

INNOVATION IN PROCESSES **MALWARE REPORT** Evolution of Trickbot *REPORT* 06/2017







5%

A BRNOR BERGERAR BERGERA



1.	INTRODUCTION	3
2.	INFECTION PROCESS	5
З.	TECHNICAL CHARACTERISTICS	6
4.	MODULE LOADING SYSTEM	13
5.	NETWORK CONNECTIONS	21
6.	ENCRYPTION MECHANISM	25
7.	IPC MECHANISM (Inter-Process Communication)	29
8.	RELATED FILES	32
9 .	DETECTION	33
10.	DISINFECTION	35
11.	ATTACKER INFORMATION	36
12.	REFERENCES	38
13.	AUTHORS	38



This document contains a review of the latest versions of a Trojan family known as <u>"Trickbot/TrickLoader"</u>. 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 <u>@hasherezade</u> and by <u>Xiaopeng Zhang</u> (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.



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.



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 Seg		
f	_vbaChkstk	.text
f	vbaExceptHandler	.text
f	vbaFPException	.text
f	_adj_fdiv_m32	.text
f	_adj_fdiv_m64	.text
f	DIIFunctionCall	.text
f	ThunRTMain	.text
f	sub_40365C	.text
f	sub_4036B0	.text
f	sub_403710	.text
f	sub_403930	.text
f	sub_4039C8	.text
f	sub_403A58	.text
f	sub_403B10	.text
f	sub_407590	.text
f	sub_407CE0	.text
f	sub_407E30	.text
f	sub_40C770	.text
f	sub_40C9E0	.text
f	sub_40CA50	.text
f	sub_40CD70	.text
f	sub_40CD80	.text
f	sub_40D760	.text
f	sub_40D9B0	.text

Unpacked

	f	sub_3D1000
	f	sub_3D1050
10	f	sub_3D10A0
10	f	sub_3D10D0
1	f	sub_3D1C30
1	f	sub_3D1E40
	f	sub_3D1EA0
	f	sub_3D1ED0
	f	sub_3D1F40
	f	sub_3D1FD0
	f	sub_3D2020
	f	sub_3D2140
	f	sub_3D21A0
	f	sub_3D2240
ļ	f	sub_3D2260
	f	sub_3D2300
	f	sub_3D2310
	f	sub_3D23B0
	f	sub_3D2470
	f	sub_3D25C0
l	f	sub_3D25F0
		-
	f	sub_3D2650
	f f	sub_3D2650 sub_3D2690
	f f f	sub_3D2650 sub_3D2690 wWinMain(x,x,x,x)
	f f f	sub_3D2650 sub_3D2690 wWinMain(x,x,x,x) sub_3D2E90
	f f f f	sub_3D2650 sub_3D2690 wWinMain(x,x,x,x) sub_3D2E90 sub_3D2E40
	f f f f f	sub_3D2650 sub_3D2690 wWinMain(x,x,x,x) sub_3D2E90 sub_3D2F40 sub_3D390
	f f f f f f	sub_3D2650 sub_3D2690 wWiMain(x,x,x,x) sub_3D2E90 sub_3D2F40 sub_3D3090 sub_3D3090
	f f f f f f	sub_3D2650 sub_3D2690 wWimMain(x,x,x,x) sub_3D2E90 sub_3D2E90 sub_3D3090 sub_3D31B0 sub_3D31B0
	f f f f f f f	sub_3D2650 sub_3D2690 wWinMain(x,x,x,x) sub_3D2E90 sub_3D2E40 sub_3D31B0 sub_3D31B0 sub_3D31E0 sub_3D31E0
	f f f f f f f f	sub_3D2650 sub_3D2690 wWinMain(x,x,x,x) sub_3D2E90 sub_3D2F40 sub_3D31B0 sub_3D31E0 sub_3D31E0 sub_3D3310 sub_3D3310 sub_3D3300
