S2 GRUPO

INNOVATION IN PROCESSES

MALWARE REPORT

Evolution of Trickbot

REPORT

06/2017

INDEX

1. INTRODUCTION

This document contains a review of the latest versions of a Trojan family known as **"Trickbot/TrickLoader".** It is a bank-type Trojan that steals credentials and bank details from infected users. Although its main objective and behavior is focused on online banking users, being a modular Trojan, it has capabilities that attackers could use for other purposes, such as document exfiltration.

You can find a lot of documentation regarding the logic and origins of this malware; part of this report is based on information from some of them in order to contrast it with the logic of the latest versions and to be able to observe its evolution and new functionalities. All sources for which information has been obtained can be found in the references section.

It should be noted that the report starts with and relies mainly on the analyses carried out by **@hasherezade** and by **Xiaopeng Zhang** (Fortinet). Based on these analyses, an attempt was made to compare whether in the last versions some aspect had changed and to deepen in the mechanisms not described until the moment.

In summary, Trickbot has the following capabilities:

- It loads the code into the system
- It creates a replica of itself in the %APPDATA%
- It applies persistence techniques
- It collects sensitive information
- It injects code into other applications to control the information they handle
- It exfiltrates the information you get to your Command and Control server

During the completion of this report the S2 Grupo Malware Laboratory worked with the samples that match the following MD5 hashes:

1000005_Trickbot_Loader.exe	**a50c5c844578e563b402daf19289f71f**
1000005_Trickbot_bot32.exe	28661ea73413822c3b5b7de1bef0b246
1000010_Trickbot_Loader.exe	218613f0f1d2780f08e754be9e6f8c64
1000010_Trickbot_bot32.exe	135e4fa98e2ba7086133690dbd631785
1000014_Trickbot_Loader.exe	e054eaae756d31a4f6e30cc74b2e51dd
1000014_Trickbot_bot32.exe	719578c91b4985d1f955f6adb688314f
1000016_Trickbot_Loader.exe	132c4338cdc46a0a286abf574d68e2e0
1000016_Trickbot_bot32.exe	e8e7b0a8f274cad7bdaedd5a91b5164d

As you can see in the previous image, four different versions of **Trickbot**. Each one of them consists of its loader and its final payload for 32-bit systems; although there is also the 64-bit version of all, it was not the subject of the analysis performed.

2. INFECTION PROCESS

The main route of infection of this malware occurs through a Word document with macros that arrives attached in an email or through an exploited vulnerability by an ExploitKit.

The **infection** follows this order of execution:

- A Trickbot sample is downloaded from a compromised domain in the% APPDATA% folder and executed
- It creates a scheduled task on the system that provides persistence
- It creates two files ("client_id" and "group_tag") in the same directory, one with a unique ID of the infected host and one with the ID of the current infection campaign or version of the configuration.
- It contacts with an external IP obtaining domain, among other things to test the connectivity and send it to your command and control servers (C2 from now).
- It contacts one of its C2 servers to get malware updates, modules that perform most of the malware logic and various configuration files.
- After all this, it begins to execute or inject in different processes its modules that are responsible for collecting information of the system and browsing credentials especially of online banking.

3. TECHNICAL CHARACTERISTICS

The main executable of **Trickbot** is usually packaged with its own **“*packer*”**, which obfuscates the functionality of the executable and prevents generic signatures from being generated from the content itself, seeing that for each version the packer causes the code to vary completely. After unpacking one can see how the number of functions of the executable increases greatly, as it now reflects the functionality of the malicious program:

Packed

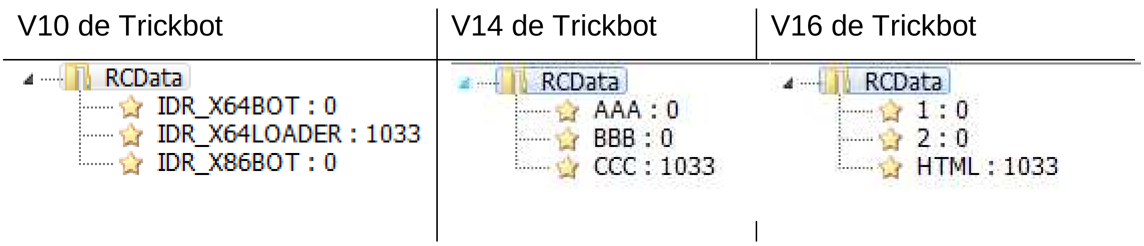
Function name	Segme
__vbaChkstk	.text
__vbaExceptHandler	.text
__vbaFPException	.text
_adj_fdiv_m32	.text
_adj_fdiv_m64	.text
DllFunctionCall	.text
ThunRTMain	.text
sub_40365C	.text
sub_4036B0	.text
sub_403710	.text
sub_403930	.text
sub_4039C8	.text
sub_403A58	.text
sub_403B10	.text
sub_407590	.text
sub_407CE0	.text
sub_407E30	.text
sub_40C770	.text
sub_40C9E0	.text
sub_40CA50	.text
sub_40CD70	.text
sub_40CD80	.text
sub_40D760	.text
sub_40D9B0	.text

Unpacked

sub_3D1000	.tex
sub_3D1050	.tex
sub_3D10A0	.tex
sub_3D10D0	.tex
sub_3D1C30	.tex
sub_3D1E40	.tex
sub_3D1EA0	.tex
sub_3D1ED0	.tex
sub_3D1F40	.tex
sub_3D1FD0	.tex
sub_3D2020	.tex
sub_3D2140	.tex
sub_3D21A0	.tex
sub_3D2240	.tex
sub_3D2260	.tex
sub_3D2300	.tex
sub_3D2310	.tex
sub_3D23B0	.tex
sub_3D2470	.tex
sub_3D25C0	.tex
sub_3D25F0	.tex
sub_3D2650	.tex
sub_3D2690	.tex
wWinMain(x,x,x,x)	**.tex**
sub_3D2E90	.tex
sub_3D2F40	.tex
sub_3D3090	.tex
sub_3D31B0	.tex
sub_3D31E0	.tex
sub_3D3310	.tex
sub_3D3340	.tex
sub_3D35B0	.tex
sub_3D3720	.tex
sub_3D3740	.tex
sub_3D3790	.tex
sub_3D3B10	.tex
sub_3D3D70	.tex
sub_3D3E00	.tex
sub_3D4050	.tex
sub_3D4260	.tex
sub_3D43B0	.tex
sub_3D4400	.tex

After the "unpack" the first stage of this malware is obtained, known as "Loader". This executable is responsible for verifying the architecture of the system and depending on whether it is a 32 or 64 bit computer, it loads the "bot" from its resources, corresponding to that architecture. The "bot" is the executable that takes care of the last stage of infection and contains all the basic malware logic.

In the first versions, the resources contained in the Loader were easily recognizable because they had descriptive names, as they identified the two versions of the Bot and a Loader to correctly load the 64-bit. In the latest versions they began to put non-descriptive names so as to make it difficult to identify them:

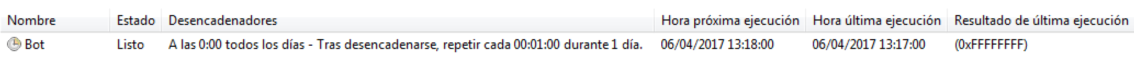


These resources, consists on executable files (PE) encrypted with the AES CBC algorithm, so after extracting them they still need to be decrypted or otherwise can be extracted from memory after running the Loader and waiting for it to perform the decryption itself and load them in RAM.

After loading the corresponding bot, it starts executing the main logic of this threat:

It first checks its location on the system, and if it is not found in% APPDATA% it copies itself to this location, starts executing its replica in that folder and ends the current process.

As a persistence technique, it uses scheduled system tasks rather than registry keys as is often the case in other samples of malware. Previous versions of **Trickbot**, in all cases created a single programmed task called "bot" and made sure that every minute was launched to keep running on the system.



Nombre	Estado	Desencadenadores	Hora próxima ejecución	Hora última ejecución	Resultado de última ejecución
Bot	Listo	A las 0:00 todos los días - Tras desencadenarse, repetir cada 00:01:00 durante 1 día.	06/04/2017 13:18:00	06/04/2017 13:17:00	(0xFFFFFFFF)

In the latest versions, if it is executed with administrator permissions in addition to the previously mentioned task, which it has called "Drives update", it creates another one that executes it when any user logs in, called "AplicationsCheckVersion"

Nombre	Estado	Desencadenadores	Hora próxima ejecución	Hora última ejecución
Appliations...	Listo	Al iniciar la sesión un usuario - El desencadenador expira a las 01/01/2020 8:00:00.		Nunca
Drivers update	Listo	A las 0:00 todos los días - Tras desencadenarse, repetir cada 00:01:00 durante 1 día.	06/04/2017 13:15:00	Nunca

Its next action is to check if it has all the configuration files with which it usually works:

Modules	17/11/2016 21:43	Carpeta de archivos	
client_id	17/11/2016 21:54	Archivo	1 KB
config.conf	11/04/2017 9:41	Archivo CONF	1 KB
group_tag	17/11/2016 21:54	Archivo	1 KB
Trickbot	17/11/2016 21:18	Aplicación	93 KB

If it does not find them, it generates them from information obtained in the system and the resources of the bot, which consist of an encrypted configuration file (CONFIG) and a key to verify the signature of the configuration and modules (KEY).

RCData
CONFIG : 0
KEY : 0

In this case there have been no changes in the names of these resources to date, although it is likely that in future iterations we will see how they eliminate these names as in the case of Loader resources.

In the first run of **Trickbot** on the computer generates a file called "client_id" that contains a token or user ID, which identifies the current host.

```
USER-PC[redacted]B4D34878BD55D02B2
```

Trickbot obtains it's configuration from a file in a disk with the name config.conf or from the resources of its own binary. This configuration will be decrypted, and after decryption it can be seen that it contains the version of the malware itself, a campaign code or version of the configuration, the addresses of several of its main C2s, and the list of modules that it must download and run automatically from any of its C2s.


```
<ver>1000014</ver>
<gtag>pre7</gtag>
<servs>
<srv>190.xxxxxxx:443</srv>
<srv>200.xxxxxxx:443</srv>
<srv>36.xxxxxxx:443</srv>
<srv>221.xxxxxxx:443</srv>
<srv>80.xxxxxxx:443</srv>
<srv>203.xxxxxxx:443</srv>
<srv>84.xxxxxxx:443</srv>
</servs>
<autorun>
<module name="systeminfo" ctl="GetSystemInfo"/>
<module name="injectDll"/>
</autorun>
</mcconf>
```


It then checks the connectivity by making a request to an external domain that reports the victim's IP esternal address, this domain comes from a list contained inside the malware and which have been increasing during the different version updates.

