Affected platforms: Microsoft Windows Impacted parties: 64-bit Windows Users Impact: Controls a victim's device and collects sensitive information Severity level: Critical

Fortinet's FortiGuard Labs recently captured more than 500 Microsoft Excel files involved in a campaign to deliver a fresh Emotet Trojan onto the victim's device.

Emotet, known as a modular Trojan, was first discovered in the middle of 2014. Since then, it has become very active, continually updating itself. It has also been highlighted in cybersecurity news from time to time. Emotet uses social engineering, like email, to lure recipients into opening attached document files (including Word, Excel, PDF, etc.) or to click links within the content of the email that downloads the latest Emotet variant onto the victim's device and then executes it.

In Part I of this post, I explained how this variant of Emotet is spread by malicious VBA code in Excel documents, how the downloaded Emotet malware runs within a Rundll32 program, what kind of anti-analysis techniques this variant uses., how it encrypts and submits its victim's data to its C2 server., what Emotet does when it receives response data from the C2 server, and what Emotet does to enable persistence on the victim's device.

In this post, you will learn what the data in response packets with malicious modules look like, what modules have been received from the C2 server for the current Emotet campaign, and how they are deployed in the victim's device. You will also discover what sensitive data those modules steal from a victim's device.

When X.dll Receives a Response with a Module

Once the C2 server has processed and detected the first submitted packet that includes critical data——such as the victim's device system version, Windows architecture, etc.——it replies with malicious modules for Emotet to execute in the victim's device. All the received modules are fileless. That is, they only exist in memory and are processed by the X.dll (the core of Emotet) running in Rundll32.exe.

Figure 1.1 – A decrypted module in the packet

Figure 1.1 is a screenshot of X.dll's code and memory. The bottom is a C2's response packet, just decrypted in memory by calling a function of 10012371. Referring to Figure 5.3 in part I of this series will help you understand the structure of the packet.

The box marked in red is the verification data (99 DE … DD A5), a signed hash of the rest data of the packet. The following dword, 0x00000000, marked in yellow, is a flag that tells Emotet how to run the replied module. 0x00 tells it to execute the module in a newly-created thread. The binary block in blue is the module. It starts with the module size, 0x79400 in this example, and the rest part is the module binary data (4D 5A 90 00 …).

Emotet has to verify the decrypted data, as shown in Figure 1.1, using the 40H verification data.

It then deploys the received module into memory and prepares to execute it. It then calls its entry point in a newly created thread. This post will refer to this module as a "thread-module." Its primary purposes are to extract and execute the final functional module that steals sensitive data from the victim's device and to submit the stolen data to its C2 server, which will be discussed later in this analysis. Figure 1.2 shows where the thread function ASM code calls the entry point of the deployed thread-module.

Figure 1.2 – Emotet thread function to call the thread-module's entry point

Thread-Module —— Performs Process Hollowing

The thread-module proceeds to decrypt a PE file, the final functional module, from its .text section into memory. To execute this module, it performs process hollowing. It does this by copying a Windows file, "certutil.exe", from either "%Windir%\SysWOW64\certutil.exe" or "%Windir%\system32\certutil.exe" into the "%temp%" folder. It then renames it to a random file name, like "uvbubqj.exe". Next, the thread-module creates a suspended process with this file.

Figure 2.1 – Call API CreateProcessW() to create a suspended process

As you may see in the command line string in Figure 2.1, "uvbubqj.exe" is the copied "certutil.exe", "/scomma" and the subsequent temporary file ——"C:\Users\Bobs\AppData\Local\Temp\60B2.tmp" —— are the parameters for the process. The temporary file name is generated by calling the API GetTempFileNameW(). The path of the temporary file "60B2.tmp" is read by the functional module and used to save stolen information. The sixth argument to CreateProcessW() is 0x00000004, which is a creation flag indicating "CREATE_SUSPENDED" with which CreateProcessW() creates a process and enters suspended status.

It then calls a group of APIs, like GetThreadContext(), VirtualAllocEx(), ReadProcessMemory(), WriteProcessMemory(), and so on, to inject the final functional module into the new process' memory. The API SetThreadContext() is called later to set the new process EIP register pointing to the entry point of the functional module, which is invoked after calling the API ResumeThread().

Afterward, the thread-module starts to monitor the temporary file in a loop until it is created with the stolen information from the victim's device.