	<i>f f f f f f f f f f</i>	sub_3D2650 sub_3D2690 wWinMain(x,x,x,x) sub_3D2E90 sub_3D2F40 sub_3D3090 sub_3D31B0 sub_3D31E0 sub_3D310 sub_3D3310 sub_3D3340 sub_3D3580
	<i>f f f f f f f f f f</i>	sub_3D2650 sub_3D2690 wWimMain(x,x,x,x) sub_3D2E90 sub_3D2F40 sub_3D3090 sub_3D31B0 sub_3D31B0 sub_3D310 sub_3D310 sub_3D3340 sub_3D3580 sub_3D3240
	<i>f f f f f f f f f f</i>	sub_3D2650 sub_3D2690 wWiMain(x,x,x,x) sub_3D2E90 sub_3D2E90 sub_3D3090 sub_3D31B0 sub_3D31E0 sub_3D31E0 sub_3D3310 sub_3D3340 sub_3D35B0 sub_3D35B0 sub_3D3720 sub_3D3720
	<i>f f f f f f f f f f</i>	sub_3D2650 sub_3D2690 wWiMain(x,x,x,x) sub_3D2E90 sub_3D2400 sub_3D3180 sub_3D3180 sub_3D3180 sub_3D3310 sub_3D3340 sub_3D3580 sub_3D3580 sub_3D3720 sub_3D3720 sub_3D3740 sub_3D3790
	f f f f f f f f f f f f f f f f f f f	sub_3D2650 sub_3D2690 wWinMain(x,x,x,x) sub_3D2E90 sub_3D2E90 sub_3D3090 sub_3D31B0 sub_3D31E0 sub_3D3310 sub_3D3310 sub_3D3340 sub_3D35B0 sub_3D3720 sub_3D3740 sub_3D3740 sub_3D3700 sub_3D3700
	<i>f f f f f f f f f f f f f f f f f f f </i>	sub_3D2650 sub_3D2690 wWinMain(x,x,x,x) sub_3D2E90 sub_3D2F40 sub_3D3180 sub_3D3180 sub_3D3180 sub_3D310 sub_3D310 sub_3D330 sub_3D320 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370
	f f f f f f f f f f f f f f f f f f f	sub_3D2650 sub_3D2690 wWinMain(x,x,x,x) sub_3D2690 sub_3D2F40 sub_3D3180 sub_3D3180 sub_3D3180 sub_3D310 sub_3D3310 sub_3D3340 sub_3D3340 sub_3D3580 sub_3D3580 sub_3D370 sub_3D370 sub_3D370 sub_3D3600 sub_3D3070 sub_3D300
	チチチチチチチチチチチチチチチチチチ	sub_3D2650 sub_3D2690 wWiMain(x,x,x,x) sub_3D2E90 sub_3D2E90 sub_3D3090 sub_3D31B0 sub_3D31B0 sub_3D310 sub_3D310 sub_3D3310 sub_3D3580 sub_3D3580 sub_3D3740 sub_3D3740 sub_3D3740 sub_3D3740 sub_3D370 sub_3D370 sub_3D360
	<i>f f f f f f f f f f f f f f f f f f f </i>	sub_3D2650 sub_3D2690 wWiMain(x,x,x,x) sub_3D2E90 sub_3D2E90 sub_3D3090 sub_3D31B0 sub_3D31B0 sub_3D31E0 sub_3D3310 sub_3D3340 sub_3D3580 sub_3D3580 sub_3D3720 sub_3D3740 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 sub_3D4 su
	<i>f f f f f f f f f f f f f f f f f f f </i>	sub_3D2650 sub_3D2690 wWiMain(x,x,x,x) sub_3D2E90 sub_3D3090 sub_3D31B0 sub_3D31B0 sub_3D31E0 sub_3D310 sub_3D3340 sub_3D35B0 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D370 sub_3D3070 sub_3D3070 sub_3D3070 sub_3D400

