

# TrickBot ... many tricks - VinCSS Blog

By Yên Hứa

Published: 2021-10-26 · Archived: 2026-04-05 23:39:15 UTC

## Table of Contents

- [1. Introduction](#)
- [2. Analyze malicious document](#)
- [3. Analyze easyMicrosoftHop.jpg.payload \(RCSeparator.dll – 48cba467be618d42896f89d79d21121\)](#)
- [4. Analyze shellcode](#)
- [5. Analyze the first Dll \(b67694ddd98298b539bddc8cab255d\)](#)
- [6. Analyze the second Dll \(34d6a6bffa656c6b0c7b588e111dbed1\)](#)
- [7. Analyze the third Dll \(templ.dll – 3409f865936a247957955ad2df45a2cd\)](#)
- [8. Analyze the final shellcode](#)
- [9. Dump Trickbot core payload 32-bit and extract C2 configuration](#)
  - [9.1. Dump payload 32-bit](#)
  - [9.2. Analyze Trickbot core payload and extract C2s configuration](#)
    - [9.2.1. Dynamic APIs resolve](#)
    - [9.2.2. Decrypt strings](#)
  - [9.3. Decrypt the configuration and extract the C2s list](#)
    - [9.3.1. Decrypt the configuration](#)
    - [9.3.2. Extract C2s list](#)
- [10. References](#)
- [11. Appendix 1 – Complete list of decrypted strings](#)
- [12. Appendix 2 – C2s list](#)

## 1. Introduction

First discovered in 2016, until now **TrickBot** (*aka TrickLoader or Trickster*) has become one of the most popular and dangerous malware in today's threat landscape. The gangs behind TrickBot are constantly evolving to add new features and tricks. Trickbot is multi-modular malware, with a main payload will be responsible for loading other plugins capable of performing specific tasks such as steal credentials and sensitive information, provide remote access, spread it over the local network, and download other malwares.

Trickbot roots are being traced to elite Russian-speaking cybercriminals. According to these reports ([1](#), [2](#)), up to now, at least two people believed to be members of this group have been arrested. Even so, other gang members are currently continuing to operate as normal.

Through continuous cyber security monitoring and system protection for customer recently, **VinCSS** has successfully detected and prevented a phishing attack campaign to distribute malware to customer that was protected by us. After the deep dive analysis and dissection of the malware techniques, we can confirm that this is a sample of the Trickbot malware family.

In this article, we decided to provide a detail analysis of how Trickbot infects after launching by a malicious Word document, the techniques the malware uses to make it difficult to analyze. Unlike [Emotet](#) or [Qakbot](#), Trickbot hides C2 addresses by using fake C2 addresses mixed together with real C2 addresses in the configuration, we will cover how to extract the final C2 list at the end of this article. In addition, we present the method to recover the APIs as well as decode the strings of Trickbot based on [IDA AppCall feature](#) to make the analysis process easier.

## 2. Analyze malicious document

The attacker somehow infected the partner's mail server system, thereby taking control of the email account on the server, inserting email with attachment containing malware into the email exchange flow between the two parties. The content of this email is as follows:

After extracting the **request.zip** with the password provided in the email, I obtained [require 010.04.2021.doc](#):

Check the **require 010.04.2021.doc** file and found that this file contains VBA code:

I focus to the red highlight code in the above image. Extract the relevant data area and do the corresponding replacement, obtain the html content containing JavaScript as the figure below:

The JavaScript code in the figure will do the decoding of the base64 blob assigned to the rockCleanJump and rapHopWindows variables. With the first base64 blob, it will download the payload to the victim's computer and save it as **easyMicrosoftHop.jpg**:

With the second base64 blob, it will use **regsvr32** to execute the downloaded payload.

With the above information, I can conclude that **easyMicrosoftHop.jpg** is a Dll file.

**3. Analyze easyMicrosoftHop.jpg payload (RCSeparator.dll – 48cba467be618d42896f89d79d211121)**

This file is not available on VT, however if search by *imphash*: `f34a0f23e05f2c2a829565c932b87430` will get the same payloads. These payloads have been uploaded to VT recently:

Examining this payload, this is a DLL with the original name is **RCSeparator.dll**, and it has one exported function is **DllRegisterServer**.

