Deep Analysis of TrickBot New Module pwgrab

By Xiaopeng Zhang

Published: 2018-11-08 · Archived: 2026-04-05 17:35:55 UTC

The TrickBot malware family has been live for several years, mainly focused on stealing victim's online banking information. In new samples recently collected by FortiGuard Labs, we found a new TrickBot variant, with a new module pwgrab, which attempt to steal credentials, autofill data, history and so on from browsers as well as several software applications. I did a deep analysis on this pwgrab module, and in this blog I will explain how it works on a victim's system.

TrickBot downloaded by opening an excel file

The new TrickBot variant is spread by an Excel file (originally named "Sep_report.xls") using a malicious Macro VBS code that is executed when the victim opens the file in Microsoft Excel. We captured this sample on October 19th, 2018. Figure 1 shows that "Sep_report.xls" is opened in Microsoft Excel where it requests that the victim enable the embedded Macro by clicking on the "Enable Content" button.

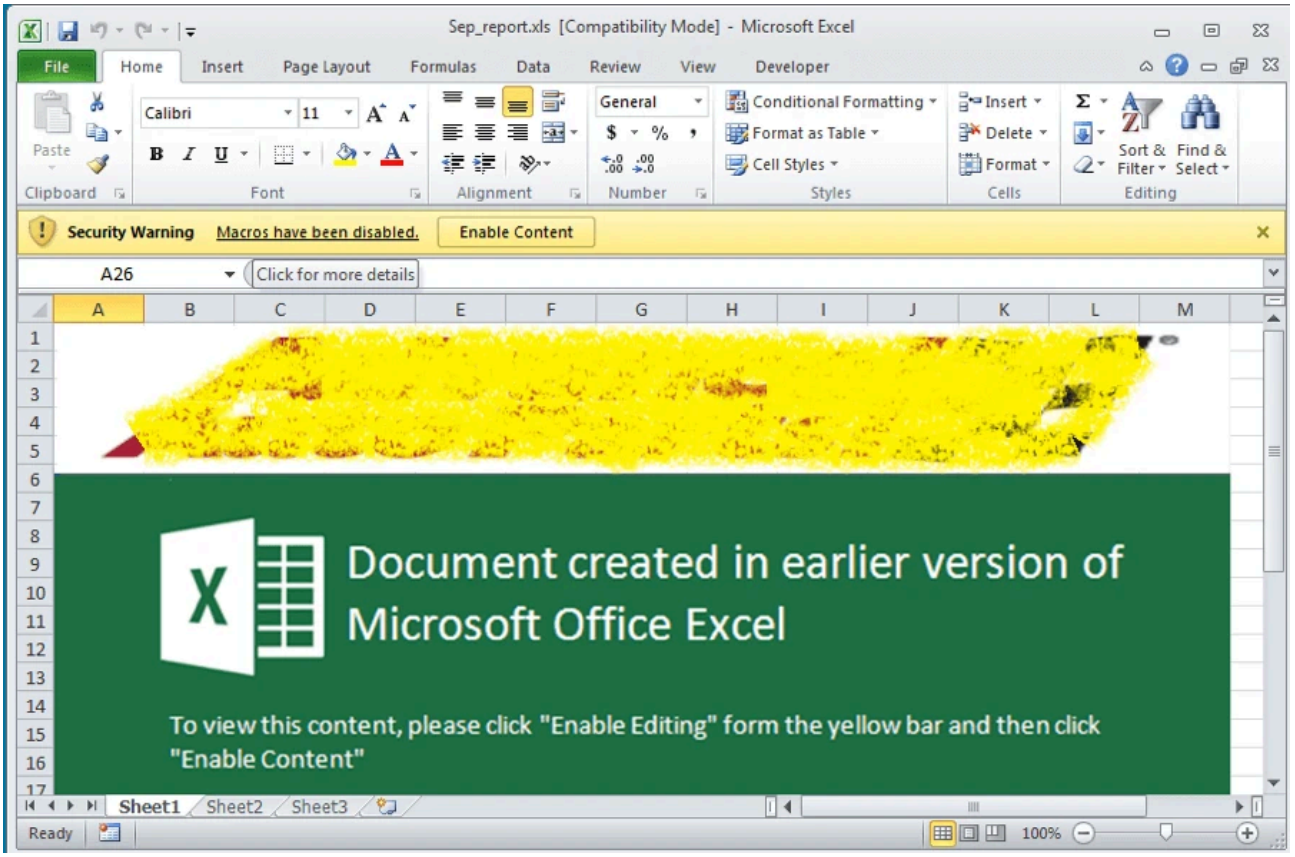

Figure 1. Sep_report.xls opened in Microsoft Excel

The VBA code is password protected for viewing. To analyze the code, I manually modified the protected flag to bypass the password protection.

The VBA code starts with the function “Workbook_Open”, which is called automatically when the Excel file is opened. It then reads data from Text control, which is encoded Powershell code. In Figure 2 you can see part of the decoded Powershell code.

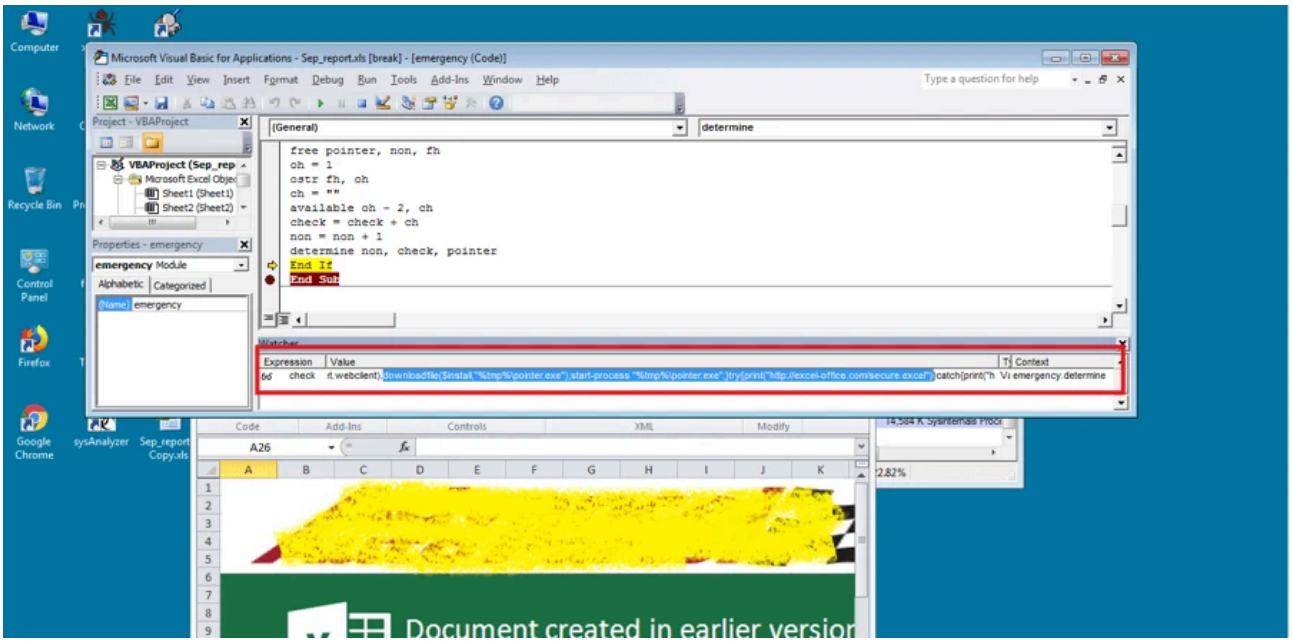

Figure 2. Decoded Powershell code

Finally, the Powershell code is executed to download the file from “*hxxp://excel-office.com/secure.excel*” and save it to a local temporary folder with the name “pointer.exe” whereupon it runs it. As you may have guessed, the “pointer.exe” file is actually TrickBot.