Version 7

```
unicode 0, <ip.anysrc.net>,0
dd offset aMyexternalip_c ; DATA XREF: sub_1D6F60+6A↑r
                          ; "myexternalip.com"
dd offset aApi_ipify_org ; "api.ipify.org"
dd offset aIcanhazip_com ; "icanhazip.com"
dd offset aBot_whatismyip ; "bot.whatismyipaddress.com"
dd offset aIp_anysrc_net ; "ip.anysrc.net"
```

Version 14

```
dd offset aCheckip_amazon ; DATA XREF: sub_3D95B0+4B↑r
                          ; "checkip.amazonaws.com"
dd offset aIpecho_net_0 ; "ipecho.net"
dd offset aIpinfo_io_0  ; "ipinfo.io"
dd offset aApi_ipify_or_0 ; "api.ipify.org"
dd offset aIcanhazip_co_0 ; "icanhazip.com"
dd offset aMyexternalip_0 ; "myexternalip.com"
dd offset aWtfismyip_co_0 ; "wtfismyip.com"
dd offset aIp_anysrc_ne_0 ; "ip.anysrc.net"
```

If it receives the response it expects from this request, it starts contacting the C2s it has obtained from its configuration to start reporting information on the new victim, check for updates, and receive new modules that expand its capabilities.

In normal configurations, after making certain requests with different commands that report host information to one of the C2s in its configuration, it obtains the IP of a specific server from which it can download new modules through port 447/tcp.

All downloads of configurations and modules are encrypted with the same algorithm (AES CBC) and all the files are saved encrypted to the disk. After updating and downloading the configurations and modules that it has in the configuration, it decrypts and maps the first module in the memory of its own process, "**systeminfo**", which is responsible for collecting information such as OS version, CPU type, the amount of RAM, the users of the system and the list of installed programs and services:

```
<systeminfo>
<general>
<os>Microsoft Windows 7 Professional  Service Pack 1 32 bits</os>
<cpu>Intel(R) Core(TM) i3</cpu>
<ram>3,41 GB</ram>
</general>
<users>
<user>Administrador</user>
<user>Invitado</user>
<user>jhon</user>
</users>
<installed>
<program>7-Zip 16.04</program>
<program>AddressBook</program>
<program>Adobe Flash Player 17 ActiveX</program>
<program>Adobe Flash Player 17 NPAPI</program>
<program>Connection Manager</program>
<program>DirectDrawEx</program>
<program>DXM_Runtime</program>
<program>Fontcore</program>
<program>IE40</program>
<program>IE4Data</program>
<program>IE5BAKEX</program>
<program>IEData</program>
<program>MobileOptionPack</program>
<program>MPlayer2</program>
```

```
<service>.NET CLR Data</service>
<service>.NET CLR Networking</service>
<service>.NET CLR Networking 4.0.0.0</service>
<service>.NET Data Provider for Oracle</service>
<service>.NET Data Provider for SqlServer</service>
<service>.NET Memory Cache 4.0</service>
<service>.NETFramework</service>
<service>1394 OHCI Compliant Host Controller</service>
<service>Controlador Microsoft ACPI</service>
<service>ACPI Power Meter Driver</service>
<service>Adobe Acrobat Update Service</service>
<service>adp94xx</service>
<service>adpahci</service>
```

Then it loads the **injectDll32** module together with its configuration files:

injectDll32_configs	17/11/2016 21:43	Carpeta de archivos	
injectDll32	30/03/2017 11:06	Archivo	512 KB
systeminfo32	30/03/2017 11:06	Archivo	22 KB

dinj	30/03/2017 11:06	Archivo	41 KB
dpost	30/03/2017 11:06	Archivo	1 KB
sinj	30/03/2017 11:06	Archivo	23 KB

Once this module is loaded, in case the user visits one of the websites listed in the configuration files (such as * cey-ebanking.com / CLKCCM / *) of this module, it captures the relevant browsing data and sends them to their C2:

```
POST /pre7/                                          2B2/60/ HTTP/1.1
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC
6.0; InfoPath 3; .NET4.0C; .NET4.0E; Tablet PC 2.0)
Host
Connection: close
Content-Type: multipart/form-data; boundary=---------JHFBUYJHBODCCZHU
Content-Length: 2384

-----------JHFBUYJHBODCCZHU
Content-Disposition: form-data; name="data"

POST /_vti bin/Lists.asmx HTTP/1.1
Host: www.          .com
Connection: keep-alive
Content-Length: 698
Accept: application/xml, text/xml, */*; q=0.01
Origin: https://www          .com
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36
Content-Type: text/xml:charset='UTF-8'
Referer: https://www.          .com/pages/default.aspx
Accept-Encoding: gzip, deflate
Accept-Language: es-ES,es;q=0.8
Cookie: TLTSID=6CEE02641ECD101E006AE38B698C7673; TLTUID=6CEE02641ECD101E006AE38B698C7673; cmTPSet=Y; CMAVID=none; _ga=GA1.2.479686626.1491918949; _gat=1

<soap:Envelope xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' xmlns:xsd='http://www.w3.org/2001/XMLSchema' xmlns:soap='http://schemas.xmlsoap.org/soap/
envelope/'><soap:Body><GetListItems xmlns='http://schemas.microsoft.com/sharepoint/soap/'><listName>PhotoGallery</listName><viewName></
viewName><query><Query><Where><Eq><FieldRef Name='BaseName'/><Value Type='Text'>04-11-2017.jpg</Value></Eq></Where></Query></
query><viewFields><ViewFields><FieldRef Name='Title'/><FieldRef Name='Customer_x0020_name'/><FieldRef Name='Photo_x0020_Location'/></ViewFields></
viewFields><rowLimit>1</rowLimit><queryOptions><QueryOptions></QueryOptions></queryOptions></GetListItems></soap:Body></soap:Envelope>...~I3...7....?
(.......<soap:Envelope xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' xmlns:xsd='http://www.w3.org/2001/XMLSchema' xmlns:soap='http://
schemas.xmlsoap.org/soap/envelope/'><soap:Body><GetListItems xmlns='http://schemas.microsoft.com/sharepoint/soap/'><listName>PhotoGallery</listName><viewName></
viewName><query><Query><Where><Eq><FieldRef Name='BaseName'/><Value Type='Text'>04-11-2017.jpg</Value></Eq></Where></Query></
query><viewFields><ViewFields><FieldRef Name='Title'/><FieldRef Name='Customer_x0020_name'/><FieldRef Name='Photo_x0020_Location'/></ViewFields></
viewFields><rowLimit>1</rowLimit><queryOptions><QueryOptions></QueryOptions></queryOptions></GetListItems></soap:Body></soap:Envelope>
-----------JHFBUYJHBODCCZHU
Content-Disposition: form-data; name="keys"


-----------JHFBUYJHBODCCZHU
Content-Disposition: form-data; name="link"

https://www.          com/_vti_bin/Lists.asmx
-----------JHFBUYJHBODCCZHU--
```

As discussed in the **DevCentral** report, version 9 of **trickbot**, a new module was added to the Trickbot toolset called "**mailsearcher**". Then in the case of being in the configuration will also be loaded into the victim system. The order in which the modules are loaded will depend on the configuration file.

"**mailsearcher**" is responsible for searching all the files of each disk connected to the system and comparing the extensions of the files with the following list:

```
                              ; "avi"
dd offset aMov                ; "mov"
dd offset aMkv                ; "mkv"
dd offset aMpeg               ; "mpeg"
dd offset aMpeg4              ; "mpeg4"
dd offset aMp4                ; "mp4"
dd offset aMp3                ; "mp3"
dd offset aWav                ; "wav"
dd offset aOgg                ; "ogg"
dd offset aJpeg               ; "jpeg"
dd offset aJpg                ; "jpg"
dd offset aPng                ; "png"
dd offset aBmp                ; "bmp"
dd offset aGif                ; "gif"
dd offset aTiff               ; "tiff"
dd offset aIco                ; "ico"
dd offset aXlsx               ; DATA XREF:
                              ; "xlsx"
dd 0                          ; DATA XREF:
dd offset aDocx               ; "docx"
align 10h
dd offset aZip                ; "zip"
```

This module reports itself to a specific C2 that it obtains from its own configuration:

```
5...j...<mail> ..<handler>___________8.78:4
43</handler>..</mail>.K..L......-F,...(.
```

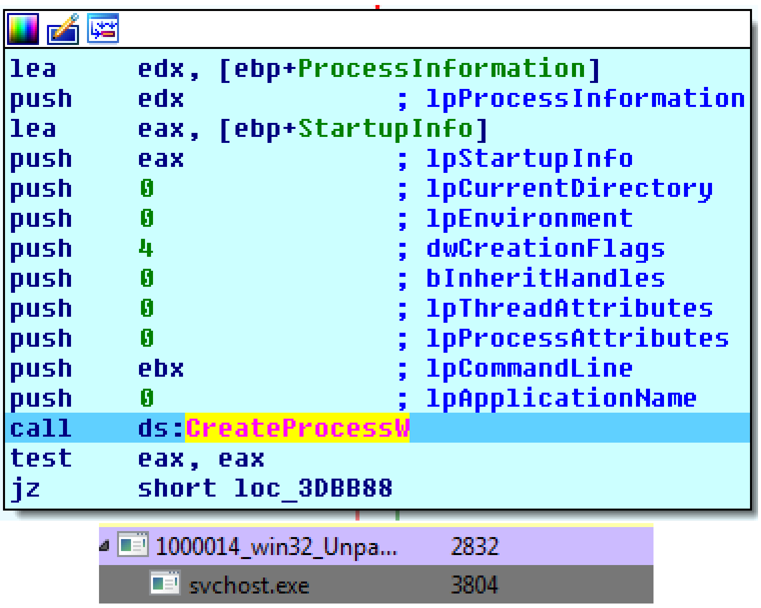
The URI of the request is different from the one used by the "core" of **Trickbot**, since in this case it has the structure "[IP]/[group_id]/[client_id]/send/" and uses its own User-Agent "KEFIR ! "Which makes it much more independent than the other modules found to date.

What is seen in this section describes the actions performed by **Trickbot** after its first execution. From this moment **Trickbot** enters a loop where from time to time it checks if there is a new configuration and if there are new versions of the malware or of some of the modules. In addition, within the same loop, it performs reports with the information it collects.

4. MODULE LOADING SYSTEM

During the analysis it has been observed that **Trickbot** uses events to control the execution flows between the core and the modules. In addition, the core performs the resolution of the Windows APIs of the modules. Let's see how this core communication system works with the modules.