.tex

.tex .tex .tex .tex .tex .tex .tex .tex .tex .tex .tex .tex .tex .tex .tex .tex .tex .tex .tex .tex .tex .tex .te> .tex tev

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.

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:

길 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).

⊿{	RCD	ata
	····· 🏠	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 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.



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

un.	unicoue 0, \ip.anysrc.net/,0			
dd	offset	aMyexternalip_c ; DATA XREF: sub_1D6F60+6A1r		
		; "myexternalip.com"		
dd	offset	<pre>aApi_ipify_org ; "api.ipify.org"</pre>		
dd	offset	<pre>alcanhazip_com ; "icanhazip.com"</pre>		
dd	offset	<pre>aBot_whatismyip ; "bot.whatismyipaddress.com"</pre>		
dd	offset	<pre>alp_anysrc_net ; "ip.anysrc.net"</pre>		

Version 14

	-
dd offset aCheckip_amazon	; DATA XREF: sub_3D95B0+4B1r
3	"checkip.amazonaws.com"
dd offset alpecho_net_0 ;	"ipecho.net"
dd offset alpinfo_io_0 ;	"ipinfo.io"
dd offset aApi_ipify_or_0	; "api.ipify.org"
dd offset alcanhazip_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>ACPI Power Meter Driver</service>
<service>Adobe Acrobat Update Service
<service>adp94xx</service>
<service>adp94xx</service>
<service>adpahci</service>
```

Then it loads the injectDII32 module together with its configuration files:

injectDII32_configs injectDII32	17/11/2016 21:43 30/03/2017 11:06	Carpeta de archivos 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:



As discussed in the <u>DevCentral</u> 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 a0gg	; "ogg"
dd offset aJpeg	; "jpeg"
dd offset aJpg	; "jpg"
dd offset aPng	; "png"
dd offset aBmp	; "bmp"
dd offset aG1f	; "gı+"
dd offset aTiff	; "tiff"
dd offset alco	; "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:

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.



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:

	sdu Febru Buenes	Information
rea	eax, [eop+rroces: edx	Informationj
lea	eax, [ebp+Startu	pInfo]
push	eax	; lpStartupInfo
push	0	; 1pCurrentDirectory
push	0	; 1pEnvironment
push	4	; dwCreationFlags
pusn	0 0	, DINNErILHANUIES InThreadAttributes
push	0	10ProcessAttributes
push	ebx	, 1pCommandLine
push	0	; lpApplicationName
call	ds: <mark>CreateProcess</mark>	4
test	eax, eax	
JZ	short loc_3DBB88	
	a 🔝 1000014_win32_Unpa	2832
	svchost.exe	3804

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, WCh	
push	<mark>esi</mark>	; lpName
push	<mark>esi</mark>	; bInitialState
push	<mark>esi</mark>	; bManualReset
push	<mark>esi</mark>	; lpEventAttributes
nov	esi, ds:CreateEve	entW
call	esi ; CreateEvent	EW
push	0	; lpName
push	0	; bInitialState
push	0	; bManualReset
push	0	; lpEventAttributes
nov	[edi+6Ch], eax	•
call	esi ; CreateEvent	EW
push	0	; lpName
push	1	; bInitialState
push	1	bManualReset
push	0	; lpEventAttributes
nov	[edi+70h], eax	•
call	esi ; CreateEvent	EW
mov	ecx, [ebp+hThread	1]

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:



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.

🗾 🖆 🖼	
loc_3DCC18:	
cmp edx, 9	· marketel
jnz short 1	oc_3DCC54
) 🛄 🛃 🖼	
mov edx, [eax+28h]
push edx	
mov edx, [eax+24h]
mov ecx,	eax]
push edx	
mov edx, [[eax+20h]
push edx	
mov edx, [[eax+1Ch]
push edx	
mov edx, [eax+18h]
push edx	
mov edx,	[eax+14h]
push edx	
mov edx,	[eax+10h]
push edx	
mov edx, [[eax+0Ch]
mov eax,	[eax+8]
push edx	
push eax	
call ecx	
mov [esi+4	44h], eax
mov ecx, 1	
jmp short	10C_3DCC54

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 the second s	
debug006:000800F7 mov	[esi+48h], ecx	EAX 0000000 4
debug006:000800FA mov	ecx, [esi+4]	
debug006:000800FD push	ecx 🧲	
debug006:000800FE mov	dword ptr [esi+20h], 0	ECA 0000008 SID del evento para notificar a Trickbot
debuq006:00080105 call	edx 🧲	EDX 770F16DD 🖌 kernel32.dll:kernel32 ResetEvent
debug006:00080107 mov	eax, [esi+4]	ESI 000D0000 🖌 debug009:unk_D0000
debug006:0008010A mov	ecx, [esi]	EDI 0000000 4
debug006:0008010C mov	edx, [esi+0Ch]	FBP 8816FAFC IN Stack[88888FF81:8816FAFC
debug006:0008010F push	0	