Task Schduler Starts TrickBot to load pwgrab32

When “pointer.exe” runs for the very first time, it creates the “%AppData%\VsCard” folder as its home folder, then copies “pointer.exe” into it and renames it as “pointes.exe”. In this version it also changes its module folder: the new one is “%AppData%\VsCard**Data**” instead of the previous “%AppData%\[random folder name]**Modules**”. Figure 3 is a screenshot of the new folder.

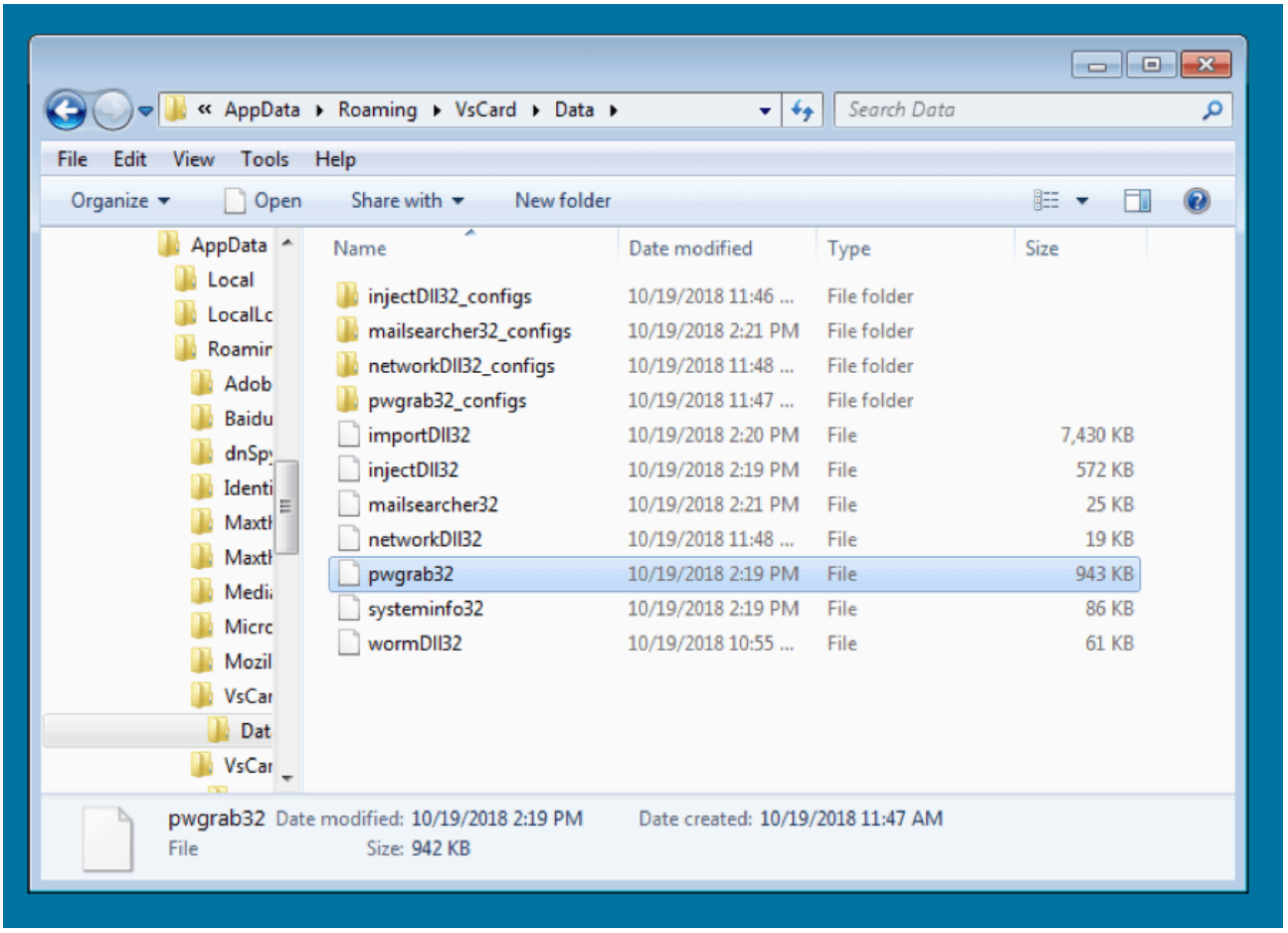

Figure 3. Screenshot of the new module folder “Data”

As with its previous version, it installs itself into the system “Task Scheduler” so it can run automatically by “Task Scheduler”.

After “pointes.exe” runs for a little while, it sends the command “5” request to its C&C server with the string “pwgrab32” for a 32-bit platform (or “pwgrab64” for a 64-bit platform) asking to download the new module of “pwgrab32”, just as it does for downloading other module files such as “systeminfo32” and “injectdll32”.

To learn more about the packet format of command “5” and the command’s purpose, you can refer to my previous blog. All files downloaded through command “5” in older versions are AES encrypted. Recent versions have added one more XOR encryption on AES encrypted data .So to get to the original pwgrab32 module we had to go through two-layer decryption. The pwgrab32 module was generated on October 16th, 2018 and was developed with Borland Delphi 3.0. Figure 4 shows the pwgrab32 module analyzed in a PE tool.