Looking at the Functional Modules

In the above analysis, I explained how a C2 module is loaded and executed in the victim's device.

The C2 server can return many modules, each going through the same process as described above. They will have a thread-module, run in their thread, and perform their own process hollowing.

I received three C2 modules. I will elaborate on how they work on the victim's device in the following sections.

Module1 - Stealing Credentials from a Victim's Browsers

A Self-Extracting packer protects this module. It decrypts a PE file when it runs, overrides the existing code of "certutil.exe", and then gets it executed.

The unpacked PE file is a freeware called "WebBrowserPassView" developed by NirSoft. It was designed as a password recovery tool but has been abused by malicious actors to steal the victim's credentials. A user interface displays the saved credentials stored within several web browsers.

Figure 3.1 — Open the WebBrowserPassView module

Figure 3.1 shows what this module looks like when I open it in my test environment. This Emotet variant uses WebBrowserPassView v2.06.

Its thread-module passes command line parameters like "/scomma C:\Users\Bobs\AppData\Local\Temp\7B3C.tmp" to the process, which can switch WebBrowserPassView to a No-Window mode and save the retrieved credentials to a given temporary file.

From its code, I learned it could collect the credentials from a variety of web browsers:

Microsoft IE, Microsoft Edge, Google Chrome, Mozilla Firefox, Opera, Apple Safari, SeaMonkey, Yandex, Vivaldi, Waterfox, and all other Chromium-based browsers.

The stolen credentials contain the following information:

• URL: The URLs that credentials are saved for • Web Browser: The browser name that holds the credentials • User Name, Password: The credentials • Password Strength: Strong or weak • User Name Field: The control name type into the user name field • Password Field: The string entered in the password field • Created Time: When it was saved • Modified Time: Time when credentials were updated • Filename: What file it has stolen the credentials from

All the credentials are saved in a temporary file.

Module2 - Stealing Email Contact Information

This module steals its victim's email contacts from their email folders inside Microsoft Outlook by going through the victim's emails one by one. It keeps the gathered contact information in a doubly-linked chain structure.

Figure 4.1 shows one email contact obtained from an email within my test Outlook account that was then added into the doubly-linked chain, as shown at the bottom. The collected data shows the Person name and Email address of the email sender. In this example, it collected "Outlook" and "outlook@email2.office.com" from the displayed email message.

Figure 4.1 — One stolen contact in a doubly-linked chain

This module enumerates all collected emails and puts the unique email contact information into the doubly linked chain. To collect Outlook's data, it has to call several APIs, including MAPIInitialize(), MAPILogonEx(), and MAPIFreeBuffer(), as well as create some COM objects by calling the API CoCreateInstance(), such as OlkAccountManager and OlkMail.

Finally, it retrieves those email contacts from the linked chain one by one and saves them into the temporary file that comes from the command line parameter. Figure 4.2 shows a screenshot of the temporary file, "%temp%\6827.tmp" in this example, along with the collected email contacts.

Figure 4.2 — The temporary file with stolen email contact information

Module3 - Stealing Account Settings of Victim's Email Clients

This functional module focuses on stealing its victim's email account settings and the credentials from their email clients. It is also a packer-protected module, so it does the same thing as Module1 when its entry point is called.

According to my analysis, the unpacked PE file is an EXE file that is another freeware from NirSoft called "Mail PassView". It was originally designed as a small password recovery tool for email clients. Emotet is using the latest version——v1.92. Figure 5.1 is a screenshot of this software running on my test environment.

Figure 5.1 — Open Mail PassView in my test environment

Going through its code and constant strings, we learned it could obtain email account settings and credentials from the following email clients or other clients that could save email credentials:

Mozilla Thunderbird, Eudora, Microsoft Outlook, Microsoft Outlook Express, Windows Mail, MSNMessenger, Windows Live Mail, Group Mail, IncrediMail, Yahoo! Mail, Yahoo! Messenger, Hotmail, Google Desktop, and Google Talk.

It collects the settings and credentials from both the system registry and the local configuration files of these email clients. Figure 5.2 is a segment of the ASM code from a common function that has predefined many value names.

The software repeatedly reads User Name, Server Address, Server Port, and similar information from the system registry through these value-names under the subkeys "HKCU\Software\Microsoft\Internet Account Manager\Accounts" and "HKCU\Software\Microsoft\Office\Outlook\OMI Account Manager\Accounts", which are the places to save the settings and credentials for Microsoft Outlook and Microsoft Outlook Express.