The file's metadata info is as follows:

The sample is not packed, but through a quick check the sections information, it can be seen that its code has been obfuscated, and the `.rsrc` section is likely to contain an encrypted payload.

By viewing resources in this sample, I found a resource named `HTML`, size `0x38333` bytes, containing random bytes. I guess that it will use this resource to decode a new payload.



- Use the resolved APIs to access and get the entire content of the resource that was mentioned above:

- Decode to shellcode and execute this shellcode by using [QueueUserAPC](#) and [NtTestAlert](#) functions.

Dump shellcode for further analysis. Parse this shellcode and found that it has **3 embedded** Dlls as following:

#### **4. Analyze shellcode**

The code of the above shellcode will call the **f\_dll\_loader** function to load the first Dll into memory with the following parameter:

At the function **f\_dll\_loader**, the shellcode finds the addresses of Windows API functions on runtime according to the pre-computed hashes:

The entire **f\_dll\_loader** function will perform the task of a loader, after mapping the Dll into memory will find the Dll's **DllEntryPoint** address and call this address to execute the code of first Dll:

Here, I dumped the first Dll to disk for further analysis.

#### **5. Analyze the first Dll (b67694ddf98298b539bdc8cab255d)**

This file is not available on VT, however if search by *imphash*: 1f6199c52a5d3ffac2a25f6b3601dd22 thi will get the same payloads:

According to the information that Import Directory provides, it can be guessed that this DLL will also do the job of a loader:

The code at **DllEntryPoint** will call the function responsible for loading and executing the second DLL:

The entire **f\_dll\_loader** function has the same code as the shellcode analyzed above, after mapping the entire second DLL into memory, it will retrieve the DLL's **DllEntryPoint** address and call this address to execute the next stage:

I dumped the second Dll to disk for easier analysis.

#### **6. Analyze the second Dll (34d6a6bffa656c6b0c7b588e111dbed1)**

This Dll has already been uploaded to [VirusTotal](#). Imports of the second Dll are the same as the first one:

The code at the **DllEntryPoint** function of this Dll performs the following task:

- Mapping the third DLL into memory.

- Find the **DllRegisterServer** function and call to this function:

I again dumped the third DLL to disk for further analysis.

#### **7. Analyze the third DLL (*templ.dll* – *3409f865936a247957955ad2df45a2cd*)**

Examining the above dumped DLL, its original name is **templ.dll**, and it has one exported function is **DllRegisterServer**.

This dll is also not available on VT, but searching by *imphash: b79a86dfbbe6d8e177dfb7ae70d4922* will returns some similar files.

The file is not packed, its code is obfuscated or will decode the new payload:

The code at the **DllRegisterServer** function of this DLL performs the following tasks:

- Allocate a memory area to store the decrypted payload.
- Perform the decryption routine to decrypt new payload into the allocated memory area. This payload is a shellcode.
- Call to shellcode to execute the final stage.

The decryption function uses a loop to xor the data as follows:

To be quick, I use **x64dbg** for debugging. Shellcode after decoding will be as follows:

## **8. Analyze the final shellcode**

Observe this shellcode and I see that it stores strings near the end of the file. In my personal experience these are likely base64 strings and keys for decoding

Perform decoding, I got the following strings:

Based on the above decoding information, I guess that this shellcode will continue to inject the payload into the **wermgr.exe** process. To verify, I debug this shellcode right after the **templ.dll** does the decoding and calls to the shellcode. Set breakpoint at **CreateProcessInternalW** function and execute:

So, as you can see in the above figure, the shellcode injects the payload into the **wermgr.exe (64-bit)** process. Under the cover of the **wermgr.exe** system process, the malicious code will now make connections to many C2 addresses as the following picture below:

## 9. Dump Trickbot core payload 32-bit and extract C2 configuration

### 9.1. Dump payload 32-bit

According to the above shellcode analysis results, it can be seen that the final payload has been injected into the **wermgr.exe (64-bit)** process, so this payload is also 64-bit. However, **templ.dll** is a 32-bit Dll, so to make it easier to gain an understand of the payload's code as well as extract the C2 configuration, we will dump the core 32-bit payload of malware. I debug shellcode when it is called by **templ.dll**, set breakpoints at **VirtualAlloc**, **GetNativeSystemInfo** functions. Execute shellcode, break at **GetNativeSystemInfo** function:

Follow in Dump the address will receive information about **SystemInfo**, execute the function and return to malware code. Modify the return result of **wProcessorArchitecture**:

Continuing to execute and follow the address allocated by the **VirtualAlloc** function, shellcode will unpack the main payload into the allocated memory, but the **"MZ"** signature has been wiped.

Dump payload to disk and fix **MZ** signature. I have the [core binary \(32-bit\) of Trickbot](#):

Payload has no information about Imports, so it will retrieve the addresses of APIs during runtime.

## **9.2. Analyze Trickbot core payload and extract C2s configuration**

### **9.2.1. Dynamic APIs resolve**

Similar to the [Emotet](#), [Qakbot](#), ... Trickbot payload also finds the address of the API function(s) through searching the pre-computed hash based on the API function name. Information about the DLLs as well as the pre-computed hashes is stored in the global variable with the following structure:

These fields have the following meanings:

- **dll\_str\_idx**: is used to decode the name of the DLL that Trickbot will use. And then, get the base address of this DLL.
- **nHashValue**: number of hash is pre-computed, corresponding to the number of API functions to find.
- **pre-computed hash**: are the pre-computed hash values of the API function.
- **nOrdinalVal**: number of ordinal values, corresponding to functions that will be retrieved the address based on the calculated ordinal's information.
- **Orinal\_value**: values are used to calculate the actual ordinal value of the API function that need to retrieve address.

Based on these fields, Trickbot will retrieving the addresses of the APIs as following:

The pseudocode of the function that calculates the hash based on the name of the API function:

Based on the above pseudocode, I can rewrite the hash calculation code in Python as follows:

All real addresses of APIs after being obtained will be stored at the address 0x00420000 as shown in the picture. Therefore, in order to get all the information about the APIs that Trickbot will use, I apply the method described [in this article](#). The result after restore the API(s) functions as the figure below:

### 9.2.2. Decrypt strings

All the main strings that used by payload are encrypted and stored at the **.data** section as following:

The decode function receives the input parameter as the index value of the string, then decodes the string using the base64 algorithm with the custom character set:

To be able to decode these strings and add related annotations in IDA, I use IDA's [Appcall](#) feature and refer to the code [here](#). The entire python code is as follows:

The results before and after the script execution will make the analysis easier:

In addition, for easy tracking and comparison, we can also write a standalone decryption script to get the entire list of strings. Please see the **Appendix 1 – Complete list of decrypted strings** below.

### **9.3. Decrypt the configuration and extract the C2s list**

#### **9.3.1. Decrypt the configuration**

Trickbot stores encrypted configuration information in the .text section, when executed it will get information about the size of the data and allocate memory accordingly. After that will perform data decryption by using a xor loop.

The data obtained after the above step will be decrypted again by using AES algorithm (MODE\_CBC) to get the C2s list. Before decryption, Trickbot will generate the AES key and IV:

The calculated **aes\_key** and **aes\_iv** values will then be used for data decryption as followings:

Based on the pseudocodes above, combined with the [hashherezade](#) code reference [here](#), I can rewrite the python code that decrypts the C2 configuration that Trickbot uses in this sample:

With the above decrypted configuration, we get the C2s list as shown above. However, in this list:

- IP addresses in the `<srv> </srv>` tag are real C2 addresses.
- IP addresses in the `<srva> </srva>` tag will be later transformed by Trickbot.

Trickbot use the following code to convert the addresses in the `<srva>` `</srva>` tag to real C2 addresses.

The above pseudocode is converted to python code as below:

Here is the C2 list after the transformation:

Please see **Appendix 2 – C2s list** below for the complete list.