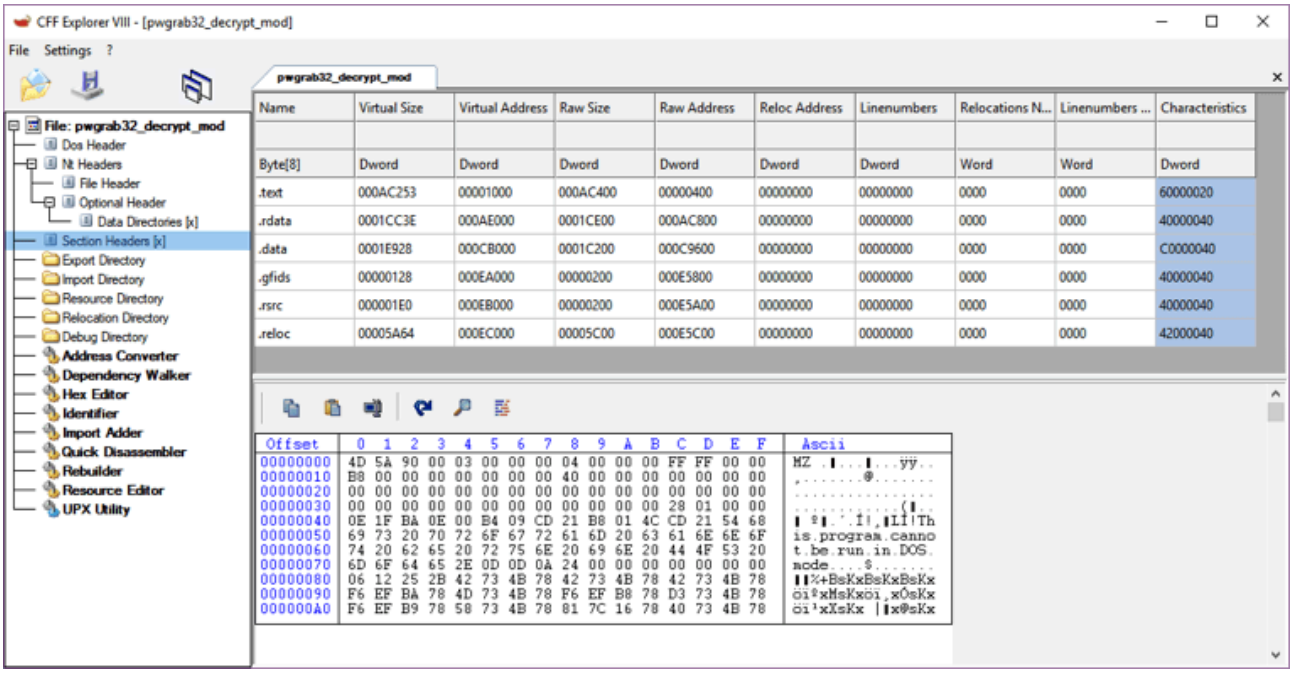

Figure 4. Decrypted pwgrab32 being analyzed in CFF Explorer

During my analysis of “pointes.exe” I can see that it uses some anti-analysis techniques to make it harder to be analyzed. For example, it encrypts all string information to protect itself from being analyzed statically and dynamically loads APIs during running time.

From the file name of “pwgrab32” we can guess it will grab password information from the victim’s system. Let’s go on to see how it will do this.

After downloading and decrypting “pwgrab32”, “pointes.exe” continues to load “pwgrab32”. Just like when loading other modules, it calls the API “CreateProcessAsUserW” to create a suspended “svchost.exe” process. It then injects a piece of code from the “pointes.exe” memory to this svchost.exe process memory by calling the API “WriteProcessMemory”. By calling the API “ZwQueryInformationProcess”, “pointes.exe” can get “svchost.exe”’s ProcessBasicInformation from which it can locate the OEP (Original Entry Point) of “svchost.exe” in its PE structure. Furthermore, it can modify the code at OEP to execute the copied piece of code. It then calls “ResumeThread” to resume running “svchost.exe”. Figure 5 is a code snippet of finding “svchost.exe”’s OEP.

```
:010F4FE7        push    eax             ; ;; returned ProcessInformation
:010F4FE8        mov     esi, ecx
:010F4FEA        mov     ecx, ds:APIsTable ; ;1118850, the old data is 0x74E250
:010F4FF0        mov     edx, [ecx+1F0h] ; will call=>ZwQueryInformationProcess
:010F4FF6        push    ebx             ; ; 0 means ProcessBasicInformation
:010F4FF7        push    edi             ; ;ProcessHandle
:010F4FF8        call    edx             ; ;; call ZwQueryInformationProcess
:010F4FFA        test    eax, eax
:010F4FFC        js      short loc_10F505B
:010F4FFE        mov     ecx, [ebp+var_24]
:010F5001        push    10h
:010F5003        lea     eax, [ebp+var_10]
:010F5006        push    eax
:010F5007        push    ecx
:010F5008        push    edi
:010F5009        mov     ecx, esi
:010F500B        call    sub_10F52C0     ; ;call=>ReadProcessMemory
:010F5010        test    eax, eax
:010F5012        jz      short loc_10F505B
:010F5014        mov     eax, [ebp+var_8] ; ;;; base address of svchost.exe
:010F5017        push    40h
:010F5019        lea     edx, [ebp+var_68]
:010F501C        push    edx
:010F501D        push    eax
:010F501E        push    edi
:010F501F        mov     ecx, esi        ; ;; read MZ  header.
:010F5021        call    sub_10F52C0     ; ;call=>ReadProcessMemory
:010F5026        test    eax, eax
:010F5028        jz      short loc_10F505B
:010F502A        mov     eax, [ebp+var_2C]
:010F502D        add     eax, [ebp+var_8]
:010F5030        push    0F8h
:010F5035        lea     ecx, [ebp+var_160]
:010F503B        push    ecx
:010F503C        push    eax
:010F503D        push    edi
:010F503E        mov     ecx, esi        ; ;;; PE structure.
:010F5040        call    sub_10F52C0     ; ;call=>ReadProcessMemory
:010F5045        test    eax, eax
:010F5047        jz      short loc_10F505B
:010F5049        mov     eax, [ebp+var_138] ; ;; OEP offset.
:010F504F        add     eax, [ebp+var_8] ; ;; conculate OEP of svchost.exe.
:010F5052        pop     edi
:010F5053        pop     esi
:010F5054        pop     ebx
:010F5055        mov     esp, ebp
:010F5057        pop     ebp
:010F5058        retn    4
```

Figure 5. Find svchost.exe OEP

Next, API “WriteProcessMemory”, “SignalObjectAndWait”, and “WaitForSingleObject” are called a number of times by both “pointes.exe” and “svchost.exe” to maintain synchronicity to finish copying the decrypted pwgrab32 and related information, such as copying the C&C server IP list from “pointes.exe” onto “svchost.exe”. Finally, “pwgrab32!10006634” (the OEP of pwgrab32) is called by the copied piece of code mentioned above.

From this point on, the pwgrab32 takes over the work to collect any password related data.

pwgrab32 Collects Credentials from Browsers of Victim’s System

At first “pwgrab32” decodes the “core-parser.dll” module , loads into the memory, and makes it ready for use. It has several export functions, as shown in Figure 6.