debug006:00080111 push	ØFFFFFFFh	ESP 0010FHF4 SCACK[00000EE0]:0010FHF4
debug006:00080113 push	eax	EIP 00080105 🖌 debug006:00080105
debug006:00080114	ecx	EFL 00000293
debug006:00080115 call	edx	
debug006:00080117 jmp	loc_80007	
d-L		1

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:

debug006:000800F7 mov debug006:000800FA mov debug006:000800FA mov debug006:000800FE mov debug006:000800FE mov debug006:00080107 mov debug006:00080107 mov	<pre>[esi+48h], ecx ecx, [esi+4] ecx dword ptr [esi+20h], edx edx eax, [esi+4] ecx [esi+4] ecx [esi+4]</pre>	EAX 00000008 + EBX 7EFDE 000 + debug016:7EFDE 000 - IDs de Eventos ECX 00000004 + EDX 7710F8A4 + kernel32.dll:kernel32 SignalObjectAndWait ESI 00000000 + debug009:unk_D0000
debug006:000800FD push debug006:000800FE mov debug006:00080105 call debug006:00080107 mov debug006:00080107 mov debug006:0008010F push debug006:00080111 push debug006:00080113 push debug006:00080114 push debug006:00080117 call debug006:00080117 jmp	ecx dword ptr [esi+20h], 0 edx eax, [esi+4] ecx, [esi] edx, [esi+60n] 0 0FFFFFFFFFF eax ecx edx loc 80007	ECX 00000000
debug006:0008011C ;		

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:

	00252104	;	
-	00252104	рор	eax Offset de los datos inyectados
1	00252105	push	offset unk_90000
	0025210A	push	eax Offset de la función inyectada
	0025210B	jmp	near ptr unk_80000
	00252110	;	
1	00252110	call	loc_251D8A
	00252115	xor	ebx, ebx
1	00252117	MOV	[ebp+var_4], ebx
1	0025211A	MOV	eax, large fs:18h
	00252120	MOV	esi, [eax+4]
	00252123	MOV	[ebp+var_1C], ebx
	00252126	MOV	edi, offset unk_255080
	00050400		

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:

		00		11							200					June	Concoo soo y	10 10		10 10
4 0x	10000	000		1	Priva	ate					532	kВ	RW)	X	_			532 kB		532 kB
	0x10	0000	00		Priva	ate: (Com	nit			4	kВ	R		N	hot	ulo maneado en	4 kB		4 kB
	0x10	0010	00	1	Priva	ate: (Com	nit			72	kВ	RX				aria por	72 kB		72 kB
	0x10	0130	00	1	Priva	ate: (Com	nit			28	kВ	R			ien		28 kB		28 kB
	0x10	01a0	00	1	Priva	ate:	Com	nit			416	kВ	RW		S	ecci	ones	416 kB		416 kB
	0x10	0820	00	1	Priva	ate: (Com	nit			12	kВ	R					12 kB		12 kB
svchost.	exe (2172) (0>	100	0000	0 - (0x10	0010	000)										×	8 kB 8 kB
													Ca	bec	era	MZ	del modulo			12 kB
0000000) 4d	5a	90	00	03	00	00	00	04	00	00	00	ff	ff	00	00	MZ		~	12 kB
0000010) b8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00			=	8 LB
0000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00				O LD
0000030	00	00	00	00	00	00	00	00	00	00	00	00	fO	00	00	00				O KD
0000040) 0e	1f	ba	0e	00	b4	09	cd	21	b 8	01	4c	cd	21	54	68	!L.!Th			8 kB
0000050	69	73	20	70	72	6f	67	72	61	6d	20	63	61	6e	6e	6f	is program canno			8 kB
0000060	74	20	62	65	20	72	75	6e	20	69	6e	20	44	4f	53	20	t be run in DOS			28 kB
0000070) 6d	6f	64	65	2e	0d	0d	0a	24	00	00	00	00	00	00	00	mode\$			
0000080	28	43	45	6f	6c	22	2b	3c	6c	22	2b	3c	6c	22	2b	3c	(CEol"+<1"+<1"+<			
0000090	61	70	ca	3c	47	22	2b	3c	61	70	f4	3c	76	22	2b	3c	ap. <g"+<ap.<v"+<< td=""><td></td><td></td><td></td></g"+<ap.<v"+<<>			
00000a0	61	70	cb	3c	19	22	2b	3c	b1	dd	e0	3c	65	22	2b	3c	ap.<."+< <e"+<< td=""><td></td><td></td><td></td></e"+<<>			
			-	-				-				-				-				

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.

▲ 0x30000 0x30	0000			Priv Priv	ate ate:	Com	mit			4	kB kB	RW RW	x x				
svchost.e	xe (i	1136) (0)	300	00 -	0x31	.000)									
000000000000000000000000000000000000000	53 00 f8	75 00 03	63 43 00	63 02 00	65 38 60	73 ee 9d	73 43 43	00 02 02	00 00 a0	00 00 f4	00 00 18	00	08 30 00	00 04 00	07 04 43	0f 30 02	<u>Success</u>

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
0000001	00002190	0000	0000517F	Control
0000002	00002230	0001	00005187	FreeBuffer
0000003	00002180	0002	00005192	Release
0000004	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.





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, <O1 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.



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.								
4	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/								
10	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]/								