First it creates a svchost.exe child process suspended with the CreateProcessW function:



Later with the CreateEventW function, it creates three events that will be used to manage the waits and communications between the main executable (**Trickbot**) and the svchost child process.

```
add     esp, 0Ch
push    esi                ; lpName
push    esi                ; bInitialState
push    esi                ; bManualReset
push    esi                ; lpEventAttributes
mov     esi, ds:CreateEventW
call    esi ; CreateEventW
push    0                  ; lpName
push    0                  ; bInitialState
push    0                  ; bManualReset
push    0                  ; lpEventAttributes
mov     [edi+6Ch], eax
call    esi ; CreateEventW
push    0                  ; lpName
push    1                  ; bInitialState
push    1                  ; bManualReset
push    0                  ; lpEventAttributes
mov     [edi+70h], eax
call    esi ; CreateEventW
mov     ecx, [ebp+hThread]
```

Once it has the handlers of the three events, using VirtualAllocEx and WriteProcessMemory it injects in the suspended svchost process 32 bytes of data like the following:

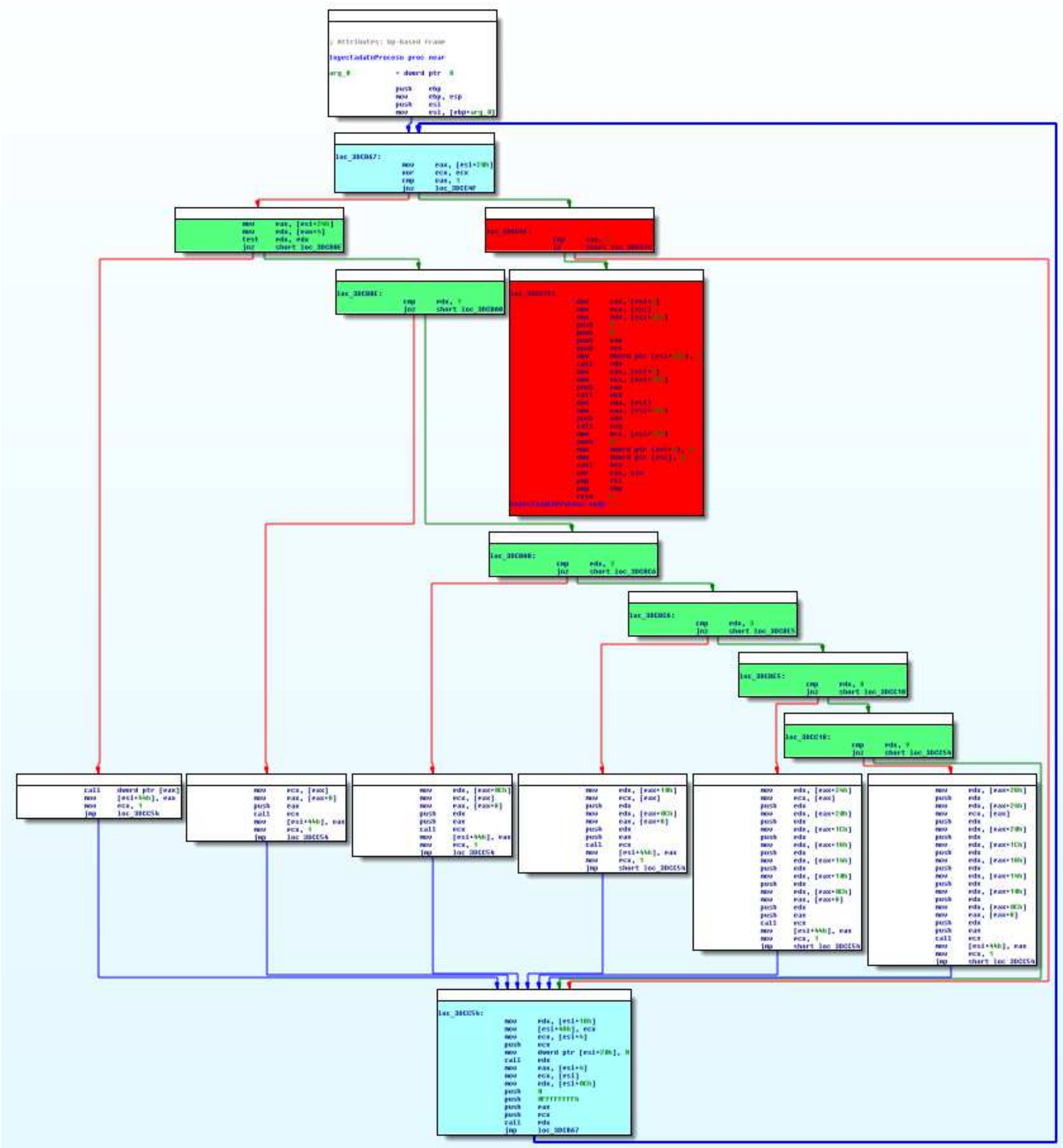
```
00090000  04 00 00 00 08 00 00 00  0C 00 00 00 A4 F8 10 77  ............ñ°.w
00090010  36 11 0F 77 10 14 0F 77  DD 16 0F 77 10 7A 0F 77  6..w...w¦..w.z.w
```

The first three groups of 4 Bytes (in **red** boxes) represent the identifiers of events that have created **trickbot** previously and that will use for their communication, in this case 4, 8 and C respectively.

The following 5 groups of 4 Bytes (in **purple** boxes) represent the offsets in the memory itself of the svchost process, from the following functions of the kernel32.dll library:

- SignalObjectAndWait
- WaitForSingleObject
- CloseHandle
- ExitProcess
- ResetEvent

Using the same injection method, it loads its own function into another offset of the svchost memory that will be used as the intermediary between **Trickbot** and the module code.



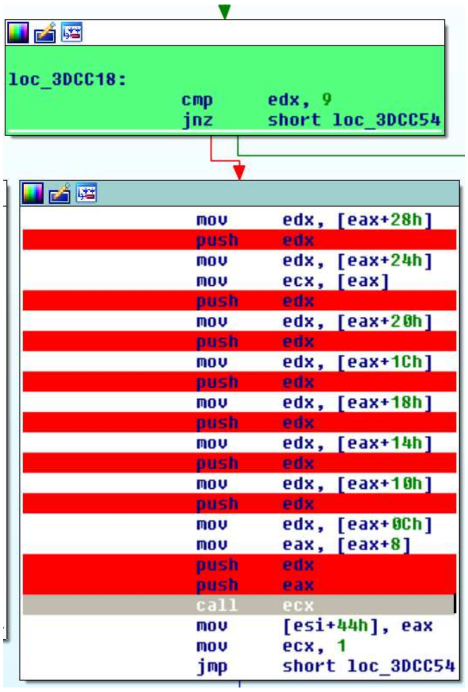
This feature is one of the most characteristic details of the **Trickbot** module management.

It is in charge of keeping itself waiting for orders from the main process. These come as offsets from functions within the memory of the svchost process itself and parameters with which to call them. This information is obtained through scripts in its own memory by Trickbot as detailed in the previous case.

Most of its logic consists of a loop that starts and ends in code zones with a **blue** background; after the first instructions, in case of detecting a problem with the process, it enters the area marked in **red** that closes the handlers of the events and the process itself.

In case everything goes correctly, the zone in which it enters consists of a switch, marked in **green**. Depending on the number of parameters needed by the function to call, enter one of the blank zones.

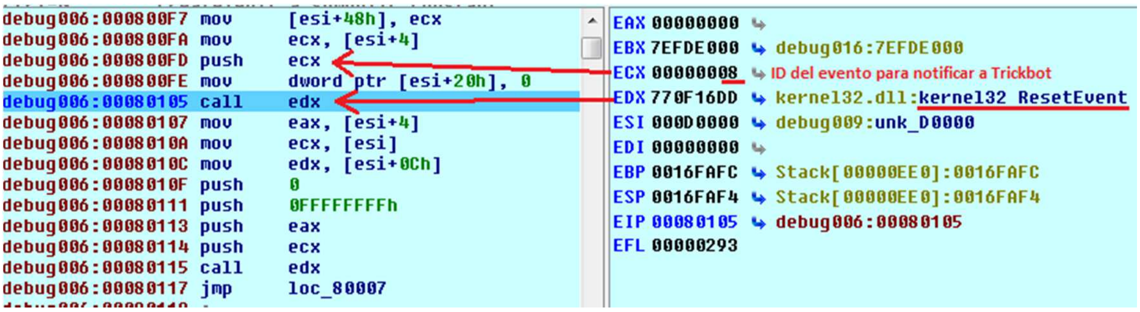
In the case of the following screenshot, if the number of parameters (which it has loaded in edx) coincides with 9, it enters a zone with nine calls to "push edx" with which it is loading parameters in the stack extracted from consecutive offsets after eax. Finally, it makes a call to ecx, where it has loaded the first offset of eax in the fourth instruction of this zone and that corresponds to the position of a function.



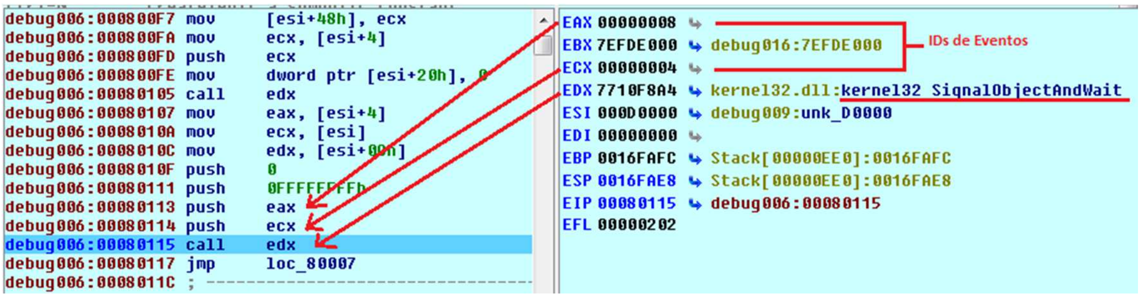
In the next screenshot you can see an example of calling a function like this and the status of the registers during the execution.

To manage the wait between the parent and child process, **Trickbot** uses the events it created before the injections into the process.