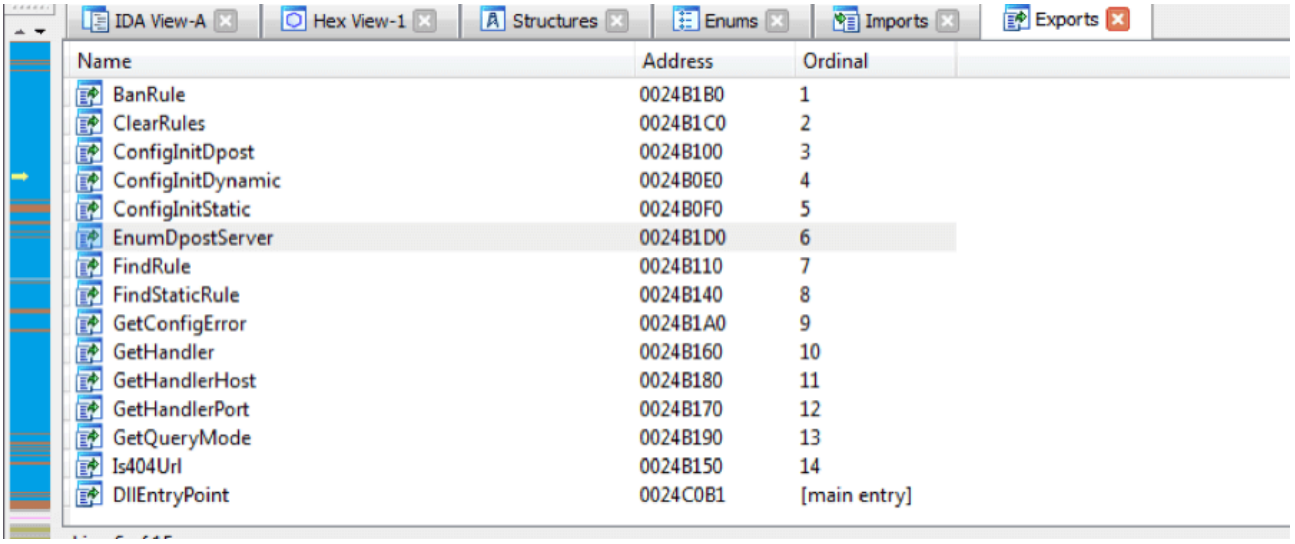

Figure 6. core-parser.dll export function list

Function “EnumDpostServer” returns the C&C server IP address, which will be called by “pwgrab32” when it wants to send data to the C&C server.

It launches three threads to grab credentials from three different browsers. They all share the same thread function but different parameters. From my analysis, parameter 1 is for Internet Explorer, 2 is for Firefox, 3 is for Chrome, and 4 is for Edge. However, in this version Edge is disabled.

There is also a very huge function, “pwgrab32!sub_100137F8”, which executes the operation of collecting saved credentials from all browsers. There are different code branches for different browsers. I will show you how it works.

One interesting thing I found in the “pwgrab32” code is that it encrypts plain text byte by byte, decrypts it back to plain text, and uses that decrypted plain text. Is this a joke by the Trickbot author? No, it should be an anti-analysis technique to hide plain text. However, I think the author simply forgot to remove the decryption function and replace the plain text with the encrypted one before compiling this module. This error appears many times throughout the pwgrab32 module. Figure 7 shows a code snippet of that.

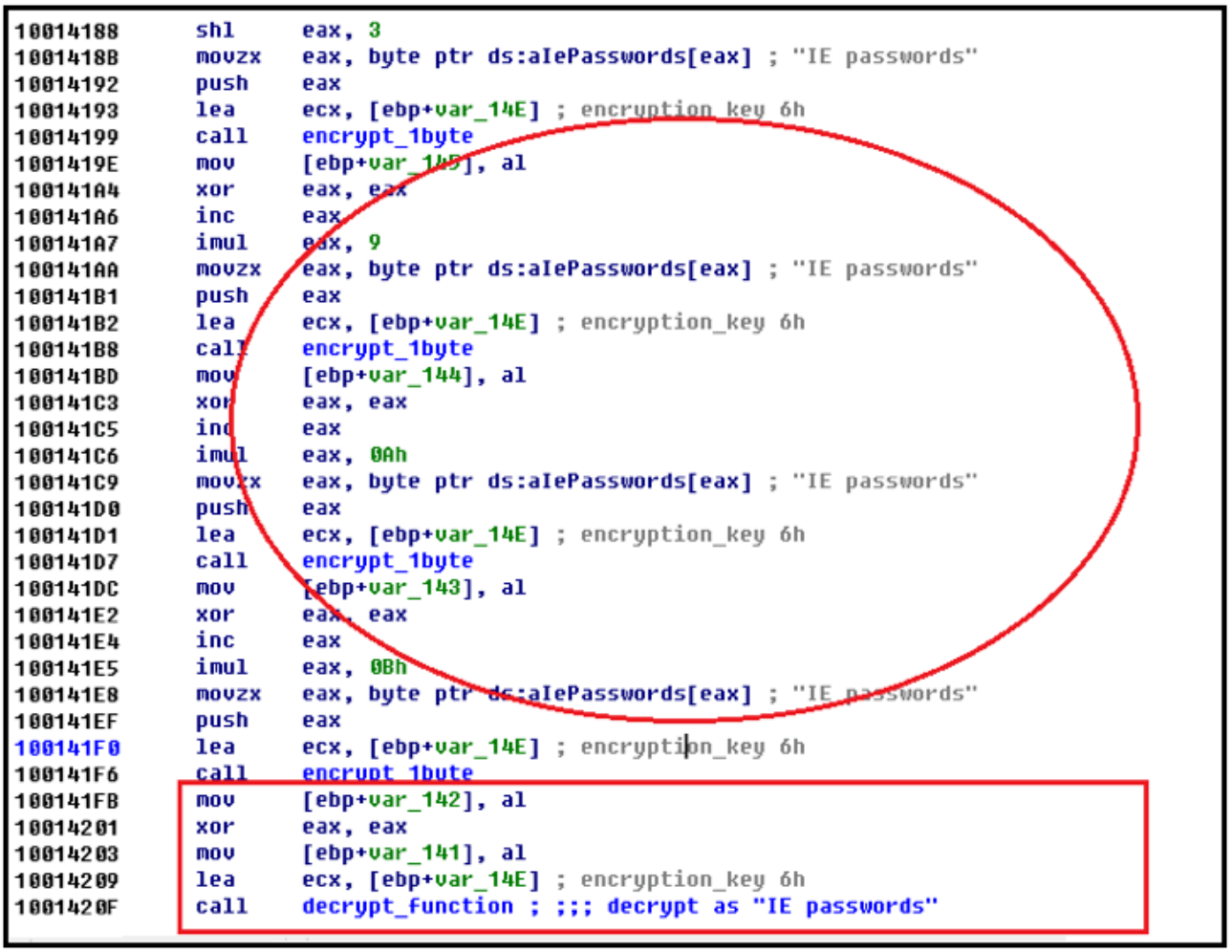

Figure 7. Plain text “IE password” being encrypted first and decrypted later

1> Thread Parameter 1 for Internet Explorer:

According to the Windows system version, there are two different code branches for IE.

If a victim’s system version is Windows 2000, Windows XP, Windows Vista, Windows Server 2008, Windows 7, Windows 10, or Windows Server 2016, it reads and enumerates values from the system registry sub-key “HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\IntelliForms\Storage2”, which contains the SHA1 hash code list of saved website hosts and saved credentials for this website. Figure 8 is a screenshot of the “Storage2” sub-key on my Windows 7 system. Calling the APIs “FindFirstUrlCacheEntryW” and “FindNextUrlCacheEntryW”, this malware can enumerate all cached websites. Furthermore it can calculate SHA1 hash code for each website host (for example “http://www.fortinet.com/”) through comparison with the hash code from the sub-key “Storage2”, whereupon it can obtain the website’s host. It then parses the third column data to get the credential for the website. Finally, it saves the collected credentials in this format:

“Website host|Login ID|Login password”.