20	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 injectDII Report								
	URI	-								
64	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.



In the great work done by <u>malwarebytes</u> (<u>@hasherezade</u>) 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, <u>@hasherezade</u> 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:



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:

.text:003D # 7DA	push	eax ; hResInfo	FAX 883E5868	14
.text:003D <mark>8</mark> 7DB	push	esi ; hModule		
.text:003D87DC	call	ds:LoadResource ; kernelbase LoadResource()		7
.text:003D87E2	EAX:	0x003e5060 ("hECS30 M.s7J.;#L.T AWA.Xup8")	ECX 75021333	4
.text:003D87E2	test	eax, eax	EDX 003E8000	4
.text:003D87E4	jz	short loc_3D87FD	ESI 00000000	ų,
.text:003D87E6	push	eax ; hResData	EDT GAGAGAGA	6
.text:003D87E7	arg	00: 0x003e5060 ("hECS30 M.s7J.;#L.T AWA.Xyp.")		
.text:003D87E7	call	ds:LockResource ; kernelbase LockResource()	CDF 0010F930	1
.text:003D87ED	EAX:	0x003e5060 ("hECS30 M.s7J.;#L.T AWA.Xyp8")	ESP 0018F92C	4
.text:003D87ED	s ar	q 00: 0x003e5060 ("hECS30 M.s7J.;#L.T AWA.Xy")	EIP 003D87E2	4
.text:003D87ED	mov	esi, eax	EFL 00000244	

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):

 6		64	P	15							
Offset	0 1	2 3	4 5	56	7 8	9	A B	C D	Е	F	Ascii
00000000 00000010 00000030 00000030 00000040 0000050 0000060	68 00 20 4D 4A C0 7C 41 1C 5E F3 B4 B5 44	00 00 F0 73 3B C6 57 EB E2 7A 67 63 D7 E1	45 43 37 B 00 IF 0B B D1 29 D3 50 38 79	3 53 5 FB 7 23 1 7 7F 9 FB 1 9 F5 9 9 3E	33 30 18 B0 EF 1C 41 A1 B6 55 5B 5F 63 2B	00 0 C0 A 4C 0 58 7 41 D D1 C 03 2	0 00 F 80 6 54 9 70 5 8E 0 56 E C8	F3 20 BB F3 A3 8F OD C3 C7 C7 B8 38	86 FB A6 A1 3E 87	DB F1 38 1E DB	hECS306.10 .M8s7µ01 *A 1>600 JA;Æ. *I L-T£ + AWës% AlXyp.Ai8 ^azN)04U0A01ÇÇ> 6'gc0P8[_NAV,810 µDxá8y>c+L.E

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:

	Breakpoints	IDA View-A, Hex View-2			🖉 General registers		8 ×	ŝ
	DA WernA .text:003DAF5C push esi .text:003DAF5D push edx .text:003DAF5D push edx .text:003DAF5D push edx .text:003DAF5D push ecx .text:003DAF6D push ecx .text:003DAF6D push esi .text:003DAF6D push edx .text:003DAF6D push edx .text:003DAF6D push edx .text:003DAF6D push edx	; _D(; _D(; _D) [ebp+var_8] ; _D([ebp+var_8] ; _D(et aEccpublicblob ; "E(; _D) ×00314a38 ("QUUU	e INORD INORD INORD INORD INORD INORD INORD INORD INORD INORD INORD INORD	*	EAX 004890E8 w debug022:004890E8 EBX 00480480 w debug022:00406610 ECX 00486480 w Stack[00000800]:0018F880 EDX 00486480 w debug022:00406058 EDX 00486658 w debug022:00406058 EDX 00486658 w debug022:00406058 EDX 00486658 w debug022:00406058 EDX 00486658 w debug022:00406058 EDX 00486678 w debug021:00393508 EDX 004958644 stack[00000800]:0018F884 EDY 0018F866 w stack[00000800]:0018F864 EIP 0018F866 w stack[00000800]:0018F868 EIP 00395760 w dub_3D4E00+90 EFL 00000246		OF C DF C IF 1 TF C SF C ZF 1 AF C PF 1 CF C	1 3 3 1 3 1 3
IP	.text:003D4F60 arg_08: 0 .text:003D4F60 arg_02: 0 .text:003D4F60 arg_02: 0 .text:003D4F60 call dwor 00004F5C 003D4F5C: sub_3D4ED0+8 <	x003c149c -> 0x00430045 (" x001868a0 ("%=.HK1F x00320aa8 ("ECS30") d_3E47E8 ; bci c	s	•	Stack view 00187866 00000000 00187866 00000000 00187866 00000000 00187867 00301000 00187867 0031000 00187867 0031000 00187867 00318700 00187867 00187800 00187867 00187800 00187867 00180000 00187878 00180000 00187878 00180000 00187878 00180000 00187878 00180000 00187878 00180000 00187878 00180000	888	6 ×	A CONTRACTOR OF A CONTRACTOR OFTA CONTRACTOR O
0 184 184 184 184 184 184 184	Hex New-2 Solution Solution	80 F3 20 86 DB 20 4D F0 73 ECS2 80 BB F3 FB F1 4A C0 3B C6 7.51 54 A3 8F A6 19 72 41 57 E0 # 70 00 C3 138 15 EE 27 # 7 70 00 C3 138 15 EE 27 # 7 70 00 C3 8F 70 85 44 07 61 EP 60 E A7 7E 28 84 67 63 0 15 60 88 87 08 85 44 07 15 EP 15 68 88 88 88 88 88 88 88 88 88 88 88 88 89 89 84<		×	06187830 00000000 07187834 00486188 debug022:00486188 07187883 00486058 debug022:00486058 07187883 00000030 0000030 07187883 00000030 0000030 07187883 00000030 0000030 07187883 00000030 0000030 07187894 00000030 00000030 07187895 00000000 00000030 07187897 00000000 00000000 07187898 00000000 00000000 07187890 00000000 0018658 07187884 00000000 00186858 07187880 00000000 00186858 07187848 00000000 00186858 07187848 00000000 00186858 07187848 00000000 00186858 07187846 00000000 00186858	with	FSPI	

