# China-based APT Mustang Panda might still have continued their attack activities against organizations in

By Yến Hứa

Published: 2022-05-20 · Archived: 2026-04-05 22:30:38 UTC

Table of Contents



## 1. Executive Summary

At VinCSS, through continuous cyber security monitoring, hunting malware samples and evaluating them to determine the potential risks, especially malware samples targeting Vietnam. Recently, during hunting on VirusTotal's platform and performing scan for specific byte patterns related to the **Mustang Panda (PlugX)**, we discovered a series of malware samples, suspected to be relevant to APT Mustang Panda, that was uploaded from Vietnam.

All of these samples share the same name as "**log.dll**" and have a rather low detection rate.

Based on the above information, we infer that there is a possibility that malware has been infected in certain orgs in Vietnam, so we decided to analyze these malware samples. During analysis, based on the detected indicators, we continue to investigate and set the scenario of the attack campaign.

A general overview of the execution flow demonstrated as follow:

Our blog includes:

- Technical analysis of the **log.dll** file.
- Technical analysis of shellcode decrypted from **log.dat**.
- Analyze **PlugX Dll** as well as decrypt PlugX configuration information.

## 2. Analyze the log.dll

In the list of hunted samples above, we choose the one with hash:
3171285c4a846368937968bf53bc48ae5c980fe32b0de10cf0226b9122576f4e

This sample was submitted to VirusTotal from **Vietnam** on **2022-04-25 14:04:36 UTC**

The information from the Rich Header suggests that it is likely compiled with **Visual Studio 2012/2013**:

By checking the sections information, we can see that it is packed or the code is obfuscated:

Sample has the original name **ljAt.dll**, and it exports two functions **LogFree** and **LogInit**:

Load sample into IDA, analyze the code of the two functions above:

- **LogFree** function:

Looking at this function, it can be seen that its code has been completely obfuscated by Obfuscator-LLVM, using the Control Flow Flattening technique:

After further analysis, I found that this function has no special task.

- **LogInit** function:

This function will call the **LogInit_0** function:

Similar to the above, the code at the **LogInit_0** function has also been completely obfuscated, it takes a long time for IDA to decompile the code of this function:

The primary task of the **LogInit_0** function is to call the function **f_read_content_of_log_dat_file_to_buf** for reading the content of **log.dat** file and execute the decrypted shellcode:

**f_read_content_of_log_dat_file_to_buf'**s code is also completely obfuscated:

The major task of this function as the following:

- Call the **GetModuleHandleW** function to retrieve the handle of **kernel32.dll**.
- Call the **GetProcAddress** function to get the addresses of the APIs: **VirtualAlloc**, **GetModuleFileNameA**, **CreateFileA**, **ReadFile**.
- Use the above APIs to retrieve the path to the **log.dat** file and read the contents of this file into the allocated memory.

- Decode the contents of **log.dat** into shellcode so that this shellcode is then executed by the call from the **LogInit_0**function.

### 3. Shellcode analysis

Based on the information analyzed above, we know that the **log.dll** file will read the content from the **log.dat** file and decrypt it into shellcode for further execution. Relying on this indicator, we continue to hunt **log.dat** file on VirusTotal which restrict the scope of submission source from Vietnam.

The results are following:

With the above results, at the time of analysis, we selected the **log.dat** file (2de77804e2bd9b843a826f194389c2605cfc17fd2fafde1b8eb2f819fc6c0c84) was submitted to VirusTotal on **2022-04-20 12:33:19 UTC** (5 days before the above **log.dll** file).

Debugging and dump the decrypted shellcode look like this:

I use two tools, FLOSS and scdbg to get an overview of this shellcode. The results can be seen in the screenshots below:

With the results obtained above, it can be seen that this shellcode will perform memory allocation and then call the **RtlDecompressBuffer** function to decompress the data with the compression format is **COMPRESSION_FORMAT_LZNT1**.

By using IDA to analyze this shellcode, its main task is to decompress a Dll into memory and call the exported function of this Dll to execute. The function that does this task is named **f_load_dll_from_memory**:

The code in this function will first get the base address of **kernel32.dll** based on the pre-calculated hash value is **0x6A4ABC5B**. This hash value has also been mentioned by us in this analysis.

Next it will retrieve the address of **GetProcAddress**:

By using the stackstring technique, the shellcode constructs the names of the APIs and gets the addresses of the following API functions:

Next, the shellcode performs a memory allocation (**compressed_buf**) of size **0x2E552**, then reads data from offset **0x1592** (on disk) and executes an xor loop with a key is **0x72** to fill data into the **compressed_buf**. In fact, the size of **compressed_buf** is **0x2E542**, but its first 16 bytes are used to store information about **signature**, **uncompressed_size**, **compressed_size**, so **0x10** is added.

Shellcode continues to allocate memory (**uncompressed_buf**) of size **0x4C000** and calls the **RtlDecompressBuffer** function to decompress the data at the **compressed_buf** into **uncompressed_buf** with the compression format is **COMPRESSION_FORMAT_LZNT1**.