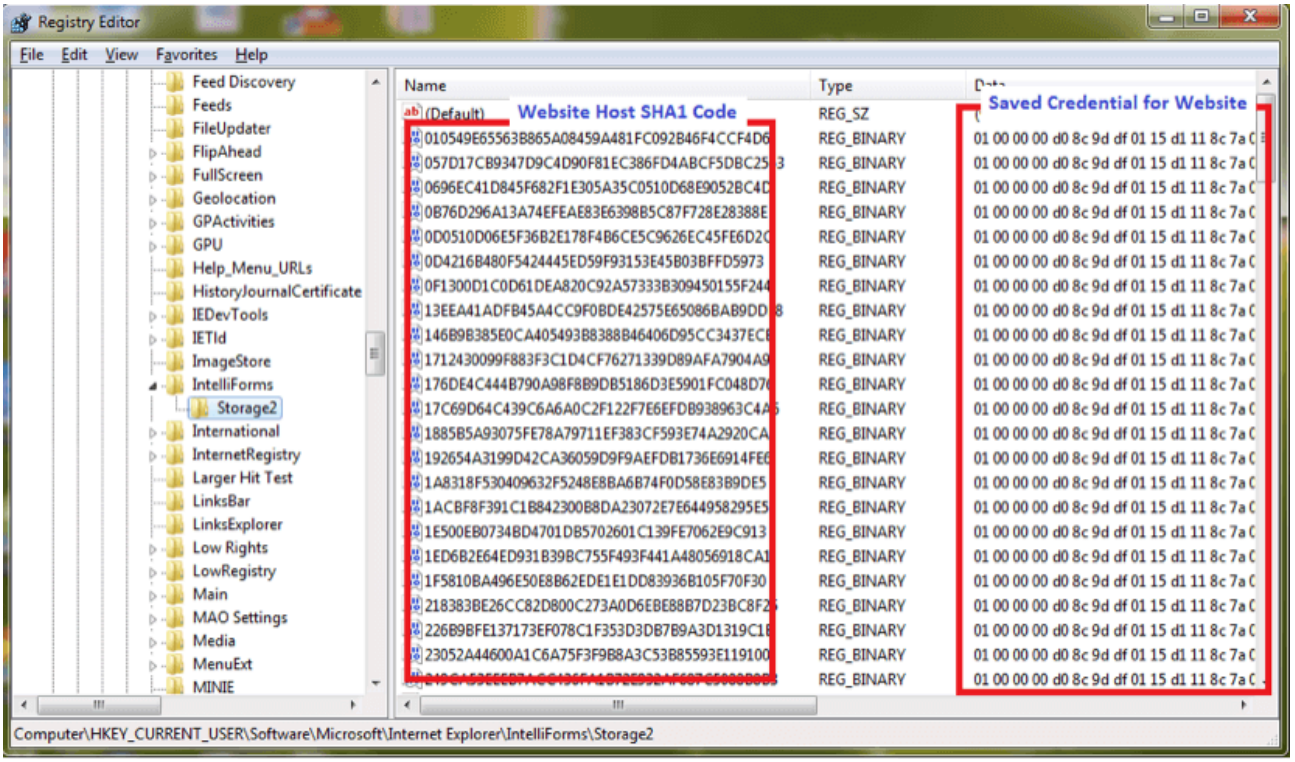

Figure 8. Screenshot of sub-key “Storage2”

When the victim’s system is another version, it calls some additional APIs to get credentials. Here is a pseudo code of this process for getting credentials.

```
if ( VaultEnumerateVaults(0, &a4, &a5) )
return 0;

  v70 = 0;  a2 = 0;
  if ( a4 )  {    v71 = 0;    a3 = 0;
while ( !VaultOpenVault(v71 + a5, 0, &vars0) && !VaultEnumerateItems(vars0, 512, &a1, &retaddr) )
  {       v72 = 0;
      if ( a1 )  {
        v73 = lpCriticalSection;
        v74 = a7;
        do {
          memset(&a65, 0, 0xE08u);
          if ( v74 )
            v75 = sub_100133F2(v72, &a65);
          else
            v75 = sub_1001329C(v72, &a65);
          if ( v75 ) {
            wnsprintfA(&String, 1024, "%S|%S|%S\n", &a66, &a67, &a68);
            v76 = sub_1000CBB8(&String);
            sub_1000D1AA(v73, &String, v76);
          }
```

```
          ++v72;
        }
        while ( v72 < a1 );
      v70 = a2;    v71 = a3;
    }
    VaultFree(retaddr);
    VaultCloseVault(vars0);
```

2> Thread Parameter 2 for Mozilla Firefox:

This code thread reads the Firefox installation path from the system registry, and then calls the API “SetCurrentDirectoryA” with the installation path to set the current directory to the Firefox installation path so it can easily read the credential files of Firefox and load a dll which is used to handle Firefox credentials.

“pwgrab32” continues to load nss3.dll of Firefox and read some Firefox files from its AppData folder, such as "%AppData%\Mozilla\Firefox\Profiles\e375zm7t.default\logins.json". It then calls the APIs of nss3.dll, like “PK11_GetInternalKeySlot”, “PK11_Authenticate”, and “PK11SDR_Decrypt” to parse saved credentials in the file, “logins.json”. Below is a piece of data from “logins.json”.

```
{"id":5,"hostname":"https://api.twitter.com","httpRealm":null,"formSubmitURL":"https://api.twitter.com","usern
```

Finally, it saves the credentials in a format like IE’s.

3> Thread Parameter 3 for Google Chrome:

Before the thread function is created, pwgrab32 makes two file backups of the files “Login Data” and “Web Data”. Both of them are located in the "%LocalAppData%\Google\Chrome\User Data\Default\” folder. Chrome stores the login credentials of the victim in the file “Login Data”, and saved autofill and credit card information is stored in the file “Web Data”. It makes a backup of the two files so it can read data from backup files instead of the original files to avoid a reading conflict when the victim is using Chrome. The two backup files are “Login Data.bak” and “Web Data.bak”. They are both SQLite database files.

“pwgrab32” uses the open source project SQLite database engine to handle the two SQLite files. In Figure 9, you can see that the data of the SQLite database engine is linked in “pwgrab32”.