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

003D4FAD	arg_00: 0x002f74e8 ("RUUU8J1H1d\d./	Xd./.")
003D4FAD	arg_04: 0x00000000 ("N/A")	
003D4FAD	arg_08: 0x00314a00 ("A].p\u%]0WaRbUsA!\(]m4w	Q")
003D4FAD	arg_0c: 0x00000030 ("N/A")	
003D4FAD	arg_10: 0x0 <mark>0314d50 (''' w` oR D 5.1 t o .1 * rb] :30 r '</mark>	XnL.K''')
003D4FAD	<pre>call dword_3E47EC ; bcrypt_BCryptVerifySignature()</pre>	
003D4FB3	EAX: 0xc000a000 (H/H)	
003D4FB3	s_arg_00: 0x002f74e8 ("RUUU8J1H1d\d./	Xd.")
003D4FB3	s_arg_04: 0x00000000 ("N/A")	
003D4FB3		.⊎)
003D4FB3	s_arg_0c: 0x00000030 ("N/A")	
003D4FB3	cmp eax, 0C000A000h	

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():

.text:003D53F7 push ed	i	×	EAX 0018F898 🗣 Stack[000005A41:0018F898
.text:003D53F8 lea ec:	x, [ebx+8]		EDV 0000000
.text:003D53FB mov dw	word ptr [ebp+ <mark>pbData</mark>], 208h		
.text:003D5402 mov [e	bp+var 3C], 6610h		ECX 0018F85C • STACK[000005H4]:0018F85C
.text:003D5409 mov [e	bp+var 38], 20h		EDX 88526448 🖌 debug822:88526448
.text:003D5410 lea ed	i, [ebp+var_34]		ESI 00528FF0 🖌 debuq022:00528FF0
.text:003D5413 rep movsd			
.text:003D5415 lea ea	x, [ebp+phKey]		
.text:003D5418 push ea:	ix ;	phKey	CDF 0010F090 Stack[000003H4].0010F090
.text:003D5419 mov ed	li, 1		ESP 0018F838 🗣 Stack[000005A4]:0018F838
.text:003D541E push ed	i ;	dwFlags	EIP 003D5427 😽 decrypt+67
.text:003D541F push eb:	x	hPubKey	EFL 00000202
.text:003D5420 push 20	h i	dwDataLen	
.text:003D5422 lea ec	x, [ebp+ <mark>pbData</mark>]		Stack vew
.text:003D5425 push ec:	x ;	pbData 🔨	
.text:003D5426 push ed	x ;	hProv	
.text:003D5427 call ds	CryptImportKey ;	sub_751D904C()	
.text:003D542D EAX: 0x	:00000001 ("N/A")		0018F80 00000020
.text:003D542D test ea	ix, eax		0018F801 F4025716
.text:003D542F jz sh	ort loc_3D54AC		0010F00 593394FH
.text:003D5431 mov ec:	x, [ebp+phKey]		0010107 5390H795
.text:003D5434 mov es	i, ds:CryptSetKeyParam		00101070 2000070
.text:003D543A push eb:	x ;	dwFlags	0010F07 11220776
.text:003D543B lea ea	x, [ebp+var_14]		0010F07 69077090
.text:003D543E push ea:	ix ;	pbData	
text:003D543E_nush4		duParan	9010L00 30304000

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.



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:

View-A, Breakpoints, Pseudocode-A	, General registers, Stack view, Hex View-1 🔀	🖪 Structures 💽	E	Enums	
IDA View-A 🔯	👔 Breakpoints 🔝 💽	Pseudocode-A		🖉 General registers 🛛	ð ×
.text:003088F3 loc_30E .text:003088F6 mov .text:003088F6 mov .text:003088F8 mov .text:003088F8 push .text:003088F0 push .text:003088F0 push .text:00308909 mov .text:00308904 mov .text:00308904 mov .text:00308904 mov .text:00308904 push .text:00308908 push .text:00308908 push	<pre>BF3: eax, [ebp+Buffer] edx, [ebx] edx, [ebp+arg_0] edx eax eax ebx, 4 ebx, 4 [ebp+var_10], eax eax, [es1+48h] ecx eax eax</pre>	; CODE XREF: leer_memoria_1+208 ; lpHumberOfBytesRead ; nSize ; lpBuffer ; lpBaseAddress ; hProcess	tj 🏠	EAX 00000300 % EAX 00016920 % debug067:00916890 EXX 00236676 % EXX 00236676 % EXX 0023678 % debug07:0003498 EJT 756EFC6 % kernel32.011:kernel32.ReadProcessHemory EBP 0018F284 % Stack[00000900]:0018F284 ESP 0018F284 % Stack[00000900]:0018F284 ESP 0018F294 % Lack[00000900]:0018F244 ESP 0018F294 % Lack[00000900]:0018F244 ESP 0018F294 % Lack[00000900]:0018F244 ESP 0018F294 % Lack[00000900]:0018F244 ESP 0018F294 % Lack[00000900]:0018F244	0F 0 DF 0 IF 1 TF 0 SF 0 ZF 0 PF 1 CF 0
.text:003DB910 call	edi : ReadProcessMemory	; Leemos el resultado getsystem	info del pr	Stack view	đ×
<pre>text:003DB912 test .text:003DB914 jz .text:003DB916 mov .text:003DB916 cmp .text:003DB91C jz .text:003DB91E loc_3DE .text:003DB91E loc_3DE .text:003DB91E mov</pre>	eax, eax short loc_3DB91E eax, [ebp+var_10] [ebp+arg_0], eax short loc_3DB927 191E: [ebp+var_4], 0	; CODE XREF: leer_memoria_1+234	tj	ODCUCCEDT D01572/d 00286670 001572/d 00286670 001572/d 00286670 001572/d 00066220 00157250 00066220 00157254 00167250 00157252 576D52D 00157252 00034398 00157254 00034398	