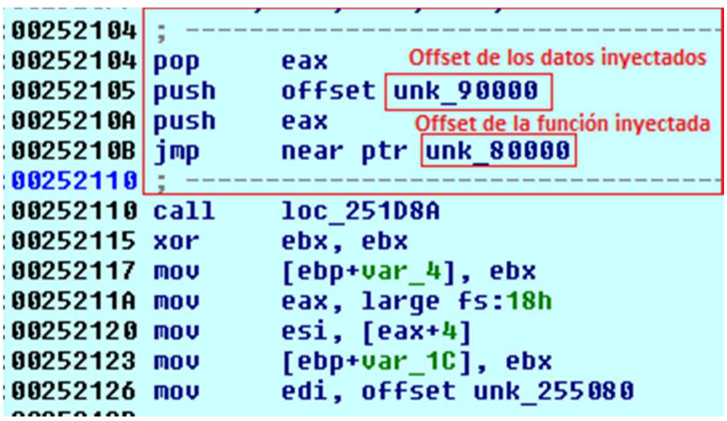
Using these events, when it reaches the last zone of the loop (in the previous screenshot marked with **blue** background) it contains two calls that correspond to a ResetEvent that notifies Trickbot that it has reached the end of the loop:



And a call after SignalObjectAndWait, to which it passes the IDs of two events. This function leaves the process suspended waiting for **Trickbot** to do a ResetEvent of the event in this case with ID 4, which means that it has loaded the new parameters into the memory for the next iteration of the loop:

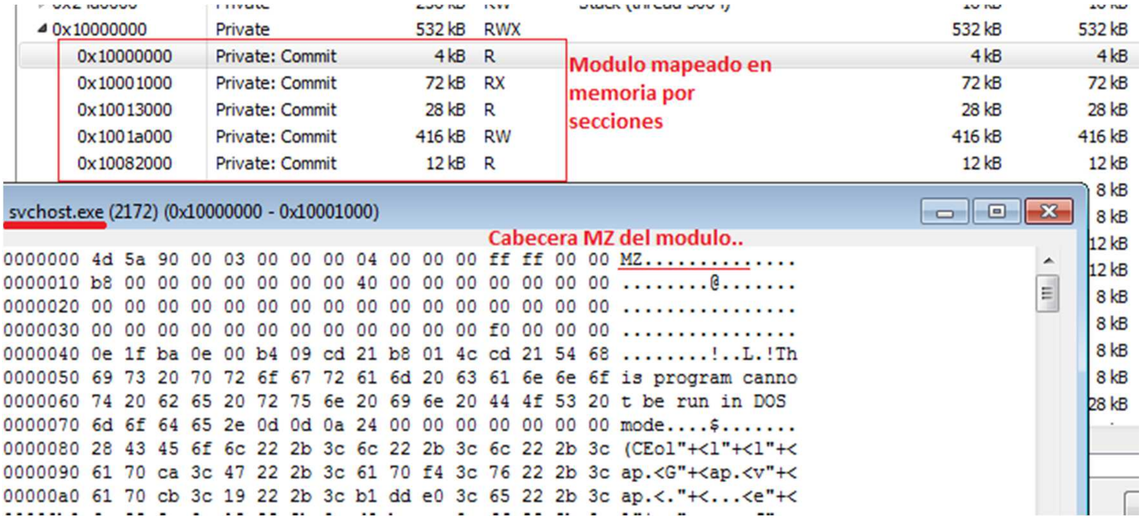


Before starting the execution of this process, it injects in the Entry Point of svchost, four lines that redirect the flow of the main thread to the previous function, passing it as a parameter, the 32 bytes of data injected at the beginning:



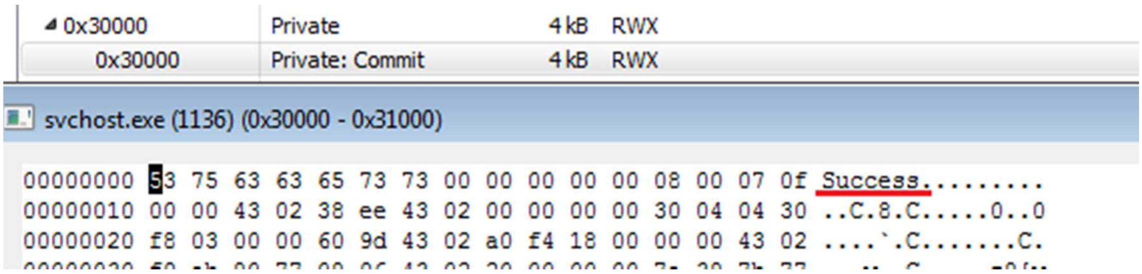
After preparing all that, it calls ResumeThread and the process goes into execution.

During the first iterations of the loop, **Trickbot** maps one of the modules in the process memory, section by section:



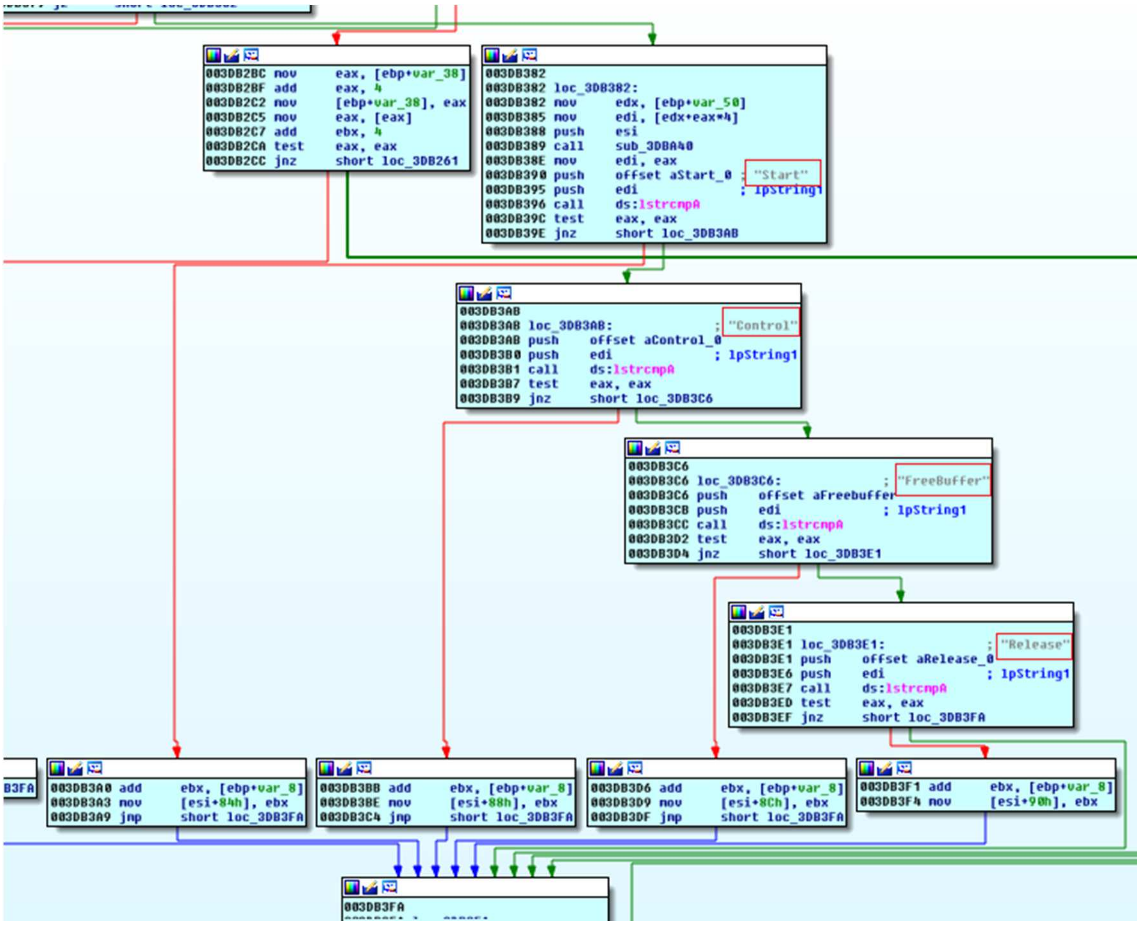
In the next iteration, using the data that the parent process has passed to it, it loads all the DLLs required by the newly loaded module with LoadLibrary and the functions of these that it will need with GetProcAddress.

Finally it calls an initialization function of its own module, which writes the "Success" string in one of the memory zones edited by **Trickbot**, in case everything is correct.



From this point, this last iteration is suspended with the call to SignalObjectAndWait, waiting for **Trickbot** to require, for example, the reporting information of said module.

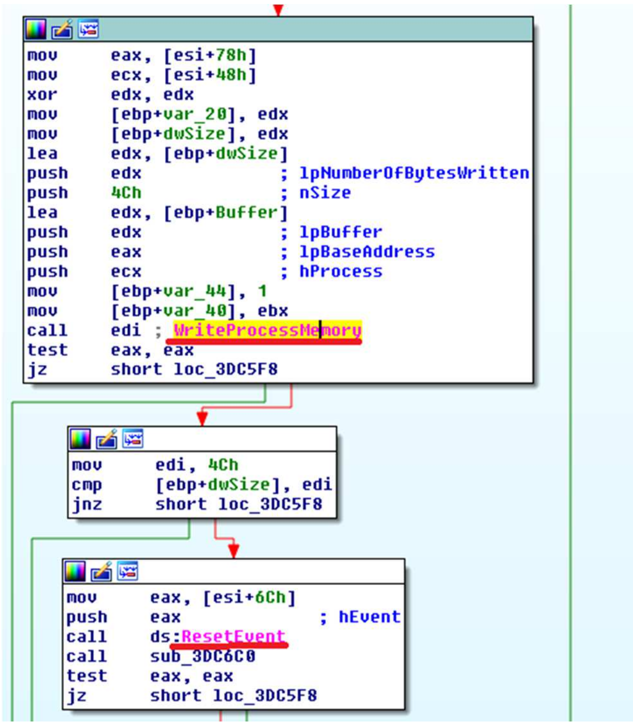
From the main process side, you can see how it contains a function to call the different functions that export each of its modules. These functions are those that each module exports, since the modules are DLL's and as such they export functions to be used by the core. To date these functions have not been changed in any of the versions and these are Start, Control, Freebuffer and Release.



systeminfo32.dll

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
(nFunctions)	Dword	Word	Dword	szAnsi
00000001	00002190	0000	0000517F	Control
00000002	00002230	0001	00005187	FreeBuffer
00000003	00002180	0002	00005192	Release
00000004	000021D0	0003	0000519A	Start

To make the transfer of information to the module, after passing through the area of the function which it wants to call, it performs a WriteProcessMemory of the data in question and calls ResetEvent for the module to start working.



5. NETWORK CONNECTIONS

For communications with its C2S, this malware uses HTTPS requests, which complicates the identification of its traffic by means of tools like NIDS to use, since that traffic is encrypted.

Usually these communications are done through port 443, although not always, since from the first versions, it began to use port 447 of some specific C2 to download the modules.

