEB
MOV ESI, DS:[ESI+0xC] ; struct _PEB_LDR_DATA *Ldr
MOV ESI, DS:[ESI+0x1C] ; ESI = Flink = Ldr->InLoadOrderModuleList
MOV EBP, DS:[ESI+0x8] ; EBP = Flink.DllBaseAddress
```

\* read more about it [here](#)

Below we can see the fragment of code that walks through exported functions of *ntdll.dll* searching for a handle to the function: *ZwAllocateVirtualMemory* (using it's checksum: 0x976055C), and then saving the found handle in a variable:

```

004011BC | $ | PUSHAD
004011BD | . | MOV EBP,EAX
004011BE | . | MOV EBX,EDX
004011BF | . | MOV EDI,DWORD PTR DS:[EBX+0x3C]
004011C0 | . | MOV EDI,DWORD PTR DS:[EDI+EBX+0x78]
004011C1 | . | ADD EDI,EBX
004011C2 | . | PUSH EDI
004011C3 | . | MOV ECX,DWORD PTR DS:[EDI+0x18]
004011C4 | . | MOV EDX,DWORD PTR DS:[EDI+0x20]
004011C5 | . | ADD EDX,EBX
004011C6 | . | DEC ECX
004011C7 | . | PUSH ECX
004011C8 | . | MOV ESI,DWORD PTR DS:[EDX+ECX*4]
004011C9 | . | ADD ESI,EBX
004011CA | . | MOV EAX,ESI
004011CB | . | XOR ECX,ECX
004011CC | . | XOR CH,BYTE PTR DS:[EAX]
004011CD | . | ROL ECX,0x8
004011CE | . | XOR CL,CH
004011CF | . | INC EAX
004011D0 | . | CMP BYTE PTR DS:[EAX],0x0
004011D1 | . | JNZ SHORT sample2_004011DE
004011D2 | . | CMP ECX,EBP
004011D3 | . | POP ECX
004011D4 | . | JNZ SHORT sample2_004011D3
004011D5 | . | POP EDI
004011D6 | . | MOV EAX,DWORD PTR DS:[EDI+0x24]
004011D7 | . | ADD EAX,EBX
004011D8 | . | MOVZX ECX,WORD PTR DS:[EAX+ECX*2]
004011D9 | . | MOV EAX,DWORD PTR DS:[EDI+0x1C]
004011DA | . | ADD EAX,EBX
004011DB | . | MOV EAX,DWORD PTR DS:[EAX+ECX*4]
004011DC | . | ADD EAX,EBX
004011DD | . | MOV DWORD PTR SS:[ESP+0x1C],EAX
004011DE | . | POPAD
004011DF | . | RETN

```

ntdll.77DA51A7  
ntdll.77D980D0  
ntdll.<ModuleEntryPoint>  
ntdll.77D96190  
ntdll.<ModuleEntryPoint>  
pointer to the exported name  
ntdll.<ModuleEntryPoint>  
ntdll.77DA51A0  
next character of the name  
ntdll.77DA51A7  
compare with hardcoded  
check next  
ntdll.77D96190  
ntdll.<ModuleEntryPoint>  
ntdll.<ModuleEntryPoint>  
ntdll.<ModuleEntryPoint>  
save handle in a variable

Address	Hex dump	ASCII
77DA51A1	63 73 74 6F 75 6C 00 B8 00 00 00 00 BA 00 03 FE	ostoul.S.... .##
77DA51B1	7F FF 12 C2 18 00 90 B8 01 00 00 00 BA 00 03 FE	Δ †.E\$0... .##

Thanks to this trick Smoke Loader can operate without having any import table. (The same method is utilized by *Stage#2* to fill its imports).

The stored handle is used to make an API call and allocate additional memory:

```

0040121C | . | PUSH 0x40
0040121E | . | PUSH 0x3000
00401223 | . | PUSH ECX
00401224 | . | PUSH EAX
00401225 | . | PUSH EDX
00401226 | . | PUSH -0x1
00401228 | . | CALL DWORD PTR DS:[0x4010BD]
0040122E | . | MOV EAX,[LOCAL.2]
00401231 | . | MOV ESP,EBP
00401233 | . | POP EBP
00401234 | . | RETN

```

ntdll.KiFastSystemCallRet  
ntdll.ZwAllocateVirtualMemory  
allocated page  
00170000

Stack SS:[0006FF78]=00170000  
EAX=00000000

Address	Hex dump	ASCII
00170000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00170010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

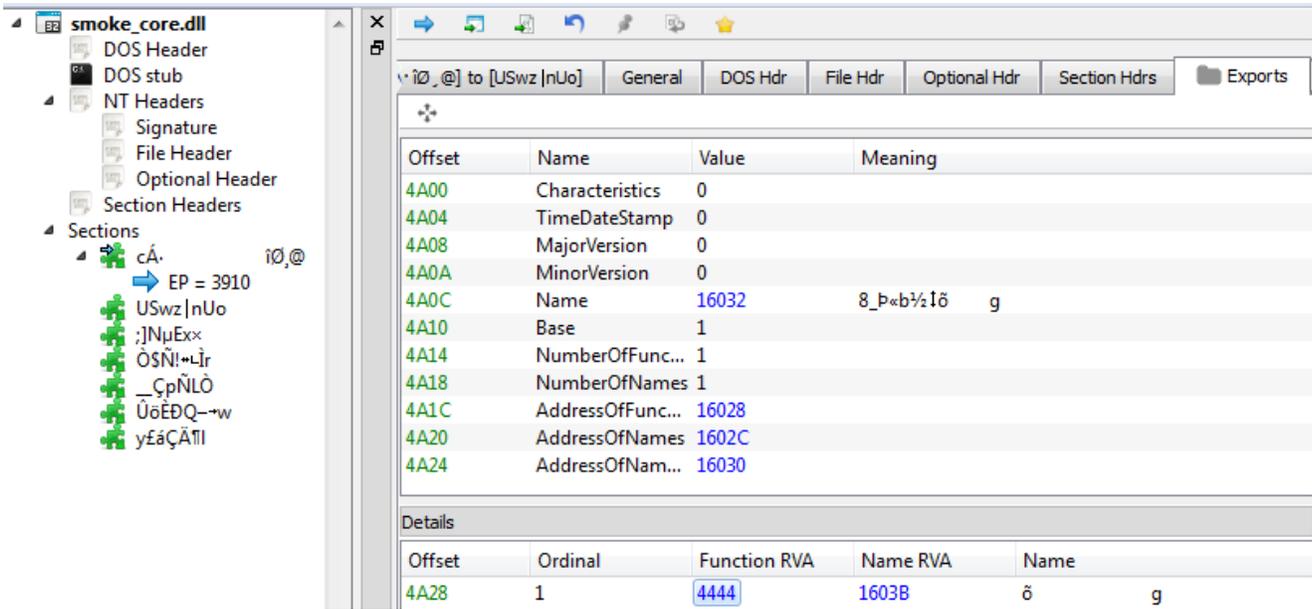
In this added memory space, *Stage#2* is being unpacked. This new module is a PE file with headers removed (it is a common anti-dumping technique). Below, you can see the part that was erased at the beginning of the file (marked red):

```

FB
_00020000.mem
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 4D 5A 50 00 02 00 00 00 04 00 0F 00 FF FF 00 00 MZP.....
00000010 B8 00 00 00 00 00 00 00 40 00 1A 00 00 00 00 00 .....@.....
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 .....
00000040 BA 10 00 0E 1F B4 09 CD 21 B8 01 4C CD 21 90 90 g.....!!,Li!..
00000050 54 68 69 73 20 70 72 6F 67 72 61 6D 20 6D 75 73 This program mus
00000060 74 20 62 65 20 72 75 6E 20 75 6E 64 65 72 20 57 t be run under W
00000070 69 6E 33 32 0D 0A 24 37 00 00 00 00 00 00 00 00 in32..$7.....
00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000100 50 45 00 00 4C 01 07 00 19 5E 42 2A 00 00 00 00 PE..L...^B*....
00000110 00 00 00 00 E0 00 8E A1 0B 01 02 19 00 36 00 00 ....f.Z^.....6..
00000120 00 1A 00 00 00 00 00 00 10 45 00 00 00 10 00 00 .....E.....
00000130 00 50 00 00 00 00 00 00 40 00 00 10 00 00 02 00 00 ..P....@.....
00000140 04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00 .....
00000150 00 90 01 00 00 04 00 00 00 00 00 00 02 00 01 00 .....
00000160 00 00 00 00 00 00 00 00 00 00 10 00 00 10 00 00 .....
00000170 00 00 00 00 10 00 00 00 00 60 01 00 40 00 00 00 .....`..@...

```

If we add the missing part, we can parse it as a typical PE file. It turns out to be a DLL exporting one function. Exactly the same technique was used before by older versions of Dofail. In the past, the name of the module was *Stub.dll* and the exported function was *Works*. Now the names are substituted by garbage.



This piece is loaded by the dedicated function inside *Stage#1*, that takes care of all the actions typically performed by the Windows Loader.

First the unpacked content is in raw format (Size of Headers: 0x400, File Alignment: 0x200):