0000B910 003DB910: leer_m	memoria_1+230 (Synchronized with Hex	View-1)	+	UNKNOWN 0018F244: Stack[000009D0]:0018F244 (Synchronized with	ESP) -

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

eneral Statistic	s Performance Threads Token Module	s Memory	Environment	Handles	Comment
V Hide unname	d handles				
Туре	Name	Handle			
Thread	1000014_win32_Unpacked_custom	0x434			
Thread	1000014_win32_Unpacked_custom	0x41c			
Thread	1000014_win32_Unpacked_custom	0x404			
Thread	1000014_win32_Unpacked_custom	0x3f8			
Process	svchost.exe (2544)	0x3d0			
File	C:\Users\john\AppData\Roaming\Mi	0x37c			
Key	HKCU\Software\Policies\Microsoft\S	0x344			
Key	HKLM\SOFTWARE\Policies\Microsoft\	0x308			
Key	HKLM\SOFTWARE\Microsoft\Enterpri	0x2d4			
Key	HKCU	0x2d0			
Key	HKCU\Software\Microsoft\SystemCe	0x2c8			
Key	HKLM\SOFTWARE\Microsoft\Enterpri	0x2c4			
Key	HKCU	0x2c0			
Key	HKLM\SOFTWARE\Microsoft\System	0x2bc			
Key	HKCU\Software\Microsoft\SystemCe	0x2b8			
Key	HKCU\Software\Microsoft\SystemCe	0x2b4			
Key	HKLM\SOFTWARE\Microsoft\System	0x2b0			
Key	HKLM\SOFTWARE\Microsoft\Enterpri	0x2ac			
Key	HKLM\SOFTWARE\Microsoft\System	0x2a8			
Key	HKCU\Software\Microsoft\SystemCe	0x2a4			
Key	HKLM\SOFTWARE\Microsoft\System	0x2a0			
Key	HKLM\SOFTWARE\Microsoft\Enterpri	0x29c			
Кеу	HKCU	0x298			

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

en plugin_nost.exe	1432		15,52 IVID John-pc/John	
a 💦 idaq.exe	528	0,13	240,62 MB john-pc\john	The Interactive Disassembler
4 1000014_win32_Unpacked_custom_noverify_config_conf_4.exe	3640		3,73 MB john-pc\john	
svchost.exe	2544		1,5 MB john-pc\john	Proceso host para los servicios de Windows

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:

neral	Statistics	Perf	formance	Thre	eads	s T	oker	n N	1odu	les	Mer	mory	E	nviro	nme	nt	Han	dles	Co	mme	nt				
✔ Hide	free regior	ns																							
Base A	Address		Туре						Size	Pr	otec	t	Us	e								Total W	s	Private WS	5
⊳ 0x :	140000		Private						4 kB	R	N											4 ki	в	4 kB	
⊳ 0x	150000		Private						4 kB	R	ΝX											4 ki	в	4 kB	
⊳ 0x:	160000		Private					25	6 kB	R	N		Sta	ack (threa	ad 13	396)					16 ki	в	16 kB	
⊳ 0x	1a0000		Private						4 kB	R	ΝX											4 ki	в	4 kB	
⊳ 0x:	1b0000		Private						4 kB	R	ΝX											4 ki	в	4 kB	
⊳ 0x	1d0000	C		_												-			• •	_			-		_
⊳ 0x2	220000		💷 svcho	st.ex	(2	2544) (0x	230	000	- 0xí	2920	00)													
⊿ 0x3	230000																								
	0x230000		000566	fO	3c	73	79	73	74	65	6d	69	6e	66	6f	3e	0d	0a	3c	67	<syst< td=""><td>eminfo></td><td><g< td=""><td></td><td></td></g<></td></syst<>	eminfo>	<g< td=""><td></td><td></td></g<>		
	0x292000		000567	00	65	6e	65	72	61	6C	3e	0d	0a	3c	6f	73	3e	4d	69	63	enera	1> <os>M</os>	ic		
⊳ 0x3	330000		000567	10	72	6I 40	73	61	66	74	20	57	69	6e	64	6I 75	11	73	20	37	rosoi	t Windows	2		
⊳ 0x3	3e0000		000567	20	20 65	40 72	76	69	63	65	20	50	61	63	69 6h	20	31	20	36	34	ervio	e Pack 1	5 64		
⊳ 0x4	410000		000567	40	20	62	69	74	73	3c	2f	6f	73	3e	0d	0a	3c	63	70	75	bits	03 <0	nu		
⊳ 0x4	430000		000567	50	3e	49	6e	74	65	6c	28	52	29	20	43	6f	72	65	28	54	>Inte	1(R) Core	(T		
⊳ 0x4	440000		000567	60	4d	29	20	69	35	2d	35	32	30	30	55	20	43	50	55	20	M) i5	-5200U CP	Ū		
⊳ 0x	5d0000		000567	70	40	20	32	2e	32	30	47	48	7a	3c	2f	63	70	75	3e	0d	0 2.2	OGHz <td>>.</td> <td></td> <td></td>	>.		
⊳ 0x	760000		000567	80	0a	3c	72	61	6d	3e	37	36	37	20	4d	42	3c	2f	72	61	. <ram< td=""><td>i>767 MB<!--</td--><td>ra</td><td></td><td></td></td></ram<>	i>767 MB </td <td>ra</td> <td></td> <td></td>	ra		
⊳ 0x	1b80000		000567	90	6d	3e	0d	0a	3c	2f	67	65	6e	65	72	61	6c	3e	0d	0a	m><	/general>			
⊳ 0x	1c70000		000567	aD	30	75	73	65	72	73	3e	Ud	Ua C1	30	75	73	65	72	3e	41	<user< td=""><td>s><user< td=""><td>>A</td><td></td><td></td></user<></td></user<>	s> <user< td=""><td>>A</td><td></td><td></td></user<>	>A		
⊳ 0x	1d70000		000567	00	64	001 72	30	0e	09	13	75	72	65	04 72	01 3e	12	3C 6f	21	15	13	omini er>	/>ropersec.	uS eC		