## 10. References

- [Trickbot Still Alive and Well](#)

- [Trickbot Brief: Creds and Beacons](#)
- [Importing Ollydbg Addresses into IDA](#)
- [https://github.com/hasherezade/malware\\_analysis/tree/master/trickbot](https://github.com/hasherezade/malware_analysis/tree/master/trickbot)
- [Introducing the Appcall feature in IDA Pro 5.6](#)
- <https://github.com/coldshell/IDA-appcall>
- [Detracking TrickBot Loader](#)

## 11. Appendix 1 – Complete list of decrypted strings

### All decrypted strings

index : 0 → Decoded string : b'checkip.amazonaws.com'  
index : 1 → Decoded string : b'ipecho.net'  
index : 2 → Decoded string : b'ipinfo.io'  
index : 3 → Decoded string : b'api.ipify.org'  
index : 4 → Decoded string : b'icanhazip.com'  
index : 5 → Decoded string : b'myexternalip.com'  
index : 6 → Decoded string : b'wtfismyip.com'  
index : 7 → Decoded string : b'ip.anysrc.net'  
index : 8 → Decoded string : b'api.ipify.org'  
index : 9 → Decoded string : b'api.ip.sb'  
index : 10 → Decoded string : b'ident.me'  
index : 11 → Decoded string : b'www.myexternalip.com'  
index : 12 → Decoded string : b'/plain'  
index : 13 → Decoded string : b'/ip'  
index : 14 → Decoded string : b'/raw'  
index : 15 → Decoded string : b'/text'  
index : 16 → Decoded string : b'/?format=text'  
index : 17 → Decoded string : b'zen.spamhaus.org'  
index : 18 → Decoded string : b'cbl.abuseat.org'  
index : 19 → Decoded string : b'b.barracudacentral.org'  
index : 20 → Decoded string : b'dnsbl-1.uceprotect.net'  
index : 21 → Decoded string : b'spam.dnsbl.sorbs.net'  
index : 22 → Decoded string : b'bdns.at'  
index : 23 → Decoded string : b'bdns.by'  
index : 24 → Decoded string : b'bdns.co'  
index : 25 → Decoded string : b'bdns.im'  
index : 26 → Decoded string : b'bdns.link'  
index : 27 → Decoded string : b'bdns.nu'  
index : 28 → Decoded string : b'bdns.pro'  
index : 29 → Decoded string : b'b-dns.se'  
index : 30 → Decoded string : b'ruv\_'  
index : 31 → Decoded string : b'<UserId>  
index : 32 → Decoded string : b'rundll32.exe '  
index : 33 → Decoded string : b'control'  
index : 34 → Decoded string : b' %u %u %u %u'  
index : 35 → Decoded string : b'</BootTrigger>n'  
index : 36 → Decoded string : b'path'  
index : 37 → Decoded string : b'Toolwiz Cleaner'  
index : 38 → Decoded string : b'GET'  
index : 39 → Decoded string : b'WTSGetActiveConsoleSessionId'  
index : 40 → Decoded string : b'Param 0'  
index : 41 → Decoded string : b'Create ZP failed'  
index : 42 → Decoded string : b'%s/%s/64/%s/%s/%s/'  
index : 43 → Decoded string : b'Decode param64 error'  
index : 44 → Decoded string : b'client is not behind NAT'  
index : 45 → Decoded string : b'Windows Server 2003'  
index : 46 → Decoded string : b'start'  
index : 47 → Decoded string : b'SYSTEM'  
index : 48 → Decoded string : b'kernel32.dll'  
index : 49 → Decoded string : b'SeDebugPrivilege'  
index : 50 → Decoded string : b'.txt'  
index : 51 → Decoded string : b'Load to M failed'  
index : 52 → Decoded string : b'winsta0/default'