Based on the above analysis results, it is easy to get the extracted Dll file (however, the file header information was destroyed):

Fix the header information and check with PE-bear, this Dll has the original name is **RFPmzNfQQFPXX** and only exports one function named **Main**:

Back to the shellcode, after decompressing the Dll into memory, it will perform the task of a loader to map this Dll into a new memory region. Then, call to the exported function (here is the **Main** function) to perform the the main task of malware:

***Note**: At the time of analyzing this shellcode, we have not yet confirmed it is a variant of the PlugX malware, but only raised doubts about the relationship. It was only when we analyzed the above extracted Dll, then we confirmed for sure that this was a variant of PlugX and renamed the fields in the struct for understandable reasons as screenshot above.*

We will not go into detailed analysis of this Dll, but only provide the necessary information to prove that this is a PlugX variant as well as the process of decrypting the configuration information that the malware will be used.

### 4.1. How PlugX calls an API function

In this variant, information about API functions is stored in **xmmword**, then loaded into the **xmm0** (128-bit) register, the missing part of the function name will be loaded through the stack. The malicious code gets the

handle of the Dll corresponding to these API functions, then uses **GetProcAddress** function to retrieve the address of the specified API function to use later:

### 4.2. Create main thread to execute

The malware adjusts the **SeDebugPrivilege** and **SeTcbPrivilege** tokens of its own process in order to gain full access to system processes. Then it creates its main thread, which is named "**bootProc**":

### 4.3. Communicating with C2

The malware can communicate with C2 via TCP, HTTP or UDP protocols:

### 4.4. Implemented commands

The malware will receive commands from the attacker to execute the corresponding functions related to *Disk*, *Network*, *Process*, *Registry*, etc.

The entire list of commands as shown in the table below that the attacker can execute through this malware sample:

| Command Group | Sub-command | Description |
|---|---|---|
| Disk | 0x3000 | Get information about the drives (type, free space) |

| | | |
|---|---|---|
| | 0x3001 | Find file |
| | 0x3002 | Find file recursively |
| | 0x3004 | Read data from the specified file |
| | 0x3007 | Write data to the specified file |
| | 0x300A | Create a new directory |
| | 0x300C | Create a new process on hidden desktop |
| | 0x300D | File action (file copy/rename/delete/move) |
| | 0x300E | Expand environment-variable strings |
| Nethood | 0xA000 | Enumeration of network resources |
| Netstat | 0xD000 | Retrieve a table that contains a list of TCP endpoints |
| | 0xD001 | Retrieve a table that contains a list of UDP endpoints |
| | 0xD002 | Set the state of a TCP connection |
| Option | 0x2000 | Lock the workstation’s display |
| | 0x2001 | Force shut down the system |
| | 0x2002 | Restart the system |
| | 0x2003 | Shut down the system safety |
| | 0x2005 | Display massage box |
| PortMap | 0xB000 | Perform port mapping |
| Process | 0x5000 | Retrieve processes info |
| | 0x5001 | Retrieve modules info |
| | 0x5002 | Terminate specified process |
| RegEdit | 0x9000 | Enumerate registry |
| | 0x9001 | Create registry |
| | 0x9002 | Delete registry |
| | 0x9003 | Copy registry |
| | 0x9004 | Enumerates the values of the specified open registry key |
| | 0x9005 | Sets the data and type of a specified value under a registry key |

| | | |
|---|---|---|
| | 0x9006 | Deletes a named value from the specified registry key |
| | 0x9007 | Retrieves a registry value |
| Service | 0x6000 | Retrieves the configuration parameters of the specified service |
| | 0x6001 | Changes the configuration parameters of a service |
| | 0x6002 | Starts a service |
| | 0x6003 | Sends a control code to a service |
| | 0x6004 | Delete service |
| Shell | 0x7002 | Create pipe and execute command line |
| SQL | 0xC000 | Get SQL data sources |
| | 0xC001 | Lists SQL drivers |
| | 0xC002 | Executes SQL statement |
| Telnet | 0x7100 | Start telnet server |
| Screen | 0x4000 | simulate working over the RDP Protocol |
| | 0x4100 | Take screenshot |
| KeyLog | 0xE000 | Perform key logger function, log keystrokes to file "**%allusersprofile%MSDN6.0USER.DAT**" |

### 4.5. Decrypt PlugX configuration

As analyzed above, the malware will connect to the C2 address via HTTP, TCP or UDP protocols depending on the specified configuration. So where is this config stored? With the old malware samples that we have analyzed (1, 2, 3, 4), the PlugX configuration is usually stored in the **.data** section with the size of **0x724 (1828)** bytes.

Going back to the sample we are analyzing, we see that before the step of checking the parameters passed when the malware executes, it will call the function that performs the task of decrypting the configuration:

Diving into this function, combined with additional debugging from shellcode, renaming the fields in the generated struct, we get the following information:

- PlugX's configuration is embedded in shellcode and starts at offset **0x69**.
- The size of the configuration is **0x0150C** (**5388**) bytes.
- Decryption key is **0xB4**.