⊳ 0x	1ee0000		000567	d0	72	6f	75	70	55	73	65	72	24	30	2f	75	73	65	72	3e	roupU	ser\$ <td>r></td> <td></td> <td></td>	r>		
⊳ 0x3	2220000		000567	e0	0d	0a	3c	75	73	65	72	3e	49	6e	76	69	74	61	64	6f	<us< td=""><td>er>Invita</td><td>do</td><td></td><td></td></us<>	er>Invita	do		
⊳ 0x	10000000		000567	fO	3c	2f	75	73	65	72	3e	0d	0a	3c	75	73	65	72	3e	6a	<td>r><user< td=""><td>>j</td><td></td><td></td></user<></td>	r> <user< td=""><td>>j</td><td></td><td></td></user<>	>j		
			000568	00	6f	68	6e	3c	2f	75	73	65	72	3e	0d	0a	3c	2f	75	73	ohn </td <td>user><!--</td--><td>us</td><td></td><td></td></td>	user> </td <td>us</td> <td></td> <td></td>	us		
			000568	10	65	72	73	3e	0d	0a	3c	69	6e	73	74	61	6c	6c	65	64	ers>.	. <install< td=""><td>ed</td><td></td><td></td></install<>	ed		
			000568	20	3e	0d	0a	3c	70	72	6f	67	72	61	6d	3e	41	64	64	72	> <p< td=""><td>rogram>Ad</td><td>dr</td><td></td><td></td></p<>	rogram>Ad	dr		

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.



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".

injectDII32_configs	06/04/2017 13:05	Carpeta de archivos	
injectDII32	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.



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	rule MALW systeminfo trickbot module :
{	Trojan
meta:	
author = "Marc Salinas @Bondey_m"	meta:
description = "Detects Trickbot Banking	author = "Marc Salinas @Bondey_m"
Trojan"	description = "Detects systeminfo
	module from Trickbot Trojan"
strings:	
<pre>\$str_trick_01 = "moduleconfig"</pre>	strings:
<pre>\$str_trick_02 = "Start"</pre>	
<pre>\$str_trick_03 = "Control"</pre>	<pre>\$str_systeminf_01 = "<program>"</program></pre>
<pre>\$str_trick_04 = "FreeBuffer"</pre>	<pre>\$str_systeminf_02 = "<service>"</service></pre>
<pre>\$str_trick_05 = "Release"</pre>	<pre>\$str_systeminf_03 = ""</pre>
	\$str_systeminf_04 =
condition:	"GetSystemInfo.pdb"
all of (\$str_trick_*)	<pre>\$str_systeminf_05 = ""</pre>
}	<pre>\$str_systeminf_06 = ""</pre>
	condition:
	all of (\$str_systeminf_*)
rule MALw_dllinject_trickbot_module : 1 rojan	rule MALw_mailsercher_trickbot_module :
	rojan
meta:	{ mate:
description – " Detects dllipiest module	nicia.
from Trickhot Trojan"	description – " Detects mailsonreber
Hom mekbor mojan	module from Trickhot Trojan"
strings	
str dllini 01 – "user pref("	strings:
\$str_dllini_0? = " <ignore_mask>"</ignore_mask>	\$ str mails $01 =$ "mailsearcher"
\$str_dllini_02 = " <require_header>"</require_header>	\$str_mails_0? = "handler"
$st_dlini_0 = "//dini>"$	$str_mails_02 = "conf"$
↓su_unij_0+ = \\u00eduij>	$str_mails_{04} = "ctl"$
condition:	str mails 05 = "SetConf"
all of (\$str. dllini *)	str mails 06 = "file"
}	str mails 07 = "needinfo"
	str mails 08 = "mailconf"
	condition:
	all of (\$str_mails_*)
	}



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.



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:

RouterOS v6.34.2	HIKVISION Lugie v
Webřig Login: Login: (admin Login Pessendi	
Windox Tahut Graphs Licence Help Ominos	
	delicional Optic Technolog Co., In: All type decared
4 unhandication Disquirind A variance and passions of an beau Variantization Logari	Ng mguarsid by http://34
User hallin:	

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:

Teachtal a season	
lombro común (CN)	h on that
	ta Areta
Inidad organizativa (OI	l) ret
lúmoro do corio	00/05/62/15/40/00/64/96/05
vulhero de serie	00.03.03.13.48.00.04.80.25
Emitido por	
Nombre común (CN)	rvgvtfdf
Organización (O)	tg4r6tds
Jnidad organizativa (Ol	J) rst
Periodo de validez	
Comienza el	08/06/16
Caduca el	08/06/17
1	
	24.04.00.57.00.01.00.50.70.04.04.52.20.57.50.45.
luella digital SHA-256	34:04:69:57:08:B1:08:F9:70:B4:04:E3:30:57:F8:4F:



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

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/enus/library/windows/desktop/ms682425%28v=vs.85%29.aspx

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

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

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







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