A differentiating element of its traffic is its User-Agent, since at first it identified it perfectly: it used the chain TrickLoader in all its requests:

```
unicode 0, <TrickLoader>
dd offset aTrickloader
```

In intermediate versions of the same it became somewhat less obvious, but maintained an unusual structure and easy to detect, becoming the "Xmaker" chain:

```
unicode 0, <Xmaker>,0
align 10h
```

In recent versions, as another of the changes clearly aimed at making this malware less detectable, the authors have begun to use a much more generic User Agent:

```
unicode 0, <Mozilla/5.0 (Windows NT 6.1; WOW64; rv:51.0) Gecko/201001>
unicode 0, <01 Firefox/51.0>,0
```

The requests are formed in such a way that a great amount of the information that reports to C2 goes in the URI, being the majority of these requests of type GET, excepting more extensive shipments of information collected by its modules, that it sends by POST.

```
GET /pre7/                                4BC93524F/5/spk/ HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0
Host: 203.

HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Mon, 03 Apr 2017 16:32:41 GMT
Content-Type: binary
Connection: keep-alive
Content-Length: 224

R.'.h.W,.......|..
...
..o..5..al.....N.g..#...%..t.$l..N....jL....
.....m..\t..q.e.@...z....S!.0.0.7..!.2W...
{.=~1l....i.g..5...-.........R...P:..v.> ......H.3:o)..^<..rq....?........,..._...-.y.b....'y...2h.K....0$48
.o...GET /pre7/                                 34BC93524F/0/Windows%207%20x86%
20SP1/1015/                                              9E5EA20F35C5/qLT1U1CBNUBkKztuW2gnPeZ/ HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0
Host: 203.

HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Mon, 03 Apr 2017 16:32:41 GMT
Content-Type: text/plain
Connection: keep-alive
Content-Length: 371

/1/pre7/                          C16B4BC93524F/qLT1U1CBNUBkKztuW2gnPeZ/272/
.F...P...0G...i .i.......u86..&;.. ....Ld..]..[G.z...f.!...C....)..6[...x.b.=.
.df'.=...P....<....@&w....>..D\94.:X"J.(.>"T.....Y..h9h.7.."V>.D.d.0.S@.d...>N.]..1q.z.
......Ji7X.b....H._...R..0o......t.</.K...XN....V.kY.2.Wf.>3"~.N:.Md..-8e..Q....{.......^......].....$0?.
1234567890GET /pre7/                              B4BC93524F/10/62/QEYZLSVQPLSANAVA/1/ HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0
Host: 203.
```

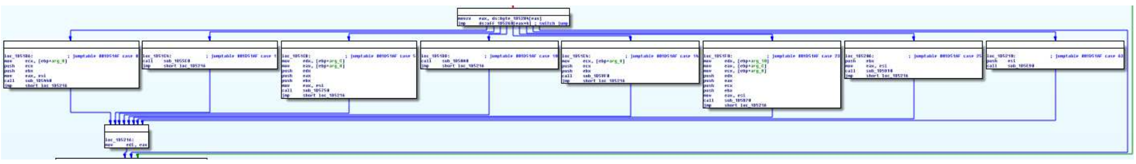
Among the data that contains the URIs of the requests, you can find the identifier of the current campaign and the user ID that it saves in the two files that it generates along with the executable, in the first stages of its execution. You can also find a number that identifies the order that it is sending to the C2 so that it can differentiate what it is requesting or reporting to it, and later different extra data related to the command in question.

From what we have analyzed and from information obtained from different external analyses, we have created the following table with a summary of the functionality of each order that we identified.

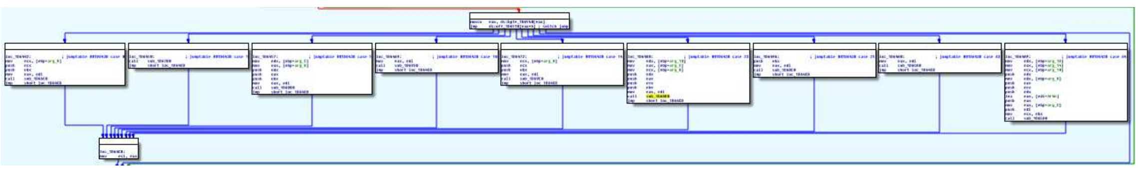
ID		
0	URI	/[group_id]/[client_id]/0/[version de windows]/[idioma del sistema]/[ip externa]/[sha256]/[key de sesión]/
	Description	Report with basic information about the client.
1	URI	/[group_id]/[client_id]/1/[key de sesión]/
	Description	Keep alive.
5	URI	/[group_id]/[client_id]/5/[modulo/configuración]/
	Description	Download of a module or configuration of a module.

10	URI	/[group_id]/[client_id]/10/62/[key de sesión]/1/
	Description	Start of module.
14	URI	/[group_id]/[client_id]/14/[key de sesión]/[value]/0/
	Description	Report with information on errors, checks, and other information
23	URI	/[group_id]/[client_id]/23/[config ver]/
	Description	Base configuration update
25	URI	/[group_id]/[client_id]/25/[key de sesión]/
	Description	Bot update
60	URI	/[group_id]/[client_id]/60/
	Description	Traffic report captured by the injectDll module
63	URI	/[group_id]/[client_id]/63/[module name]/[module command]/[result - base64]/[root tag of output XML]/
	Description	Systeminfo o injectDll Report
64	URI	-
	Description	Everything points to a command related to the mailsearcher module. What we have seen is that it performs POST requests with multipart content. It aims to be an exfiltration command, but that is still being verified.

From the **Trickbot** code, you can see how in one of its functions it contains the switch that is in charge of directing the execution flow that generates these requests depending on the command. In the following image you can see this code for one of its older versions (Version 1000005):



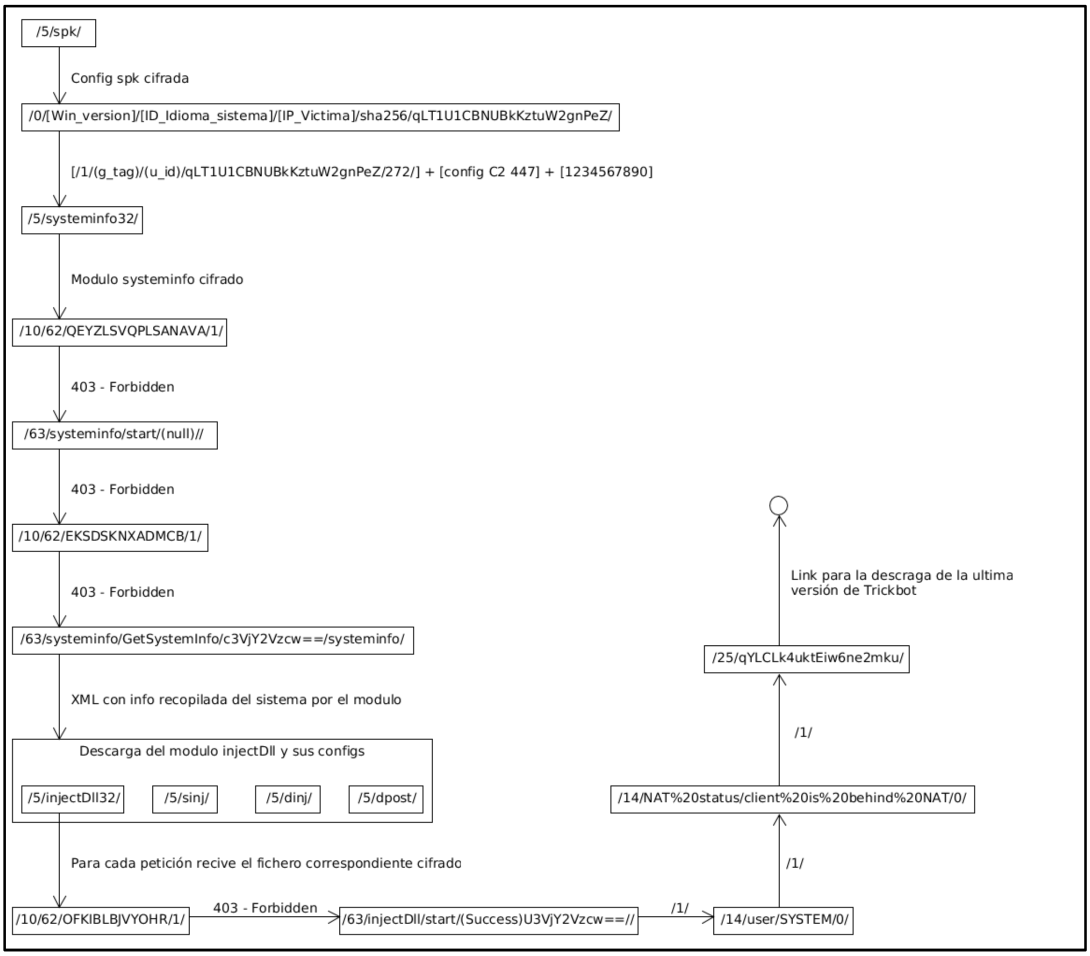
Analyzing the same function of one of the most recent versions (Version 1000010), we can see how they have added an extra option after the last one, which corresponded to



the command with number 63, and which is accessed with a new command number 64:

The functions that are executed from passing through this new area of code (command number 64) are very similar to those of the command 63, so it is probably also a command to perform reporting. The appearance of this new command (64) coincides in time with the appearance of the new module "***mailsearcher***", so everything indicates that these are related.

After the execution of the sample corresponding to version 14 in a controlled environment, we analyzed its traffic flow which shows a good part of the behavior of the execution of this malware.



The first part of the requests has been omitted to simplify commands.

6. ***ENCRYPTION MECHANISM***

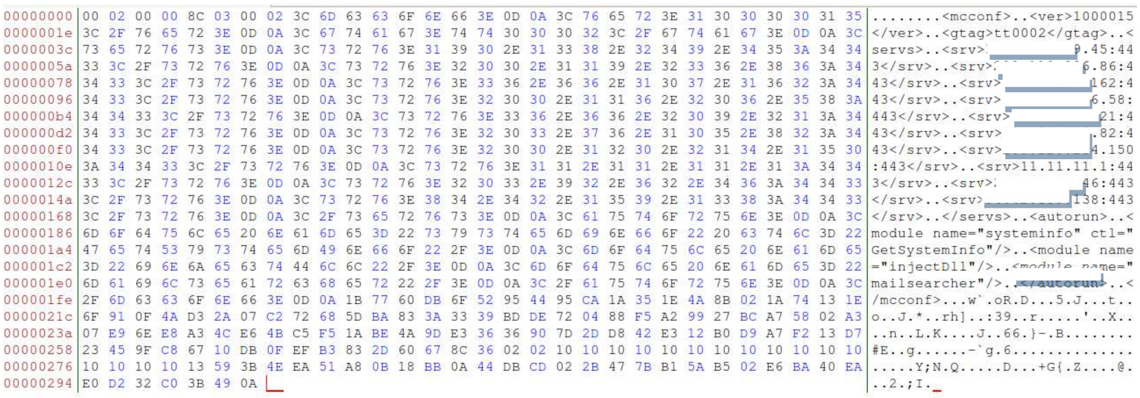
In the great work done by **malwarebytes** **(@hasherezade)** it is detailed that the encryption algorithm used by Trickbot is ***AES CBC 256 bits*****.** Also in the same entry on this subject we are told that the first DWORD is about the size of the data. In addition, **@hasherezade** offers resources after its research to decipher both the configurations as the modules, which makes it easier to understand **Trickbot** and its evolution.