index : 53 -> Decoded string : b'eventfail'  
index : 54 -> Decoded string : b'Windows 10 Server'  
index : 55 -> Decoded string : b'data'  
index : 56 -> Decoded string : b' working'  
index : 57 -> Decoded string : b'%u%u%u.'  
index : 58 -> Decoded string : b'</LogonTrigger>n'  
index : 59 -> Decoded string : b'shlwapi'  
index : 60 -> Decoded string : b'cn\  
index : 61 -> Decoded string : b'——Boundary%08X'  
index : 62 -> Decoded string : b'curl/7.78.0'  
index : 63 -> Decoded string : b'GetProcAddress'  
index : 64 -> Decoded string : b'</Command>n<Arguments>  
index : 65 -> Decoded string : b'^svchost.exe'  
index : 66 -> Decoded string : b'-%s-rnm'  
index : 67 -> Decoded string : b'SignatureLength'  
index : 68 -> Decoded string : b'tmp'  
index : 69 -> Decoded string : b'in'  
index : 70 -> Decoded string : b'SeTcbPrivilege'  
index : 71 -> Decoded string : b'52'  
index : 72 -> Decoded string : b'\  
index : 73 -> Decoded string : b'0.0.0.0'  
index : 74 -> Decoded string : b'</Exec>n</Actions>n</Task>n'  
index : 75 -> Decoded string : b'ModuleQuery'  
index : 76 -> Decoded string : b'No params'  
index : 77 -> Decoded string : b'DNSBL'  
index : 78 -> Decoded string : b'%02X'  
index : 79 -> Decoded string : b'VERS'  
index : 80 -> Decoded string : b'cmd.exe'  
index : 81 -> Decoded string : b'/%s/%s/0/%s/%s/%s/%s/'  
index : 82 -> Decoded string : b'noname'  
index : 83 -> Decoded string : b'Control failed'  
index : 84 -> Decoded string : b'LoadLibraryW'  
index : 85 -> Decoded string : b'InitializeCriticalSection'  
index : 86 -> Decoded string : b'Create xml2 failed'  
index : 87 -> Decoded string : b'</Triggers>n<Principals>n<Principal id="Author">n'  
index : 88 -> Decoded string : b'not listed'  
index : 89 -> Decoded string : b'Create xml failed'  
index : 90 -> Decoded string : b'Windows Server 2012'  
index : 91 -> Decoded string : b'CloseHandle'  
index : 92 -> Decoded string : b'pIT connect failed, 0x%x'  
index : 93 -> Decoded string : b'Windows Server 2008'  
index : 94 -> Decoded string : b'WantRelease'  
index : 95 -> Decoded string : b'i:'  
index : 96 -> Decoded string : b'</Command>  
index : 97 -> Decoded string : b'client is behind NAT'  
index : 98 -> Decoded string : b'Register u failed, 0x%x'  
index : 99 -> Decoded string : b'/%s/%s/25/%s/'  
index : 100 -> Decoded string : b'/%s/%s/14/%s/%s/0/'  
index : 101 -> Decoded string : b'1108'  
index : 102 -> Decoded string : b'ExitProcess'  
index : 103 -> Decoded string : b'POST'  
index : 104 -> Decoded string : b'^cmd.exe'  
index : 105 -> Decoded string : b'PROMPT'  
index : 106 -> Decoded string : b'x64'  
index : 107 -> Decoded string : b'Windows 2000'  
index : 108 -> Decoded string : b'user'  
index : 109 -> Decoded string : b'Unable to load module from server'  
index : 110 -> Decoded string : b'/%s/%s/10/%s/%s/%u'  
index : 111 -> Decoded string : b'Process has been finishedn'  
index : 112 -> Decoded string : b'-%srnContent-Disposition: form-data; name="%S"rnrn'  
index : 113 -> Decoded string : b'Process was unloaded'  
index : 114 -> Decoded string : b'testscript'  
index : 115 -> Decoded string : b'CI failed, 0x%x'  
index : 116 -> Decoded string : b'%08lX%04lX%u'