With all the complete information as above, it is possible to recover the configuration information easily:

| IP | Port |
|---|---|
| 86.78.23.152 | 53 |
| 86.78.23.152 | 22 |
| 86.78.23.152 | 8080 |
| 86.78.23.152 | 23 |

In addition to the list of C2 addresses above, there is additional information related to the directory created on the victim machine to contain malware files as well as the name of the service that can be created:

To make our life easier, I wrote a python script to automatically extract configuration information for this variant. The output after running the script is as follows:

## 5. Conclusion

CrowdStrike researchers first published information on Mustang Panda in June 2018, after approximately one year of observing malicious activities that shared unique Tactics, Techniques, and Procedures (TTPs). However, according to research and collect from many different cybersecurity companies, this group of APTs has existed for more than a decade with different variants found around the world. Mustang Panda, believed to be a APT group based in China, is evaluated as one of the highly detrimental APT groups, applying sophisticated techniques to infect malware, aiming to gain as much long-term access as possible to conduct espionage and information theft.

In this blog we have analyzed the different steps the infamous PlugX RAT follows to start execution and avoid detection. Thereby, it can be seen that this APT group is still active and constantly looking for ways to improve their techniques. VinCSS will continue to search for additional samples and variants that may be associated with this PlugX variant that we analyzed in this article.

## 6. References


- [RE012-1]Phân tích mã độc lợi dụng dịch Covid-19 để phát tán giả mạo “Chỉ thị của thủ tướngNguyễn Xuân Phúc” – Phần 1
- [RE012-2]Phân tích mã độc lợi dụng dịch Covid-19 để phát tán giả mạo “Chỉ thị của thủ tướngNguyễn Xuân Phúc” – Phần 2
- PlugX: A Talisman to Behold
- THOR: Previously Unseen PlugX Variant Deployed During Microsoft Exchange Server Attacks byPKPLUG Group
- Mustang Panda deploys a new wave of malware targeting Europe
- BRONZE PRESIDENT Targets Russian Speakers with Updated PlugX
- China-Based APT Mustang Panda Targets Minority Groups, Public and Private SectorOrganizations

## 7. Indicators of Compromise

| |
|---|
| log.dll – db0c90da56ad338fa48c720d001f8ed240d545b032b2c2135b87eb9a56b07721 |
| log.dll – 84893f36dac3bba6bf09ea04da5d7b9608b892f76a7c25143deebe50ecbbdc5d |
| log.dll – 3171285c4a846368937968bf53bc48ae5c980fe32b0de10cf0226b9122576f4e |
| log.dll – da28eb4f4a66c2561ce1b9e827cb7c0e4b10afe0ee3efd82e3cc2110178c9b7a |
| log.dat – 2de77804e2bd9b843a826f194389c2605cfc17fd2fafde1b8eb2f819fc6c0c84Decrypted config:<br>[+] Folder name: %ProgramFiles%BitDefender Update<br>[+] Service name: BitDefender Crash Handler<br>[+] Proto info: HTTP://<br>[+] C2 servers:<br>86.78.23.152:53<br>86.78.23.152:22<br>86.78.23.152:8080<br>86.78.23.152:23<br>[+] Campaign ID: 1234 |
| log.dat – 0e9e270244371a51fbb0991ee246ef34775787132822d85da0c99f10b17539c0Decrypted config:<br>[+] Folder name: %ProgramFiles%BitDefender Update<br>[+] Service name: BitDefender Crash Handler<br>[+] Proto info: HTTP://<br>[+] C2 servers:<br>86.79.75.55:80<br>86.79.75.55:53<br>86.79.75.46:80<br>86.79.75.46:53<br>[+] Campaign ID: 1234 |

log.dat – 3268dc1cd5c629209df16b120e22f601a7642a85628b82c4715fe2b9fbc19eb0Decrypted config:

[+] Folder name: %ProgramFiles%Common FilesARO 2012

[+] Service name: BitDefender Crash Handler

[+] Proto info: HTTP://

[+] C2 servers:

86.78.23.152:23

86.78.23.152:22

86.78.23.152:8080

86.78.23.152:53

[+] Campaign ID: 1234

log.dat – 02a9b3beaa34a75a4e2788e0f7038aaf2b9c633a6bdbfe771882b4b7330fa0c5 (THOR PlugX)Decrypted config:

[+] Folder name: %ProgramFiles%BitDefender Handler

[+] Service name: BitDefender Update Handler

[+] Proto info: HTTP://

[+] C2 servers:

www.locvnpt.com:443

www.locvnpt.com:8080

www.locvnpt.com:80

www.locvnpt.com:53

[+] Campaign ID: 1234

*Click here for Vietnamese version.*

**Dang Dinh Phuong – Threat Hunter**

**Tran Trung Kien (aka m4n0w4r) – Malware Analysis Expert**

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

---