Based on this information and visualizing how the content is decrypted, it is easy to perform the reverse process and build a script or modify the one made by hasherezade, to provide us with the ability to encrypt configurations modified by us to more easily manipulate **Trickbot** execution flows. The implementation of the encryption function would be as simple as:

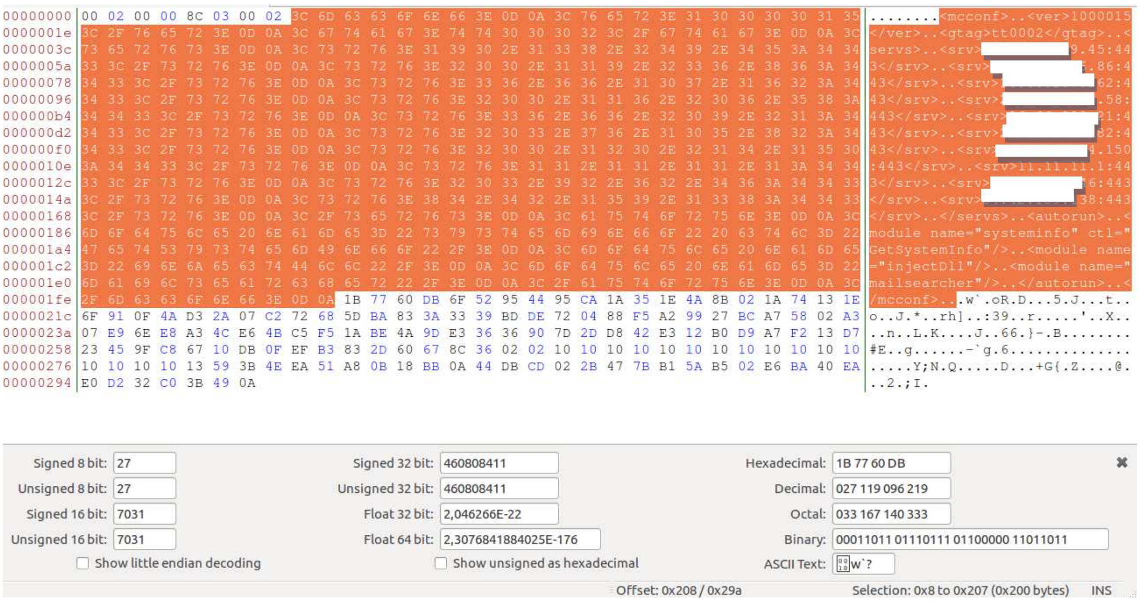
```python
def aes_encrypt(data):
    token = Random.new().read(0x30)
    key = hash_rounds(token[:0x20])[:0x20]
    iv = hash_rounds(token[0x10:0x30])[:0x10]
    aes = AES.new(key, AES.MODE_CBC, iv)
    data = pad(data)
    result = token + aes.encrypt(data)
    return result
```

To perform this process we can start from a configuration that we get encrypted and with the **@hasherezade** script we can decrypt it. Once decrypted, we can modify it, as in the following example where we add the local IP address 11.11.11.1:443 (ip of the laboratory environment) and load the module “***mailsearcher***”. With this we expect it to use the IP 11.11.11.1:443 as command and control and load the module "***mailsearcher***" which does not usually come by default.

After modifying it with a hexadecimal editor we would have the following:



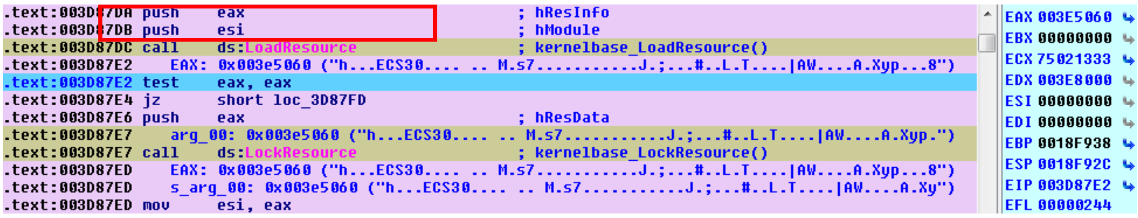
After the first 8 bytes is when the configuration data starts as such. In these first 8 bytes, it will be where **Trickbot** will look for the size of the data that will come next. In the case of example that corresponds to the value **02 00** (in the image it is upside down, 00 02), this would be 0x200 bytes. If we select the dataset we will see that it has just the right size of 0x200 bytes:



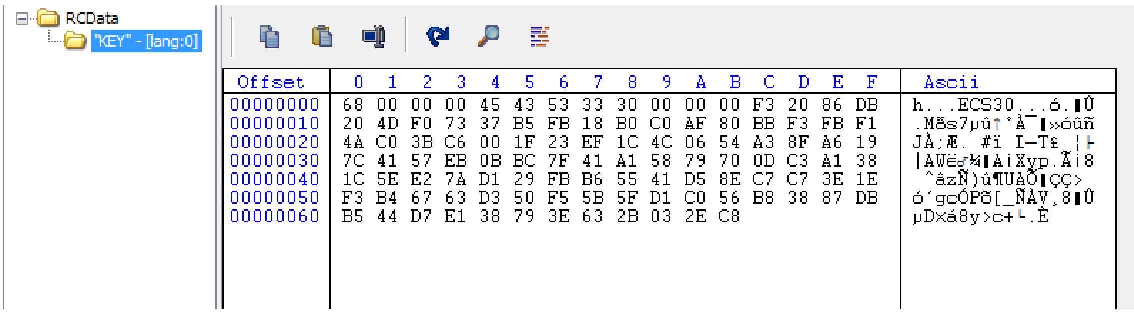
Therefore, after modifying the information we must set the first bytes to tell **Trickbot** the exact size of the data. Then we encrypt with the function we have called aes_encrypt(). With this we will have a new configuration that will not yet be fully functional.

The reason it does not work is because Trickbot, after the encrypted data, places **the hash signature of the data**. Therefore if we modify the content of the configuration we have to calculate the signature of the data since it verifies it after reading the configuration. To calculate the hash signature of the data that it has just read it uses the KEY that comes in the binary resources. We will see below how it loads the resources key:

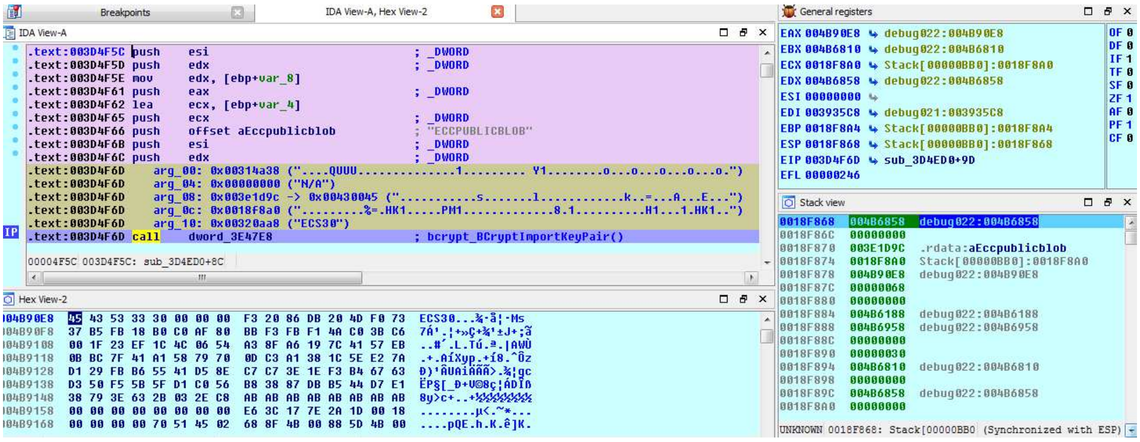
Then it will execute the function LoadResource () and we will see in EAX the value where the KEY will be:



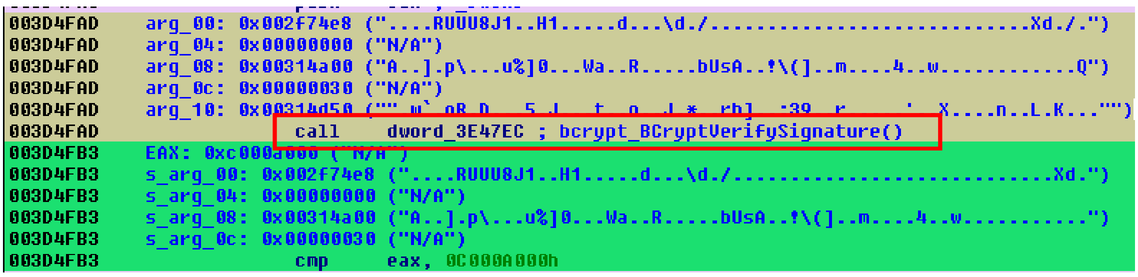
This is what the key in the resources looks like (you will see that the presented binary does not have the typical CONFIG resource of version 14 of Trickbot, this is to force it to read the configuration of the config.conf file. This is not necessary but we have done it so that you can change the configuration in a simpler way):



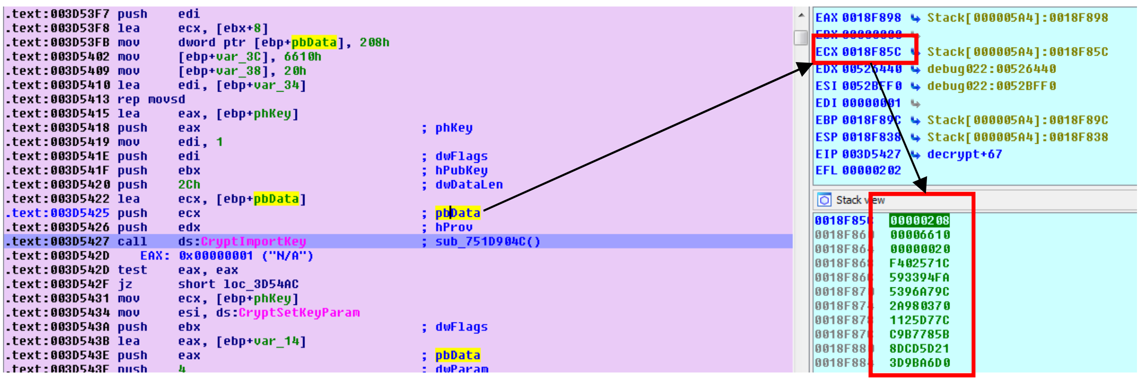
And we shall see that this is the key that imports the function BCryptImportKeyPair() when it does the push eax. The value of EAX is equal to 0x004B90E8, which as we can see in the hexadecimal view corresponds to the key that was in the resources:



After importing the key, it uses the BcryptVerifySignature() function to do the signature verification.