index : 117 → Decoded string : b'Invalid params count'  
index : 118 → Decoded string : b'WTSQueryUserToken'  
index : 119 → Decoded string : b'S-1-5-18'  
index : 120 → Decoded string : b'\Toolwiz-Cleaner'  
index : 121 → Decoded string : b'dsize:%u'  
index : 122 → Decoded string : b'GetParentInfo error'  
index : 123 → Decoded string : b'reload%d'  
index : 124 → Decoded string : b'/%s/%s/5/%s/'  
index : 125 → Decoded string : b' '  
index : 126 → Decoded string : b'D;(A;;GA;;;WD)(A;;GA;;;BA)(A;;GA;;;SY)(A;;GA;;;RC)'  
index : 127 → Decoded string : b'explorer.exe'  
index : 128 → Decoded string : b'Unknown'  
index : 129 → Decoded string : b'x86'  
index : 130 → Decoded string : b'Content-Type: multipart/form-data; boundary=%srnContent-Length: %dmrn'  
index : 131 → Decoded string : b'pIT GetFolder failed, 0x%x'  
index : 132 → Decoded string : b'%s %s'  
index : 133 → Decoded string : b'Windows 7'  
index : 134 → Decoded string : b'en-EN\  
index : 135 → Decoded string : b't:'  
index : 136 → Decoded string : b'Execute from user'  
index : 137 → Decoded string :  
b'</Principal>n</Principals>n<Settings>n<MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>n<DisallowStartIfOnBatteries>>false  
Context="Author">n<Exec>n<Command>'  
index : 138 → Decoded string : b'Windows Server 2008 R2'  
index : 139 → Decoded string : b'Windows Vista'  
index : 140 → Decoded string : b'Run D failed'  
index : 141 → Decoded string : b'Win32 error'  
index : 142 → Decoded string : b'/%s/%s/1/%s/'  
index : 143 → Decoded string : b'SINJ'  
index : 144 → Decoded string : b'Module already unloaded'  
index : 145 → Decoded string : b'%016lX%016lX'  
index : 146 → Decoded string : b'</Arguments>n'  
index : 147 → Decoded string : b'Load to P failed'  
index : 148 → Decoded string : b'Module is not valid'  
index : 149 → Decoded string : b'<LogonTrigger>n<Enabled>>true</Enabled>n'  
index : 150 → Decoded string : b'<moduleconfig>\*</moduleconfig>'  
index : 151 → Decoded string : b'freebuffer'  
index : 152 → Decoded string : b'failed'  
index : 153 → Decoded string : b'listed'  
index : 154 → Decoded string : b'Windows Server 2012 R2'  
index : 155 → Decoded string : b'50'  
index : 156 → Decoded string : b'LeaveCriticalSection'  
index : 157 → Decoded string : b'info'  
index : 158 → Decoded string : b'ver.txt'  
index : 159 → Decoded string : b' /C cscript '  
index : 160 → Decoded string : b'ECCPUBLICBLOB'  
index : 161 → Decoded string : b'delete'  
index : 162 → Decoded string : b'm:'  
index : 163 → Decoded string : b'First'  
index : 164 → Decoded string : b' /C powershell -executionpolicy bypass -File '  
index : 165 → Decoded string : b'Global'  
index : 166 → Decoded string : b'kps'  
index : 167 → Decoded string : b'%s/%s/63/%s/%s/%s/%s/'  
index : 168 → Decoded string : b'%s%s'  
index : 169 → Decoded string : b'.reloc'  
index : 170 → Decoded string : b'rundll32'  
index : 171 → Decoded string : b'<?xml version="1.0" encoding="UTF-16"?>n<Task version="1.2" >n<RegistrationInfo>n<Version>1.1.1</Ve  
index : 172 → Decoded string : b'<LogonType>InteractiveToken</LogonType>n<RunLevel>LeastPrivilege</RunLevel>'  
index : 173 → Decoded string : b'SignalObjectAndWait'  
index : 174 → Decoded string : b'%s.%s.%s.%s'  
index : 175 → Decoded string : b'Windows 8'  
index : 176 → Decoded string : b'exc'  
index : 177 → Decoded string : b'Launch USER failed'  
index : 178 → Decoded string : b'regsvr32'