```

D Dump - 00370000..0037FFFF
003703A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
003703B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
003703C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
003703D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
003703E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
003703F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00370400 25 64 23 25 73 23 25 73 23 25 64 2E 25 64 23 25 %d##%##%d,%d##
00370410 64 23 25 64 23 25 64 23 25 64 23 25 64 23 25 64 d##d##d##d##d...
00370420 25 64 23 25 73 23 25 73 23 25 64 2E 25 64 23 25 %d##%##%d,%d##
00370430 64 23 25 64 23 25 64 23 25 64 23 25 73 00 00 00 d##d##d##d##%s...
00370440 25 00 73 00 5C 00 25 00 73 00 00 00 25 00 73 00 %s.\%.s...%.s.
00370450 25 00 73 00 00 00 00 00 72 00 65 00 67 00 73 00 %s....r.e.g.s.
00370460 76 00 72 00 33 00 32 00 20 00 2F 00 73 00 20 00 v.r.3.2. ./s.
00370470 25 00 73 00 00 00 00 00 25 00 73 00 5C 00 25 00 %s....%.s.\%.
00370480 73 00 2E 00 6C 00 6E 00 68 00 00 00 25 00 41 00 s...l.n.k...%.A.
00370490 50 00 50 00 44 00 41 00 54 00 41 00 25 00 00 00 P.P.D.A.T.A.%...
003704A0 2E 00 54 00 45 00 4D 00 50 00 25 00 00 00 00 00 %T.E.M.P.%....
003704B0 2E 00 65 00 78 00 65 00 00 00 00 00 2E 00 64 00 ..e.x.e.....d.
003704C0 6C 00 6C 00 00 00 00 00 75 73 65 72 33 32 00 00 l.l....user32..
003704D0 73 68 65 6C 6C 33 32 00 61 64 76 61 70 69 33 32 shell32.advapi32
003704E0 00 00 00 00 75 72 6C 6D 6F 6E 00 00 6F 6C 65 33 ....urlmon..ole3
003704F0 32 00 00 00 77 69 6E 68 74 74 70 00 48 65 6C 70 2...winhttp.Help
00370500 4C 69 6E 68 00 00 00 00 55 52 4C 49 6E 66 6F 41 Link....URLInfoA
00370510 62 6F 75 74 00 00 00 00 73 62 69 65 64 6C 6C 00 bout....sbiedll.
00370520 64 62 67 68 65 6C 70 00 71 65 6D 75 00 00 00 00 dbghelp.qemu...
00370530 76 69 72 74 75 61 6C 00 76 6D 77 61 72 65 00 00 virtual.vmware..
00370540 78 65 6E 00 66 66 66 66 63 63 65 32 34 00 00 00 xen.ffffcce24...

```

Then, the same content is realigned to a virtual format (unit size: 0x1000):

```

D Dump - 00380000..0038FFFF
00380F90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00380FA0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00380FB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00380FC0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00380FD0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00380FE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00380FF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00381000 25 64 23 25 73 23 25 73 23 25 64 2E 25 64 23 25 %d##%##%d,%d##
00381010 64 23 25 64 23 25 64 23 25 64 23 25 64 23 25 64 d##d##d##d##d...
00381020 25 64 23 25 73 23 25 73 23 25 64 2E 25 64 23 25 %d##%##%d,%d##
00381030 64 23 25 64 23 25 64 23 25 64 23 25 73 00 00 00 d##d##d##d##%s...
00381040 25 00 73 00 5C 00 25 00 73 00 00 00 25 00 73 00 %s.\%.s...%.s.
00381050 25 00 73 00 00 00 00 00 72 00 65 00 67 00 73 00 %s....r.e.g.s.
00381060 76 00 72 00 33 00 32 00 20 00 2F 00 73 00 20 00 v.r.3.2. ./s.
00381070 25 00 73 00 00 00 00 00 25 00 73 00 5C 00 25 00 %s....%.s.\%.
00381080 73 00 2E 00 6C 00 6E 00 68 00 00 00 25 00 41 00 s...l.n.k...%.A.
00381090 50 00 50 00 44 00 41 00 54 00 41 00 25 00 00 00 P.P.D.A.T.A.%...
003810A0 2E 00 54 00 45 00 4D 00 50 00 25 00 00 00 00 00 %T.E.M.P.%....
003810B0 2E 00 65 00 78 00 65 00 00 00 00 00 2E 00 64 00 ..e.x.e.....d.
003810C0 6C 00 6C 00 00 00 00 00 75 73 65 72 33 32 00 00 l.l....user32..
003810D0 73 68 65 6C 6C 33 32 00 61 64 76 61 70 69 33 32 shell32.advapi32
003810E0 00 00 00 00 75 72 6C 6D 6F 6E 00 00 6F 6C 65 33 ....urlmon..ole3
003810F0 32 00 00 00 77 69 6E 68 74 74 70 00 48 65 6C 70 2...winhttp.Help
00381100 4C 69 6E 68 00 00 00 00 55 52 4C 49 6E 66 6F 41 Link....URLInfoA
00381110 62 6F 75 74 00 00 00 00 73 62 69 65 64 6C 6C 00 bout....sbiedll.
00381120 64 62 67 68 65 6C 70 00 71 65 6D 75 00 00 00 00 dbghelp.qemu...
00381130 76 69 72 74 75 61 6C 00 76 6D 77 61 72 65 00 00 virtual.vmware..
00381140 78 65 6E 00 66 66 66 66 63 63 65 32 34 00 00 00 xen.ffffcce24...

```

Another subroutine parses and applies relocations. As we can see below, it is a typical relocations table known from PE format. Entries are stored as a continuous array of WORDs:

```

00401173 . 83C6 08 ADD ESI,0x8
00401176 . 31C0 XOR EAX,EAX
00401178 > 66:AD LODS WORD PTR DS:[ESI]
0040117A . A9 00300000 TEST EAX,0x3000
0040117F . 74 0B JE SHORT sample2_0040118C
00401181 . 25 FF0F0000 AND EAX,0xFFF

```

---

```

DS:[ESI]=[00187008]=326C
Jump from 0040118C

```

Address	Hex dump	ASCII
00187008	6C 32 7B 32 81 32 87 32 8E 32 96 32 A1 32 BB 32	2(2u2c2A2l2i2n 2
00187018	C5 32 21 33 3D 33 50 33 85 33 96 33 C8 33 0C 34	+2*3=3P383l3*3.4
00187028	12 34 23 34 29 34 3F 34 71 34 96 34 B8 34 C7 34	*4#4)4?4q4l45454
00187038	ED 34 24 35 50 35 6E 35 98 35 BA 35 E2 35 FF 35	Y4\$5P5n5\$5  505 5
00187048	43 36 50 36 5A 36 64 36 6B 36 09 37 21 37 26 37	C6P626d6k6.7?7&7
00187058	2F 37 3A 37 53 37 62 37 75 37 7C 37 A1 37 AD 37	/7:7S7b7u7l7i7s7
00187068	D4 37 F4 37 FB 37 0B 38 3A 38 7B 38 9F 38 B0 38	d7~7u78:8(888888
00187078	09 38 F3 38 FC 38 14 38 1A 38 34 38 3B 38 4E 38	'8'8R:9l;+;4;:;N;
00187088	58 38 A4 38 AF 38 06 3C 11 3C 27 3C 3B 3C 4A 3C	X;A;";*<<'<';<J<

The loader processes them one by one. First, it checks if the entry type is “32-bit field” (by **TEST EAX,0x3000**) – it is the only format supported in this case. Then, it fetches the relocation offset (**AND EAX,0xFFF**), gets the pointed address and performs calculation – by removing old ImageBase (it’s value is hardcoded) and applying the new base – offset to the dynamically allocated memory where the unpacked code was copied).

Finally execution flow can be redirected to the new code. *Stage#1* calls the exported function from the *Stage#2* DLL with three parameters. The first one is a string, different for each sample (this time it is “00018”):

```

004011B0 | .REF 3105 BYTE PTR EB1ED1J
004011B2 | .PUSH 0x0
004011B4 | .PUSH EBX
004011B5 | .PUSH sample2_.00404639          ASCII "00018"
004011B8 | .CALL EDX
004011BC | .ADD BYTE PTR DS:[EAX],AL
004011BE | .ADD BYTE PTR DS:[EAX],AL

```

EDX=00174444

The execution of *Stage#2* starts inside the dynamically allocated section:

```

00174444 | PUSH EBP
00174445 | MOV EBX,ESP
00174447 | PUSH EBX
00174448 | PUSH ESI
00174449 | PUSH EDI
0017444A | MOV EDI,DWORD PTR SS:[EBP+0xC]
0017444D | MOV ESI,0x176048
00174452 | CALL 00172D00
00174457 | PUSH 0xC14
0017445C | PUSH ESI
0017445D | CALL DWORD PTR DS:[0x179DA0]
00174463 | MOV AX,65
00174466 | TEST AX,AX
00174469 | JE SHORT 00174471
0017446B | INC BYTE PTR DS:[0x176398]
00174471 | LEA EAX,DWORD PTR DS:[ESI+0x23C]
00174477 | PUSH EAX
00174478 | CALL DWORD PTR DS:[0x179DB0]
0017447E | MOV BYTE PTR DS:[ESI+0x351],0x0
00174485 | CMP DWORD PTR DS:[ESI+0x240],0x6
0017448C | JB SHORT 001744A1
0017448E | CALL 0017436C
00174493 | CMP EAX,0x2000

```

dynamically allocated page  
sample2\_.004011A6

At this stage we can see some of the strings known from previous editions of Smoke Loader. String “2015” may suggest that this version has been written in 2015 (however, compilation timestamp of the sample is more recent: 10-th June 2016).

```

00173785 | MOV EAX,0x175048
0017378A | CALL 00171954
0017378F | MOV ESI,EAX
00173791 | MOV EDX,0x173C68          ASCII "2015"
00173796 | MOV ECX,0x1
0017379B | MOV EAX,EBX
0017379D | CALL 00171B88
001737A2 | MOV EDI,EAX
001737A4 | PUSH ESI
001737A5 | CALL DWORD PTR DS:[0x179C8C]
001737AB | PUSH EAX
001737AC | PUSH ESI
001737AD | PUSH 0x0
001737AF | CALL DWORD PTR DS:[0x179D98]
001737B5 | MOV EDX,0x175048
001737BA | XOR EAX,DWORD PTR DS:[EDX]
001737BC | INC EAX
001737BD | CMP EDI,EAX
001737BF | JNZ 00173C40
001737C5 | ADD EBX,0x4
001737C8 | MOV BYTE PTR SS:[EBP-0x71],0x0
001737CC | MOV EDX,0x173C70          ASCII "plugin_size"
001737D1 | MOV ECX,0x1

```

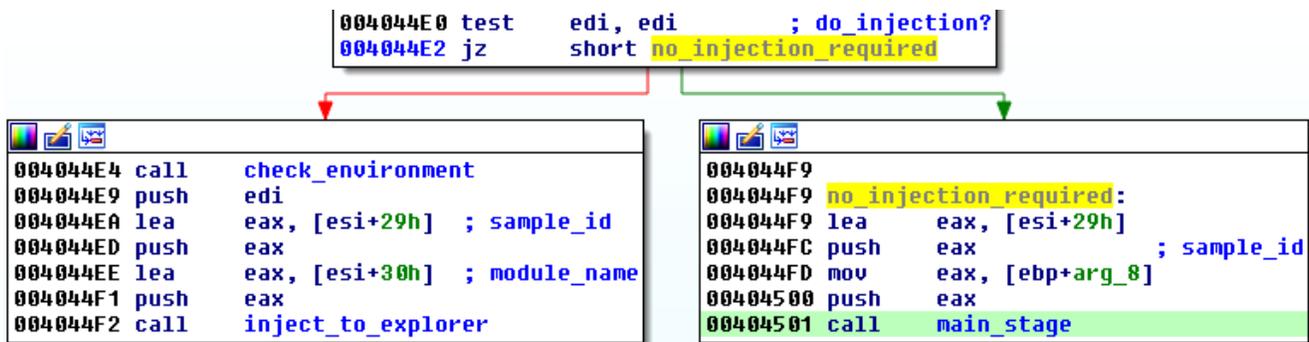
## Stage#2

While the previous stage was just a preparation, at *Stage#2* the malicious functions are deployed. Its entry lies within the exported function that has the following header:

```
int __stdcall Work(char* sample_id, bool do_injection, char* file_path);
```

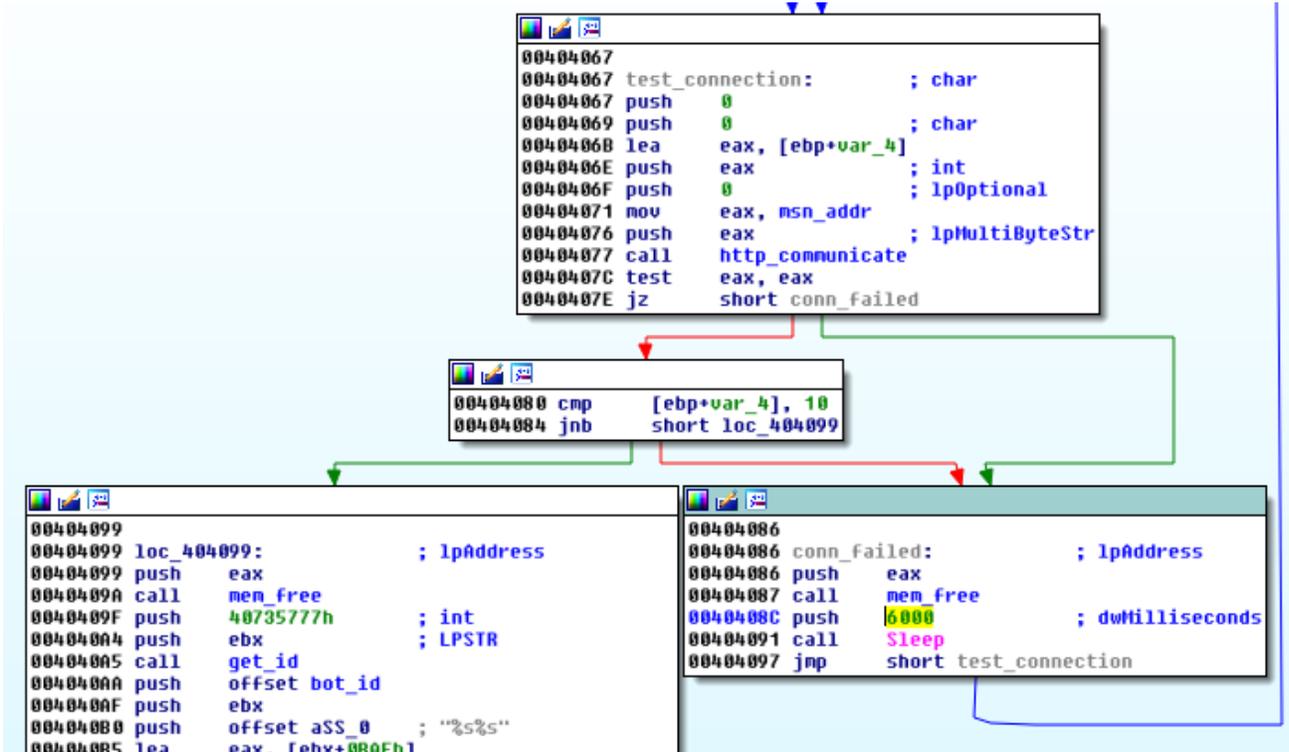
Basing on those parameters, the executable recognize its current state and the execution path to follow.

Before executing the real mission, the bot prepares a disguise – injecting its code into a legitimate process – *explorer.exe* (more about it will be explained later). Whether this path should be deployed or not, it is specified by the second parameter (denoted as *do\_injection*).

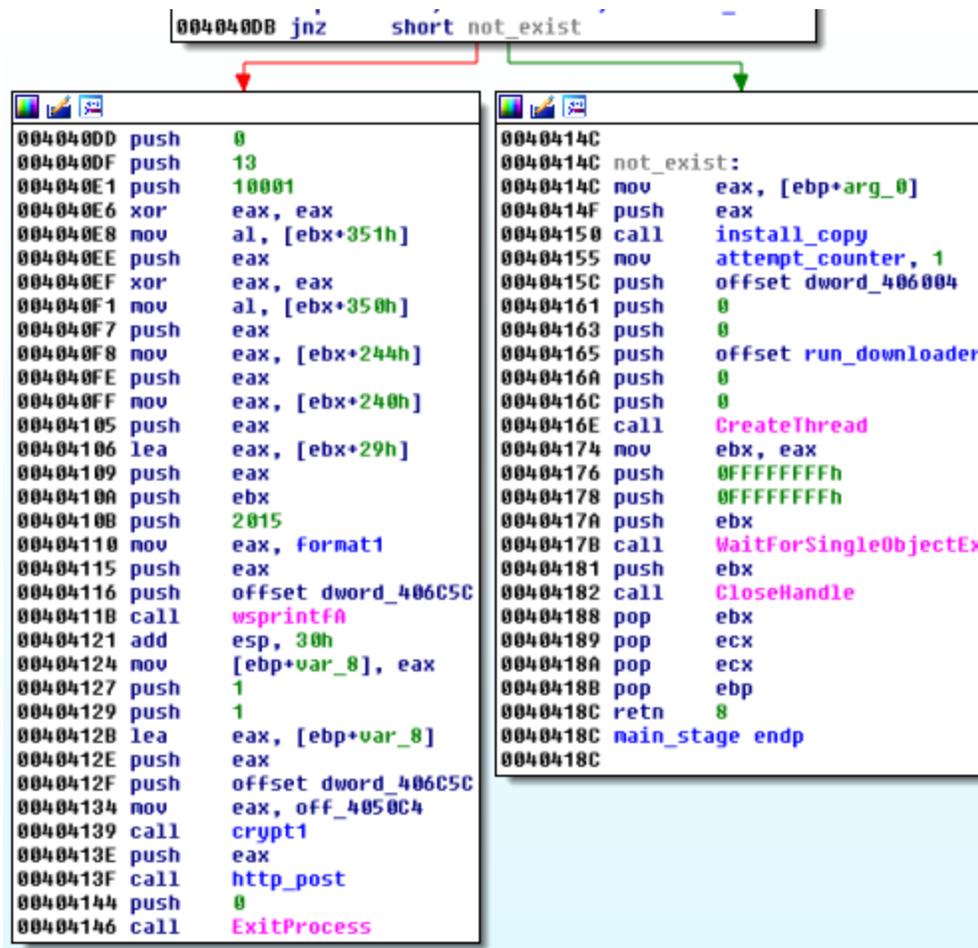


- If *Stage#2* was called with ***do\_injection* flag set**, it will inject the code into ***explorer.exe***. Before doing so, the environment is checked for the presence of tools used for malware analysis. If any symptom is detected pointing that the sample is running in the controlled environment, application goes in the infinite sleep loop.
- If *Stage#2* was called with ***do\_injection* flag cleared**, it starts proceeding to the *main path* of execution, that includes connecting to the C&C and downloading malicious modules.

If the main path of execution has been chosen, the bot proceeds to communicate with its C&C server. It is a known fact that before making the connection to the real C&C it first checks if the network is reachable. For the purpose of testing, it uses some non-malicious address – in this case it is ***msn.com***. As long as it gets no response, it keeps waiting and re-trying:



Once it found the connection working, next it verifies whether or not the application is already running (using the mutex with a name unique for the particular machine).



- If the mutex exist, program sends report to the C&C server and exits
- If the mutex does not exist (program is not yet running), it installs itself and then starts the main operations.

## Injections to other processes

The older version was injecting the code alternatively to *explorer.exe* or *svchost.exe*. Injection to *explorer.exe* employed an interesting trick that triggered a lot of attention from researchers. It is based on a PowerLoader injection technique (*Shell\_TrayWnd / NtQueueApcThread*).

Injection to *svchost.exe* was just a fail-safe, and followed more classic way similar to this one. Functions used:

```
CreateProcessInternalA
NtCreateSection
NtMapViewOfSection
RtlMoveMemory
NtUnmapViewOfSection
NtQueueApcThread
ResumeThread
```

The current version dropped that idea in favor for another method (similar to this one) – adding a new section to the remote process and copying its own code there. Functions used:

```
CreateProcessInternalA
NtQueryInformationProcess
ReadProcessMemory
NtCreateSection
NtMapViewOfSection
RtlMoveMemory
NtUnmapViewOfSection
ResumeThread
```

Now the only target of the injection is *explorer.exe*.

It patches Entry Point of *explorer* and adds there a code redirecting to the newly added section. That section contains the injected *Stage#2* DLL along with a small loader (similar to the one from *Stage#1*). Again, the loader prepares *Stage#2* and deploys it – this time with different parameters:

```
004010E5 . . MOV ECX,DWORD PTR DS:[EDI*10+4]
004010E8 . . REP STOS BYTE PTR ES:[EDI]
004010ED . . LEA EDI,DWORD PTR DS:[EBX+0x1218]
004010F3 . . MOV ECX,DWORD PTR DS:[EDI+0x150]
004010F9 . . REP STOS BYTE PTR ES:[EDI]
004010FB . . POPAD
004010FC . . LEA EDI,DWORD PTR DS:[EBX+0xFFB] sample_path
00401102 . . PUSH EDI
00401103 . . PUSH 0x0
00401105 . . LEA EDI,DWORD PTR DS:[EBX+0x120F] ASCII "00018"
00401108 . . PUSH EDI
0040110C . . CALL EDX call Stage#2
0040110E > . . PUSH 0x64
00401110 . . CALL DWORD PTR DS:[EBX+0x18]
00401113 . . ^ JMP SHORT patched1.0040110E
00401115 . . RETN
00401116 . . MOV EAX,EAX
```

## Communication protocol

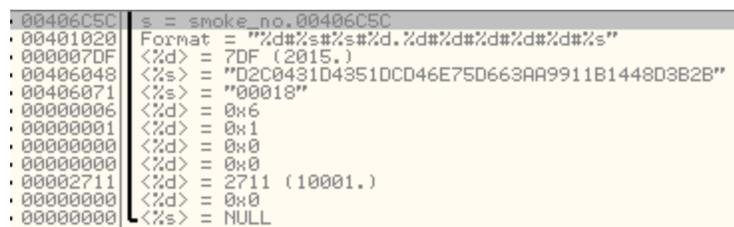
---

Old versions of Smoke Loader were using a very descriptive protocol, with commands directly pointing to the functionality. Below are the parameters used by the old version:

```
cmd=getload&login=  
&file=  
&run=ok  
&run=fail  
&sel=  
&ver=  
&bits=  
&doubles=1  
&personal=ok  
&removed=ok  
&admin=  
&hash=
```

In the current version, the sent beacon looks different – parameters are separated by a delimiter instead of following the typical, more lengthy key-value format:

```
"2015#D2C0431D4351DCD46E75D663AA9911B1448D3B2B#00018#6.1#0#0#10001#0#"
```

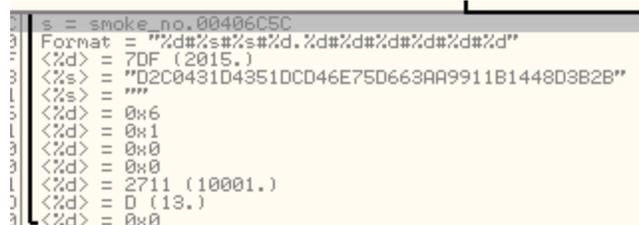


```
.00406C5C | s = smoke_no.00406C5C  
.00401020 | Format = "%d#%s#%s#%d.%d#%d#%d#%d#%d#%s"  
.000007DF | <%d> = 7DF (2015.)  
.00406048 | <%s> = "D2C0431D4351DCD46E75D663AA9911B1448D3B2B"  
.00406071 | <%s> = "00018"  
.00000006 | <%d> = 0x6  
.00000001 | <%d> = 0x1  
.00000000 | <%d> = 0x0  
.00000000 | <%d> = 0x0  
.00000000 | <%d> = 0x0  
.00002711 | <%d> = 2711 (10001.)  
.00000000 | <%d> = 0x0  
.00000000 | <%s> = NULL
```

Reading the beacon, we can confirm that the currently analyzed version is higher than the previous one. The bot also sends its ID, which is generated based on the GUID of particular system and the parameter typical for the particular sample (i.e. “00018”).

The program also reports to the C&C if there was attempt to run it more than once (mutex locked):

```
"2015#D2C0431D4351DCD46E75D663AA9911B1448D3B2B#00018#6.1#0#0#10001#13#0"
```



```
s = smoke_no.00406C5C  
3 | Format = "%d#%s#%s#%d.%d#%d#%d#%d#%d#%d"  
1 | <%d> = 7DF (2015.)  
3 | <%s> = "D2C0431D4351DCD46E75D663AA9911B1448D3B2B"  
L | <%s> = ""  
5 | <%d> = 0x6  
L | <%d> = 0x1  
3 | <%d> = 0x0  
3 | <%d> = 0x0  
L | <%d> = 2711 (10001.)  
3 | <%d> = D (13.)  
3 | <%d> = 0x0
```

## Conclusion

---

In the past Smoke Loader was extensively distributed via spam. Now we encountered it carried by an exploit kit.

Many parts of the bot didn't changed over the years, making this malware easy to identify. It still uses the same set of environment checks for its defense. Also, it waits for network accessibility in old style. The protocol used for its communication with the C&C is now less descriptive – it doesn't have so many keywords that identifies its performed actions. Like the previous, traffic is encrypted. The core features also stayed the same and the main role of this malware is to download and deploy other modules.

## Appendix

---

<http://stopmalvertising.com/rootkits/analysis-of-smoke-loader.html>

<https://blog.fortinet.com/2014/11/12/the-rebirth-of-dofail>

---

*This was a guest post written by Hasherezade, an independent researcher and programmer with a strong interest in InfoSec. She loves going in details about malware and sharing threat information with the community. Check her out on Twitter [@hasherezade](#) and her personal blog: <https://hshzrd.wordpress.com>.*