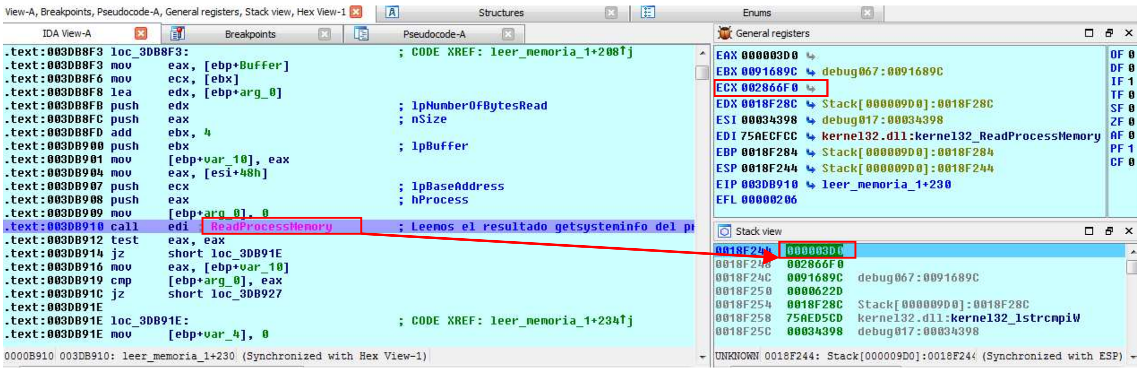
The other key that **Trickbot** uses is, as we have mentioned to decrypt the configuration and the modules, and we will see how it is imported by the function of the API CryptImportKey():



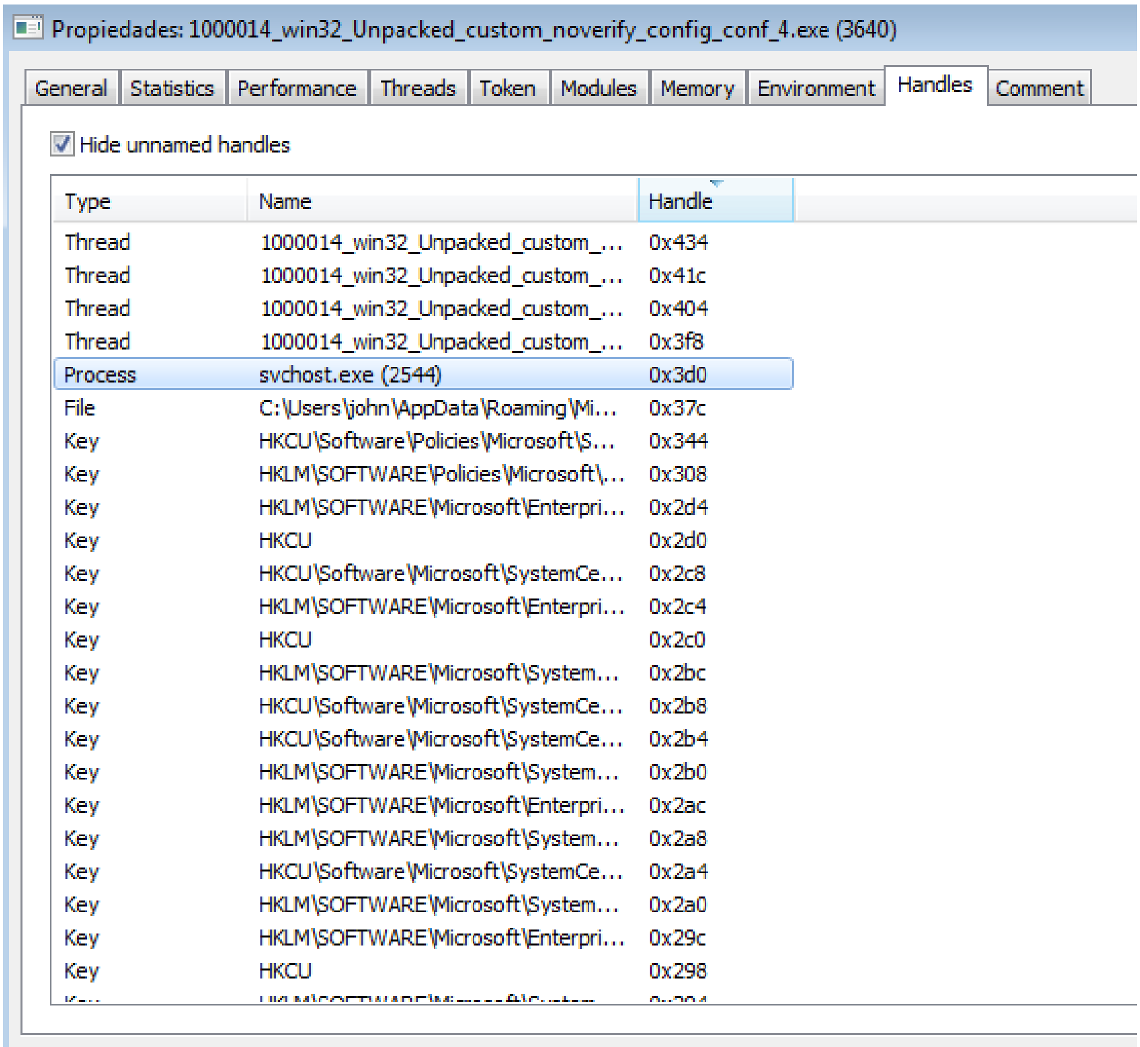
At this point we have two options: or modify the program execution flow so that the verification process will always tell us that the signature is correct or to replicate the process of signing the hash of the data that **Trickbot** performs. For simplicity we have chosen to modify the execution flow of the binary so that it does not need to be properly signed.

7. IPC MECHANISM (Inter-Process Communication)

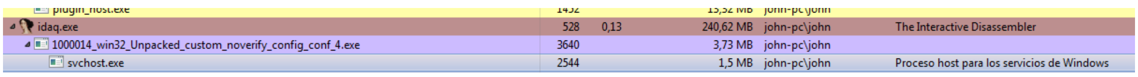
One of the interesting aspects of this malware is how it retrieves the information from the modules. It uses ReadProcessMemory over the child processes it has created. Below we will see the example where **Trickbot** (the core) reads what the systeminfo module returns. If we stop in one of the ReadProcessMemory that we have identified, we see that it passes the handle of the remote process (3D0) as a parameter:



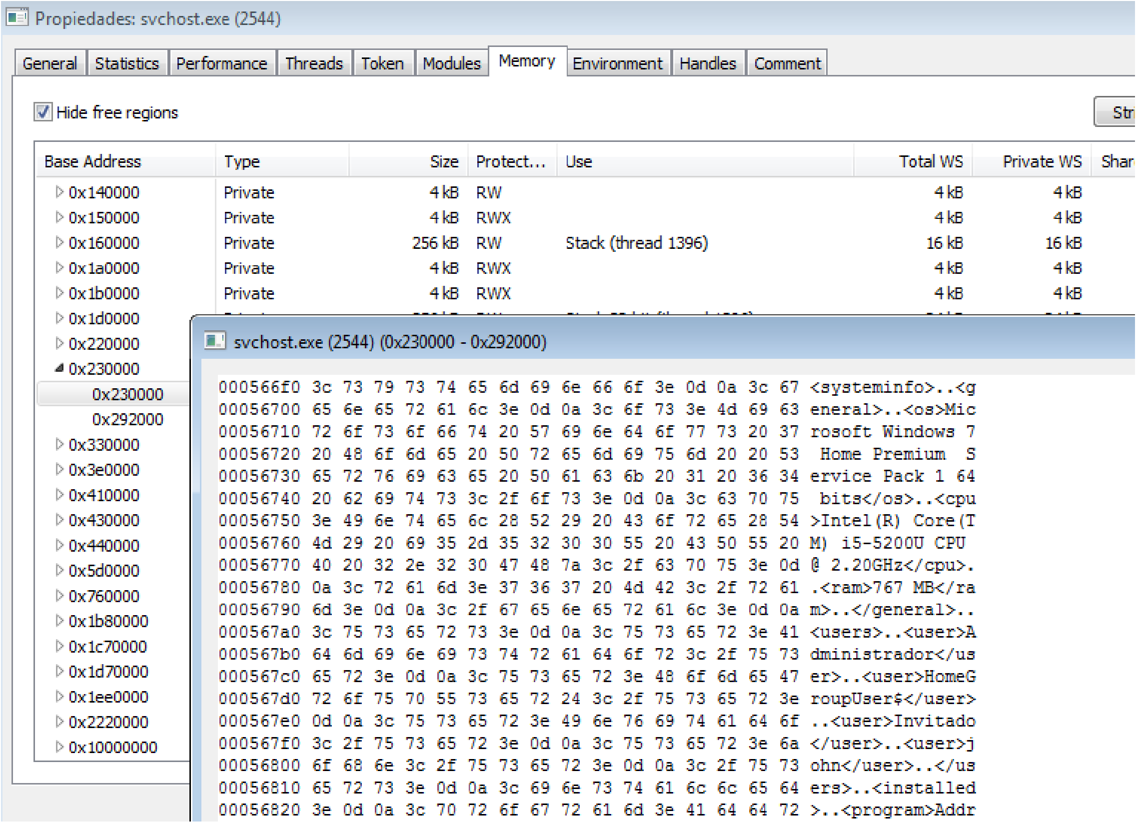
In the following image we will see better how the 3D0 handler corresponds to the child process svchost.exe:



We can see the PID of the parent and child process here:



The memory address it wants to read (lpBaseaddress) is **0x2866f0**, as we can see in the ECX register of the ReadProcessMemory() image. As we have already said it wants to read it from the remote process svchost (handler 3D0) and at that moment what contains that memory address is:



We can see in **0x2866f0 (230000 + 566f0)** that the information is collected by the module and that the core is accessing it. In this case, this information will be sent to C2 using the 63 command. We have seen an example of how the **Trickbot** core and the "systeminfo" module have exchanged the information.

8. RELATED FILES

The analyzed samples of **Trickbot** to date have always been installed in the user's %APPDATA% folder who executes it first. In this folder it copies itself and creates 2 files:

- **client_id:** It contains an infected user ID generated from system data.
- **group_tag:** A campaign code which is in the internal configuration that can be found encrypted in the resources of the executable, once unpacked, along with the decryption key.

1000014_Trickbot	13/03/2017 12:18	Aplicación	286 KB
client_id	06/04/2017 13:00	Archivo	1 KB
group_tag	06/04/2017 13:01	Archivo	1 KB

Apart from these files, if it has connectivity, it will download an updated configuration that will be saved as encrypted "config.conf" in the same folder, and will create a "Modules" folder.

In the folder called Modules it will download the modules that contain its encrypted configuration files, and folders with the configuration files of some of the modules. The folders with the configurations of each module will have names following the pattern: "<module name>_config".

injectDll32_configs	06/04/2017 13:05	Carpeta de archivos	
injectDll32	30/03/2017 11:06	Archivo	512 KB
systeminfo32	30/03/2017 11:06	Archivo	22 KB

When it obtains administration permissions, it copies itself to the folder:

```
C:\Windows\System32\config\systemprofile\AppData\Roaming
```

After executing this action, it removes the executable from the Roaming folder of the initial user, leaving the modules and configurations intact.

9. DETECTION

First, manually, you can find the files mentioned in section 8 in the folder: %APPDATA%, the only case that can vary is the main executable that can be found with different names depending on their origin, since the others to date have not changed at any time.