index : 179 -> Decoded string : b'settings.ini'  
index : 180 -> Decoded string : b'/%s/%s/23/%u/'  
index : 181 -> Decoded string : b'ECDSA\_P384'  
index : 182 -> Decoded string : b'%u.%u.%u.%u'  
index : 183 -> Decoded string : b'ResetEvent'  
index : 184 -> Decoded string : b'%s sTart'  
index : 185 -> Decoded string : b'%s %s SP%u'  
index : 186 -> Decoded string : b'.tmp'  
index : 187 -> Decoded string : b'</UserId>'  
index : 188 -> Decoded string : b'%s.%s'  
index : 189 -> Decoded string : b'/'  
index : 190 -> Decoded string : b'Register s failed, 0x%x'  
index : 191 -> Decoded string : b'mutant'  
index : 192 -> Decoded string : b'e:'  
index : 193 -> Decoded string : b'release'  
index : 194 -> Decoded string : b'wtsapi32'  
index : 195 -> Decoded string : b'Windows XP'  
index : 196 -> Decoded string : b'<BootTrigger>n<Enabled>>true</Enabled>n'  
index : 197 -> Decoded string : b'E: 0x%x A: 0x%p'  
index : 198 -> Decoded string : b'Find P failed'  
index : 199 -> Decoded string : b'Module has already been loaded'  
index : 200 -> Decoded string : b'Windows 8.1'  
index : 201 -> Decoded string : b'EnterCriticalSection'  
index : 202 -> Decoded string : b'Windows 10'  
index : 203 -> Decoded string : b'Execute from system'  
index : 204 -> Decoded string : b'<RunLevel>HighestAvailable</RunLevel>n<GroupId>NT AUTHORITY\SYSTEM</GroupId>n<LogonType'  
index : 205 -> Decoded string : b'NAT status'  
index : 206 -> Decoded string : b'Start failed'  
index : 207 -> Decoded string : b'WTSEnumerateSessionsA'  
index : 208 -> Decoded string : b'ps1'  
index : 209 -> Decoded string : b'WaitForSingleObject'  
index : 210 -> Decoded string : b'UrlEscapeW'  
index : 211 -> Decoded string : b'pIT NULL'  
index : 212 -> Decoded string : b'WTSFreeMemory'  
index : 213 -> Decoded string : b'USER32.dll'  
index : 214 -> Decoded string : b'WS2\_32.dll'  
index : 215 -> Decoded string : b'IPHLAPI.DLL'  
index : 216 -> Decoded string : b'WINHTTP.dll'  
index : 217 -> Decoded string : b'bcrypt.dll'  
index : 218 -> Decoded string : b'CRYPT32.dll'  
index : 219 -> Decoded string : b'OLEAUT32.dll'  
index : 220 -> Decoded string : b'SHELL32.dll'  
index : 221 -> Decoded string : b'USERENV.dll'  
index : 222 -> Decoded string : b'SHLWAPI.dll'  
index : 223 -> Decoded string : b'ole32.dll'  
index : 224 -> Decoded string : b'ADVAPI32.dll'  
index : 225 -> Decoded string : b'ntdll.dll'  
index : 226 -> Decoded string : b'ncrypt.dll'

## 12. Appendix 2 – C2s list

| Trickbot C2 List    |
|---------------------|
| 36.91.117.231:443   |
| 36.89.228.201:443   |
| 103.75.32.173:443   |
| 45.115.172.105:443  |
| 36.95.23.89:443     |
| 103.123.86.104:443  |
| 202.65.119.162:443  |
| 202.9.121.143:443   |
| 139.255.65.170:443  |
| 110.172.137.20:443  |
| 103.146.232.154:443 |

```
36.91.88.164:443
103.47.170.131:443
122.117.90.133:443
103.9.188.78:443
210.2.149.202:443
118.91.190.42:443
117.222.61.115:443
117.222.57.92:443
136.228.128.21:443
103.47.170.130:443
36.91.186.235:443
103.194.88.4:443
116.206.153.212:443
58.97.72.83:443
139.255.6.2:443
```

Click [here](#) for Vietnamese version.

**Tran Trung Kien (aka m4n0w4r)**

**Malware Analysis Expert**

**R&D Center – VinCSS (a member of Vingroup)**

---

Source: <https://blog.vincss.net/re025-trickbot-many-tricks/>