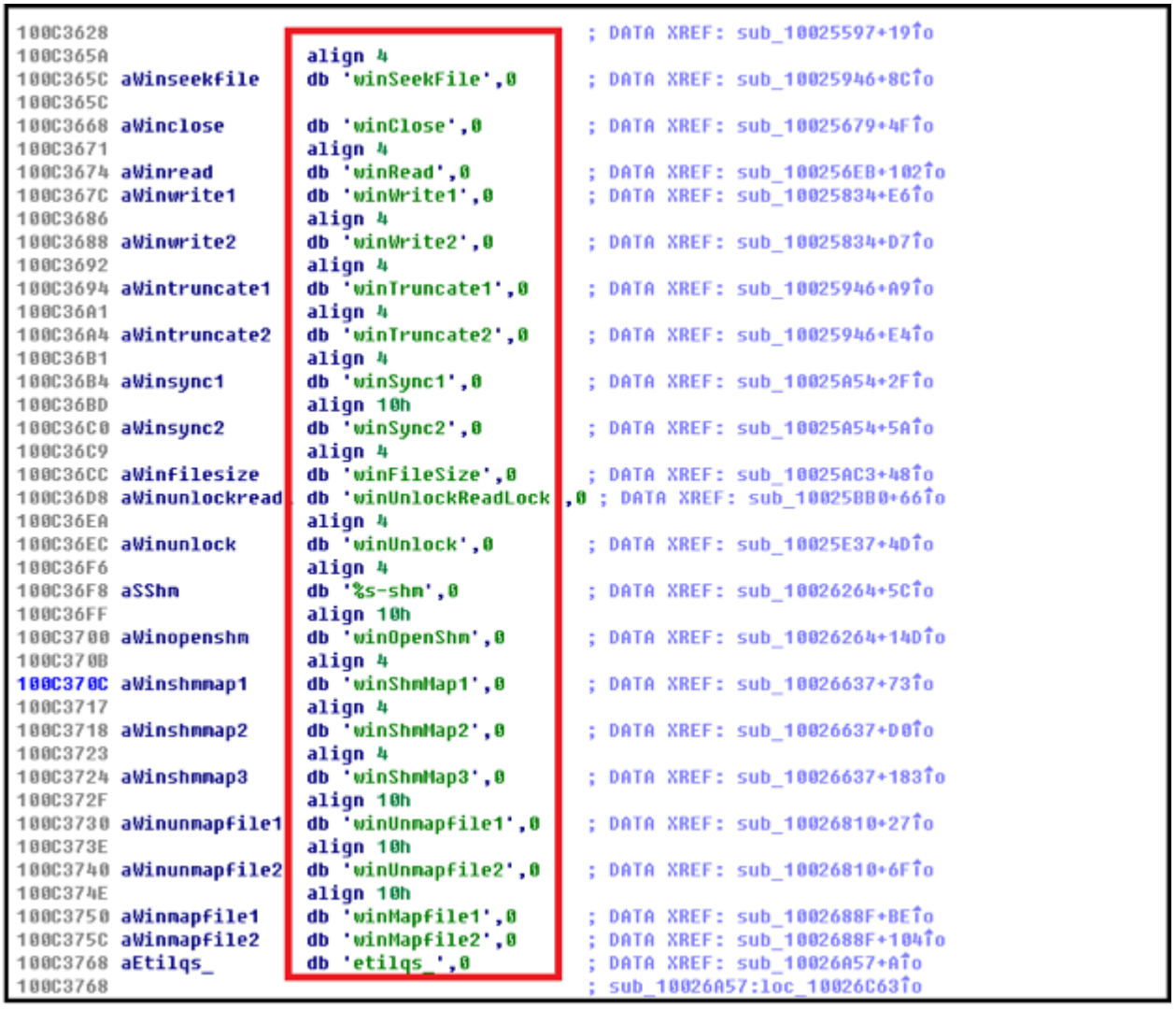

Figure 9. SQLite database engine's code

Next, "pwgrab32" executes an SQL expression like "*select origin_url, username_value, password_value, length(d_value) from logins where blacklisted_by_user = 0*" to obtain the credentials from "Login Data.bak".

"pwgrab32" continues to execute three SQL expressions to grab autofill information, credit card information, email address, country, company, street address, full name, phone number, etc. from "Web Data.bak".

The SQL expressions are decrypted from three local variables:

```
 "SELECT name, value FROM autofill WHERE name<>"cd[Meta]"" AND name<>"cd[OpenGraph]" AND name<>"cd[Schema.org]"

"SELECT expiration_month, expiration_year, card_number_encrypted, use_date, origin FROM credit_cards ORDER BY or

"SELECT profiles.origin, profiles.company_name, profiles.street_address, profiles.city, profiles.state, profiles
```

The grabbed credentials and form autofill information collected from the browsers is sent to the C&C server immediately when one is done.

Next, I'll discuss the packet format and how it's sent to its C&C server in the "Report Credentials" section below.

pwgrab32 Collects Credentials from some Clients

After all of the three threads above are finished, "pwgrab32" steals credentials from three client software sources: "Outlook", "FileZilla", and "WinSCP". In Figure 10 you can see the functions being called to collect credentials from them.

```
.text:100065E5                  lea     eax, [ebp+ThreadId]
.text:100065E8                  push    eax             ; lpThreadId
.text:100065E9                  push    ebx             ; dwCreationFlags
.text:100065EA                  push    1               ; lpParameter ;;;;;;;;;;;;; IE
.text:100065EC                  push    edi             ; lpStartAddress
.text:100065ED                  push    ebx             ; dwStackSize
.text:100065EE                  push    ebx             ; lpThreadAttributes
.text:100065EF                  call    esi ; CreateThread
.text:100065F1                  mov     ebx, eax
.text:100065F3                  lea     eax, [ebp+ThreadId]
.text:100065F6                  push    eax             ; lpThreadId
.text:100065F7                  xor     eax, eax
.text:100065F9                  push    eax             ; dwCreationFlags
.text:100065FA                  push    2               ; lpParameter ;;;;;;;;;;; FireFox
.text:100065FC                  push    edi             ; lpStartAddress
.text:100065FD                  push    eax             ; dwStackSize
.text:100065FE                  push    eax             ; lpThreadAttributes
.text:100065FF                  call    esi ; CreateThread
.text:10006601                  mov     esi, ds:__imp_WaitForSingleObject
.text:10006607                  mov     edi, eax
.text:10006609                  push    0FFFFFFFFh      ; dwMilliseconds
.text:1000660B                  push    [ebp+hHandle]   ;
.text:1000660E                  call    esi ; __imp_WaitForSingleObject
.text:10006610                  push    0FFFFFFFFh      ; dwMilliseconds
.text:10006612                  push    ebx             ;
.text:10006613                  call    esi ; __imp_WaitForSingleObject
.text:10006615                  push    0FFFFFFFFh      ; dwMilliseconds
.text:10006617                  push    edi             ;
.text:10006618                  call    esi ;   imp_WaitForSingleObject
.text:1000661A                  call    sub_10001D53    ; ;;;; Outlook  password
.text:1000661F                  call    sub_1000703E    ; ; FileZilla password
.text:10006624                  call    sub_10009D2F    ; ;WinSCP password
.text:10006629                  pop     edi
.text:1000662A                  pop     esi
.text:1000662B                  xor     eax, eax
.text:1000662D                  pop     ebx
.text:1000662E                  mov     esp, ebp
.text:10006630                  pop     ebp
.text:10006631                  retn    4
.text:10006631 Thread_fun       endp
.text:10006631
```

Figure 10. Functions to collect credential from Outlook, FileZilla and WinSCP

"Outlook"'s profile is stored in the system registry. According to different versions, its registry paths are "*HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Outlook*",

"*HKCU\Software\Microsoft\Office\15.0\Outlook\Profiles\Outlook*" and

"*HKCU\Software\Microsoft\Office\16.0\Outlook\Profiles\Outlook*".

"pwgrab32" then goes through all the keys and reads and parses the values to grab the credentials.