Depending on the scenario, you can also find one or two tasks called "bot" or "Drivers update", and "AplicationsCheckVersion", which will execute an application in the% APPDATA% directory every minute and when you log in respectively.

During its execution, it is easier to detect it among processes running on 32-bit computers, because it keeps the executable name replicated in % appdata%. On the other hand, 64-bit computers use the Microsoft svchost.exe process to hide when run by a normal system user. In the case of being invoked by the persistence task with SYSTEM permissions, it behaves the same as in 32-bit systems.

For automatic detection, there are no NIDS rules that can detect it through your traffic so far, since the fact that it is encrypted by SSL complicates it to a greater extent.

Yara rules have been developed to detect it in memory, since the executable comes packaged with different types of systems for each campaign and version, preventing a common rule.

The rules for detection in memory are as follows:

```
rule MALW_trickbot_bankBot : Trojan
{
 meta:
    author = "Marc Salinas @Bondey_m"
    description = "Detects Trickbot Banking Trojan"

strings:
        $str_trick_01 = "moduleconfig"
        $str_trick_02 = "Start"
        $str_trick_03 = "Control"
        $str_trick_04 = "FreeBuffer"
        $str_trick_05 = "Release"

condition:
        all of ($str_trick_*)
}
```

```
rule MALW_systeminfo_trickbot_module : Trojan
{
meta:
        author = "Marc Salinas @Bondey_m"
        description = "Detects systeminfo module from Trickbot Trojan"

strings:

        $str_systeminf_01 = "<program>"
        $str_systeminf_02 = "<service>"
        $str_systeminf_03 = "</systeminfo>"
        $str_systeminf_04 = "GetSystemInfo.pdb"
        $str_systeminf_05 = "</autostart>"
        $str_systeminf_06 = "</moduleconfig>"

condition:
        all of ($str_ systeminf_*)
}
```

```
rule MALW_dllinject_trickbot_module : Trojan
{
meta:
        author = "Marc Salinas @Bondey_m"
        description = " Detects dllinject module from Trickbot Trojan"

strings:
        $str_dllinj_01 = "user_pref("
        $str_dllinj_02 = "<ignore_mask>"
        $str_dllinj_03 = "<require_header>"
        $str_dllinj_04 = "</dinj>"

condition:
        all of ($str_ dllinj_*)
}
```

```
rule MALW_mailsercher_trickbot_module : Trojan
{
meta:
        author = "Marc Salinas @Bondey_m"
        description = " Detects mailsearcher module from Trickbot Trojan"

strings:
        $str_mails_01 = "mailsearcher"
        $str_mails_02 = "handler"
        $str_mails_03 = "conf"
        $str_mails_04 = "ctl"
        $str_mails_05 = "SetConf"
        $str_mails_06 = "file"
        $str_mails_07 = "needinfo"
        $str_mails_08 = "mailconf"

condition:
        all of ($str_mails_*)
}
```

10. DISINFECTION

Taking into account the detection process, in case of finding traces of this threat in the system and that none of our system protection measures are able to detect or disinfect it, the ideal steps for disinfection would be to:

- Eliminate the task that is executed every minute, so that it does not restart the execution of the malware.
- Complete the Trickbot process with the task manager or with an application such as ProcessExplorer.
- Browse to the% APPDATA% folder where it is installed, to delete the main Trickbot executable and then the three files ("user_id", "group_tag" and "config.conf") and the Modules folder.
- Browse to the SYSTEM user's APPDATA folder (C:\Windows\System32\config\systemprofile\AppData\Roaming) to delete the same files from the SYSTEM user.

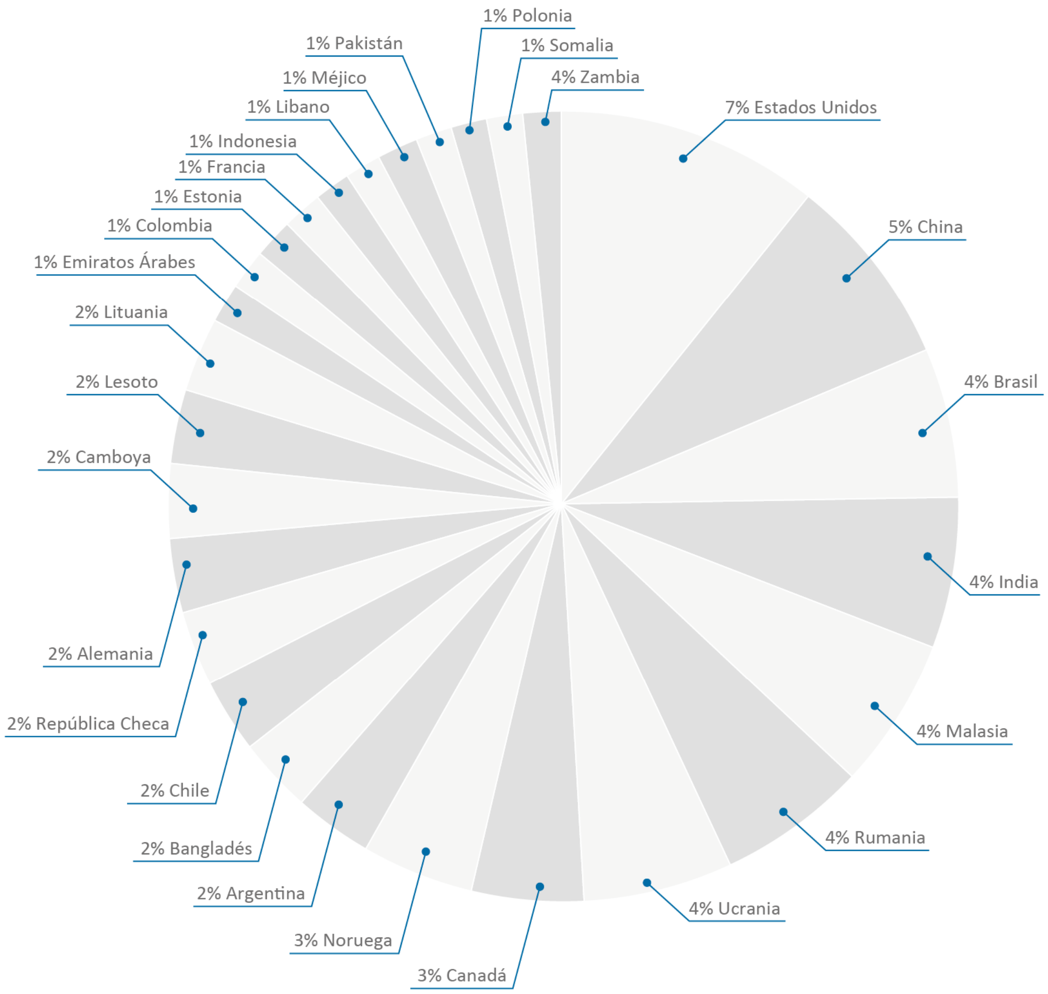
With this, we would have completely eliminated this threat from the system, although it would be advisable to review that the task of persistence has not been restored in case that just in the period of time between eliminating it and closing the process, it would have been in the early stages of execution its and would have replaced it, although it would not be dangerous as it could not find the executable in the system.

On the other hand, in cases where the infection has been through an ExploitKit, it is likely that in addition to **Trickbot**, our system is infected with other types of malware, since they usually do not install a single sample, so performing analyses with different tools would be recommended, reaching formatting in sensitive cases.

11. ATTACKER INFORMATION

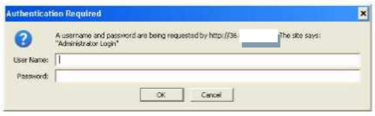
For the **Trickbot** infrastructure, as **@hasherezade** mentioned in its **post in the blog of Malwarebytes**, the IPs of its C2 correspond to devices such as Routers or IP Cameras (all tested with ARM processors) distributed by many different countries and in all the cases that we analyzed belonging to ISP of each of the countries that we will see below. The distribution of C2 countries (based on the configurations collected) is shown in the following chart where you can see how the United States and China stand out:

REPARTO DE C2 POR PAISES



Lista de países ordenada de mayor a menor número de C2 encontrados.

Most affected systems have an access web interface such as the following:



And in case of access by https to the URL formed by one of the **Trickbot** commands, the certificate that it shows us, is still the same as in the first versions analyzed in the post mentioned above:

General | Detalles

No se pudo verificar este certificado porque el emisor es desconocido.

Emitido para

Nombre común (CN)	rvgvtfdf
Organización (O)	tg4r6tds
Unidad organizativa (OU)	rst
Número de serie	00:C5:63:15:A8:0D:6A:86:E5

Emitido por

Nombre común (CN)	rvgvtfdf
Organización (O)	tg4r6tds
Unidad organizativa (OU)	rst

Periodo de validez

Comienza el	08/06/16
Caduca el	08/06/17

Huellas digitales

Huella digital SHA-256	34:04:69:57:08:B1:C8:F9:7D:B4:D4:E3:3C:57:F8:4F: 23:B0:DF:E0:BE:75:14:77:0B:43:2A:5B:A8:66:25:2D
Huella digital SHA1	92:75:D5:27:40:C0:B0:1C:E9:52:32:3D:0F:53:68:D7:8A:74:FF:BF

12. REFERENCES

https://blog.fortinet.com/2016/12/06/deep-analysis-of-the-online-banking-botnet-trickbot

http://www.threatgeek.com/2016/10/trickbot-the-dyre-connection.html

https://www.infosecurity-magazine.com/blogs/rig-ek-dropping-trickbot-trojan/

https://devcentral.f5.com/articles/is-xmaker-the-new-trickloader-24372

https://blog.malwarebytes.com/threat-analysis/2016/10/trick-bot-dyrezas-successor/

https://fraudwatchinternational.com/malware/trickbot-malware-works/

https://msdn.microsoft.com/en-us/library/windows/desktop/ms682425%28v=vs.85%29.aspx

https://msdn.microsoft.com/en-us/library/windows/desktop/aa366890%28v=vs.85%29.aspx

https://msdn.microsoft.com/es-es/library/windows/desktop/ms681674%28v=vs.85%29.aspx

https://msdn.microsoft.com/es-es/library/windows/desktop/ms682437%28v=vs.85%29.aspx

13. AUTHORS

- Marc Salinas
- José Miguel Holguín

MADRID	BARCELONA	VALENCIA	BOGOTÁ	MÉXICO D.F.
Velázquez 150, 2ª planta, 28002 T.(+34) 902 882 992	Llull, 321 (Edifici Cinc) 08019 T.(+34) 902 882 992	Ramiro de Maeztu 7, 46022 T.(+34) 902 882 992	Carrera 11 Nº 93A - 53, Of. 401 T.(+57 1) 74 5 74 39	44-7, México D.F. 06600 T.(+52) 55 2128 0681

info@s2grupo.es
www.s2grupo.es
www.securityartwork.es