Figure 5.2 — Defined value-names for reading from the system registry

This time, the command line parameter string to this software is "/scomma C:\Users\Bobs\AppData\Local\Temp\8042.tmp", where "/scomma" allows the process to run without a window and save the retrieved information to the temporary file followed.

Thread-Module — Submit Stolen Data

With the functional modules working to steal sensitive data, the thread-module keeps monitoring the temporary file until it is created with the stolen information.

It then loads the stolen data from the temporary file to memory and then deletes the file. Before submitting the stolen data to the C2 server, it compresses the data and encrypts it.

Figure 6.1 — Call BCryptEncrypt() to encrypt the stolen data

This example, shown in Figure 6.1, is where it was about to call the API BCryptEncrypt() to encrypt the packet, which begins from 4790E0. The section outlined in red is like the packet header. It contains the packet type (0x3EA) that tells the C2 server what kind of data is in the packet, a sha256 hash code (69 35 … 3C 4A) of the data, a module ID (0x14), as well as the Victim's ID. The subsequent data, marked in blue, starts with a data size (0x398) of the following data (from 10 55 52 4C … to the end), which are the compressed web browser credentials.

This thread-module uses eleven C2 servers to receive data stolen from the victim's device. The IP and Ports of these C2 servers are encrypted in memory and get decrypted before submitting the stolen data. The three downloaded modules have the same C2 server list, which can be found in the "IOC" section at the end of this analysis.

Figure 6.2 — Display of a captured packet to C2 server with encrypted data

Figure 6.2 is a screenshot of a proxy tool showing how the packet with the stolen victim's sensitive data is sent to its C2 server.

It uses the HTTP Post method with a randomized URL to submit the stolen data in the body, which consists of a 40H-long exported key at the beginning with the encrypted data following, as shown in Figure 6.2. The C2 server can decrypt the submitted data using the 40H exported key.

Conclusion

In Part II of this analysis, I started with a received module packet from a C2 server and explained the structure of the packet. Next, I showed how the module (thread-module) is executed in a newly created thread. We then walked through how the thread-module performs process hollowing to execute the functional modules.

In discussing the three received modules, I elaborated on what kind of data Emotet can steal from the victim's device, such as email contact information from the victim's email account, the email account's settings, credentials from the victim's email client, and credentials saved in a wide range of web browsers.

Finally, going back to the thread-module, Emotet reads the stolen information from the given temporary files. It then compresses and encrypts the data, which is ultimately submitted using the HTTP Post method to the C2 server.

Fortinet Protections

Fortinet customers are already protected from this malware by FortiGuard's Web Filtering, AntiVirus, FortiMail, FortiClient, FortiEDR, and CDR (content disarm and reconstruction) services, as follows:

The malicious Macro inside the Excel sample mentioned in Part I of the post can be disarmed by the FortiGuard CDR (content disarm and reconstruction) service.

All relevant URLs have been rated as "Malicious Websites" by the FortiGuard Web Filtering service.

The captured Excel sample and the downloaded Emotet dll file are detected as "VBA/Emotet.2826!tr.dldr " and " W32/Emotet.B185!tr" and are blocked by the FortiGuard AntiVirus service.

FortiEDR detects both the Excel file and Emotet dll file as malicious based on its behavior.

In addition to these protections, Fortinet also provides multiple solutions designed to help train users in detecting and understanding phishing threats:

We encourage organizations to have their end users take our FREE NSE Training: NSE 1 – Information Security Awareness. It includes a module on Internet threats designed to help end-users learn how to identify and protect themselves from various types of phishing attacks. This training can then be reinforced using our FortiPhish phishing simulation service. It uses real-world attack scenarios to train users, test awareness and vigilance, and reinforce proper practices for handling phishing incidents.

IOCs

C2 Server List in the three thread-modules:

144[.]217[.]88[.]125:443

67[.]205[.]162[.]68:8080

54[.]36[.]98[.]59:7080

45[.]184[.]36[.]10:8080

47[.]110[.]149[.]223:8080

159[.]65[.]1[.]71:8080

51[.]178[.]186[.]134:443

131[.]100[.]24[.]199:8080

51[.]91[.]142[.]158:80

51[.]79[.]205[.]117:8080

176[.]31[.]163[.]17:8080

Learn more about FortiGuard Labs global threat intelligence and research and the FortiGuard Security Subscriptions and Services portfolio.