Figure 11 shows an Outlook credential grabbed by "pwgrab32" from my test system. The format is "Host|Account name|Password".

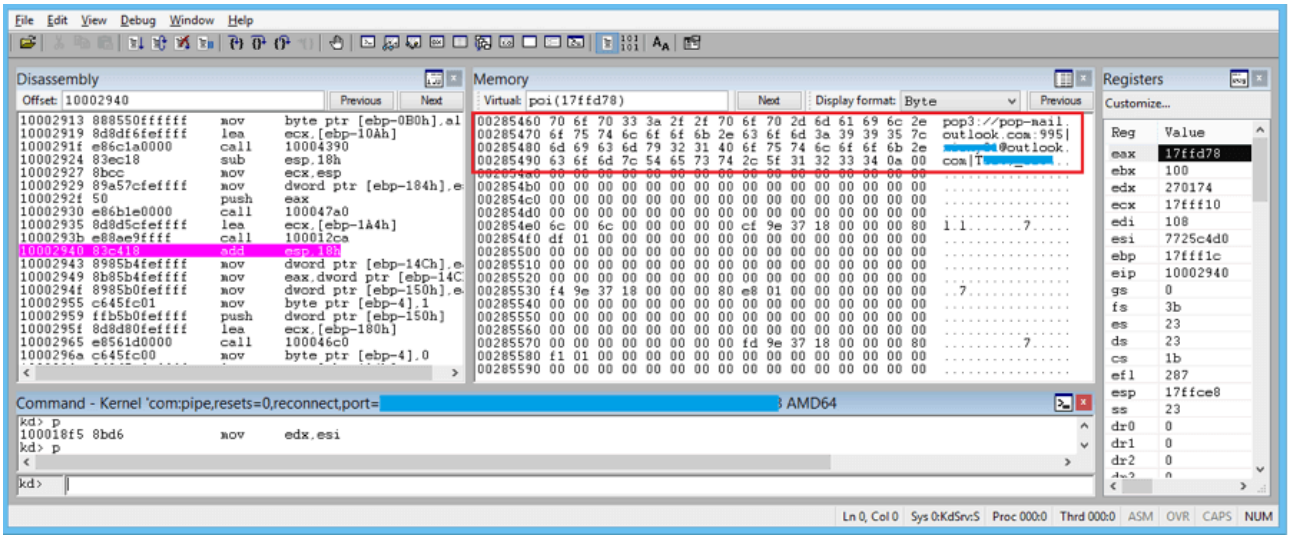

Figure 11. Grabbed credentials from Outlook

FileZilla is an FTP client software that stores its history as plain text in file "%APPDATA%\\filezilla\\recentservers.xml", and stores its login data as plain text in file "%APPDATA%\\filezilla\\sitemanager.xml". “pwgrab32” can easily obtain their history records and credentials by parsing these two XML files.

WinSCP is another FTP client software. Its credentials are stored in the system registry under the registry path “*HKCU\Software\Martin Prikryl\WinSCP 2\Sessions*”. “pwgrab32” can grab its credentials by enumerating all of the sub-keys and reading out their values “HostName”, “PortNumber”, “UserName”, “Password”, and “FSProtocol”.

Report Credentials

Trickbot has many C&C commands. I have talked about these commands in detail in my previous blog.

In module “pwgrab32”, however, I observed that it has new command numbers: 81 and 83.

Command 81 is for reporting grabbed credentials of Browsers, FTP clients, and Outlook.

Command 83 is for reporting grabbed form autofill information from Google Chrome.

It uses HTTP POST method to report the plain text credentials to the C&C server.

The POST URI format is like this:

POST /[group tag]/[Client_ID]/[Command number]/

The body part is the grabbed credentials or form autofill information in plain text.

“group tag” is “auto1”.

“Client_ID” is generated with the computer name, Windows version and random string.

Figure 12 shows reporting “chrome password” using command 81.

Figure 13 shows reporting “chrome autofill information” using command 83.

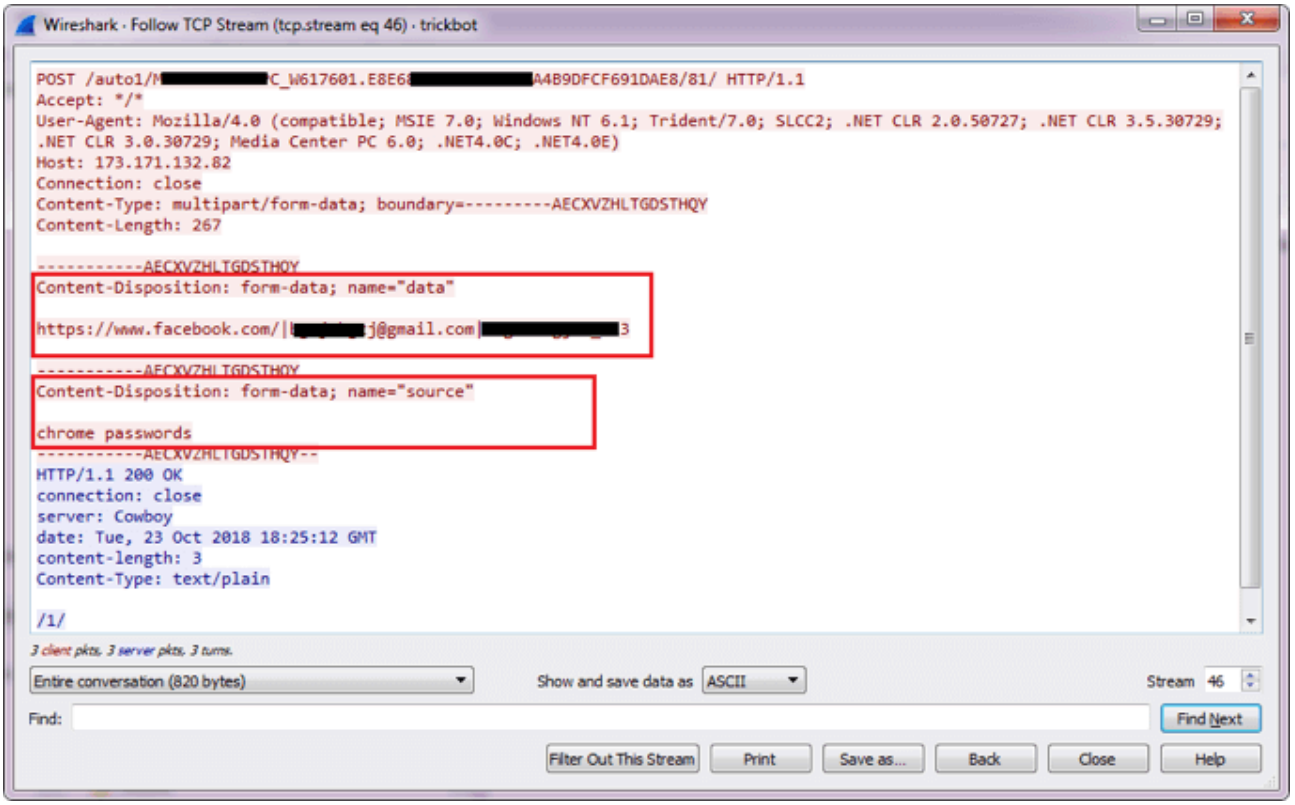

Figure 12. Report grabbed credential data from Chrome

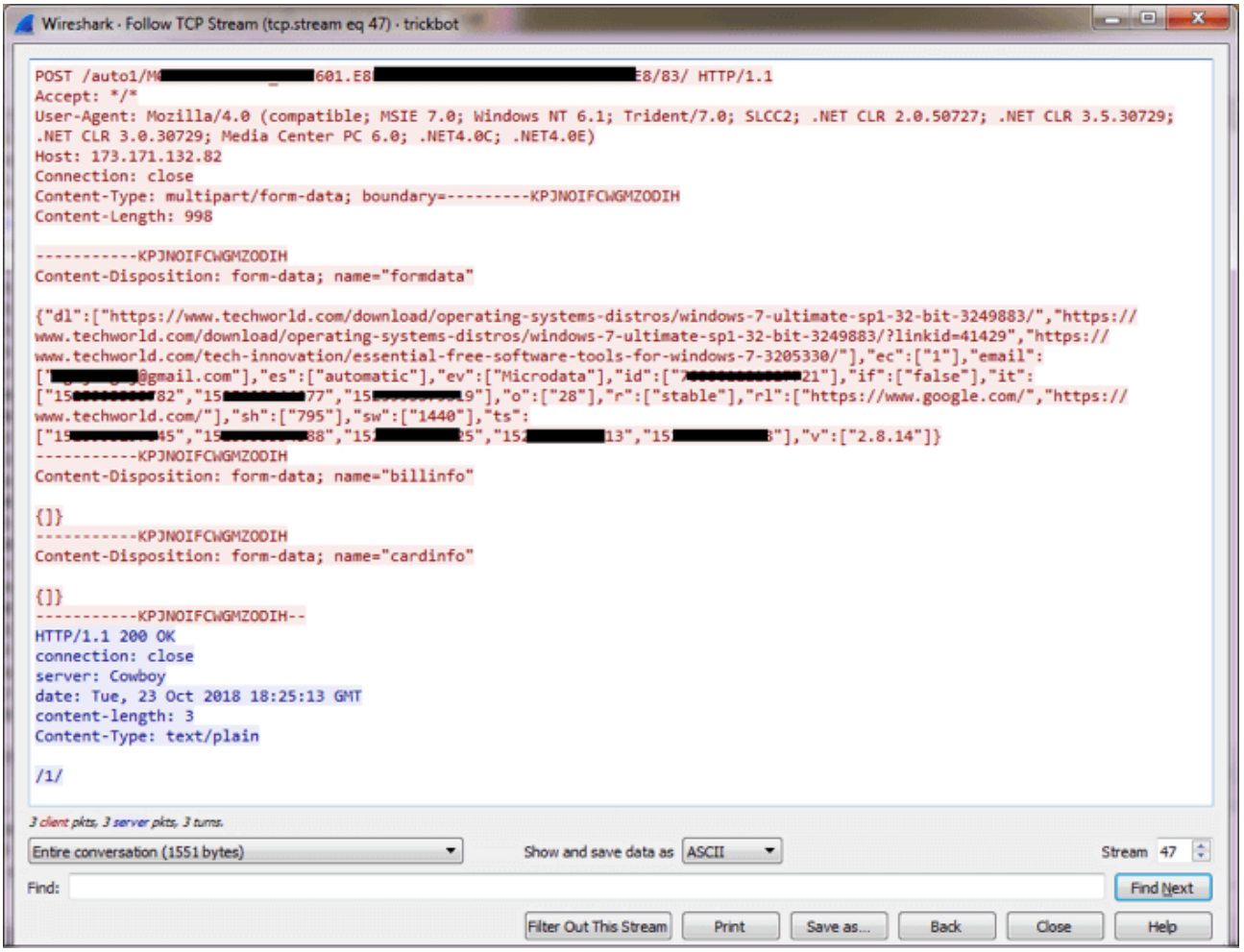

Figure 13. Report form autofill data from Chrome

Below is an IP list of the C&C servers that are used to handle the credential data. The IP list was decrypted by “pointes.exe” from the file “dpost”, and was passed to “pwgrab32” by calling the API WriteProcessMemory. Calling the API EnumDpostServer(fun_index) of core-parser.dll, we can get one IP of them by using the fun_index.

<dpost>

<handler>http://173.171.132.82:8082</handler>

<handler>http://66.181.167.72:8082</handler>

<handler>http://46.146.252.178:8082</handler>

<handler>http://97.88.100.152:8082</handler>

<handler>http://174.105.232.193:8082</handler>

<handler>http://23.142.128.34:80</handler>

<handler>http://177.0.69.68:80</handler>

<handler>http://5.228.72.17:80</handler>

<handler>http://174.105.232.193:80</handler>

<handler>http://177.0.69.68:80</handler>

<handler>http://23.226.138.220:443</handler>

<handler>http://23.226.138.196:443</handler>

<handler>http://23.226.138.221:443</handler>

<handler>http://92.38.135.151:443</handler>

<handler>http://198.23.252.204:443</handler>

</dpost>

Solutions

"hxxp://excel-office.com/secure.excel " is rated as **Malicious Websites** by the FortiGuard Webfilter service, and Sep_report.xls is detected as **VBA/Agent.JHAZ!tr.dldr** and pointer.exe as **W32/GenKryptik.COMA!tr** by the FortiGuard Antivirus service.

How to remove this malware:

1) Open Task Scheduler and go to Task Scheduler(Local) -> Task Scheduler Library

2) Select the item named “Msnetcs”, press the Delete key, and then click Yes.

3) Restart your system and delete the entire folder of %AppData%\VsCard.

IoC

URL:

"hxxp://excel-office.com/secure.excel "

Sample SHA256:

[Sep_report.xls]

41288C8A4E58078DC2E905C07505E8C317D6CC60E2539BFA4DF5D557E874CDEC

[secure.excel] or [pointer.exe] or [pointes.exe]

D5CADEF60EDD2C4DE115FFD69328921D9438ACD76FB42F3FEC50BDAAB225620D

Reference

https://www.fortinet.com/blog/threat-research/deep-analysis-of-the-online-banking-botnet-trickbot.html

https://www.fortinet.com/blog/threat-research/new-trickbot-plugin-harvests-email-addresses-from-sql-servers-screenlocker-module-not-for-ransom.html

https://blog.malwarebytes.com/threat-analysis/2017/08/trickbot-comes-with-new-tricks-attacking-outlook-and-browsing-data/

https://www.webroot.com/blog/2018/03/21/trickbot-banking-trojan-adapts-new-module/

https://blog.trendmicro.com/trendlabs-security-intelligence/trickbot-shows-off-new-trick-password-grabber-module/

Sign up for our weekly FortiGuard Threat Brief.

Know your vulnerabilities – get the facts about your network security. A Fortinet Cyber Threat Assessment can help you better understand: Security and Threat Prevention, User Productivity, and Network Utilization and Performance.
