Technical Analysis of kkRAT

Muhammed Irfan V A ⋮ ⋮ 9/9/2025

Technical Analysis

Attack chain

In early May 2025, ThreatLabz identified a malware campaign delivering multiple RATs as the final payload. The attack chain for this campaign is shown in the figure below.

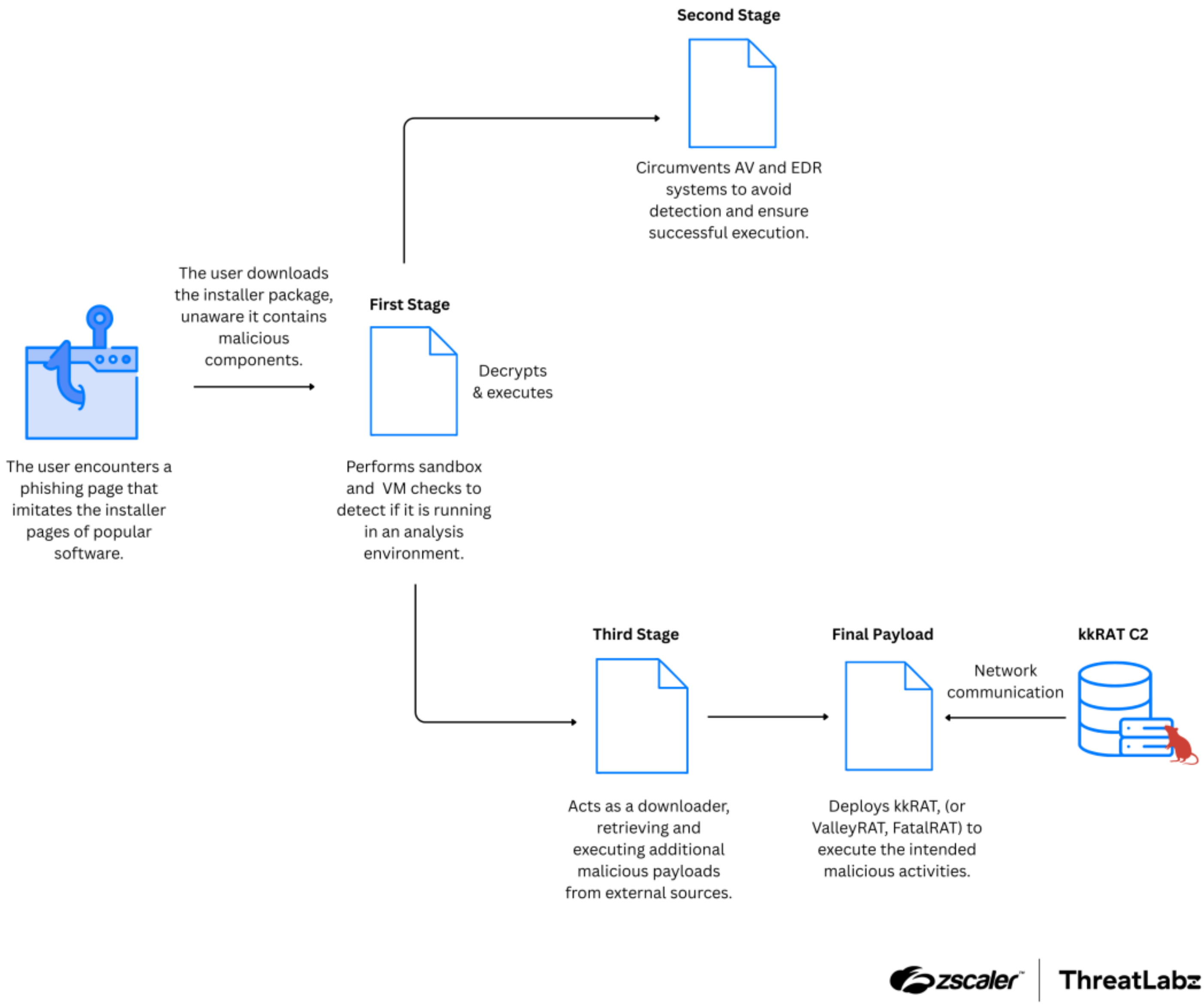

Figure 1: Attack chain for a malware campaign delivering several RATs.

The threat actor uses GitHub Pages to host phishing sites impersonating popular software installers. These installer packages are ZIP archives that contain a malicious executable file. The figure below highlights an example phishing page used in the campaign.

Figure 2: Example phishing page impersonating Ding Talk that ultimately delivers various RATs.

First stage

During the initial stage of the campaign, the malware employs two distinct methods to identify sandbox environments and virtual machines (VMs):

Time stability analysis

Using `QueryPerformanceCounter`, the malware measures the time for a repetitive operation, compares the average (expected 300 ms) to a threshold (0.0008), and identifies sandboxes/VMs if the deviation exceeds this limit.

Hardware configuration

The malware assesses disk space (minimum 50 GB) and CPU cores (minimum two). If these thresholds aren't met, the malware initiates evasive actions, including altering the Process Environment Block (PEB) structure:

- `ProcessParameters->ImagePathName` and `ProcessParameters->CommandLine` are altered to mimic `%WINDIR%\explorer.exe`.
- The malware also traverses `InLoadOrderModuleList`. If any entry's `BaseDllName` matches the current process name, both `BaseDllName` and `FullDllName` are rewritten to `%WINDIR%\explorer.exe`.

These modifications corrupt the final process snapshot taken by sandboxes and will result in the malware terminating execution.

After completing the sandbox and VM checks, the malware performs the following anti-analysis/obfuscation methods.

- **API resolution:** The malware dynamically loads required Windows API functions by performing single-byte XOR (key: 0x4) operations on stack strings.
- **Next-stage file decryption:** The malware applies single-byte XOR operations (key: 0x1) to extract decryption keys for the next-stage files.

Memory is allocated for next-stage shellcodes, which are decrypted, written, and directly executed by the first stage. All shellcodes utilized in the campaign employ pe_to_shellcode transformation logic.

Second stage

To bypass AV software and EDR systems, the malware employs several techniques. The first technique is verifying administrator privileges. If the malware does not have sufficient privileges, a message is displayed in Mandarin prompting the user for elevated access and exits. If the malware has administrator privileges, the malware enumerates all active network adapters and temporarily disables them, severing AV/EDR communication with the corresponding vendor's servers.

Following this, the malware scans the system for the presence of specific AV and EDR processes predominantly associated with China-based cybersecurity vendors. These vendors include:

- 360 Total Security
- QQ电脑管家
- HeroBravo System Diagnostics suite
- Kingsoft Internet Security
- 360 Internet Security suite

If targeted processes are detected, the malware uses a known vulnerable driver (`RTCore64.sys`) to disable AV/EDR functionalities. This is achieved by comparing the name of the AV/EDR driver that registered each callback. The complete list of targeted drivers can be found in the ThreatLabz GitHub repository.

The malware incorporates code borrowed from the RealBlindingEDR project to remove registered system callbacks, targeting three specific types of callbacks for elimination:

- ObRegister callback: Monitors, blocks, or modifies how the system creates and duplicates handles using callback routines.
- MiniFilter callback: Allows minifilter drivers to filter specific file Input/Output (I/O) operations.
- CmRegister callback: Monitors, blocks, or modifies Windows registry operations via callback routines.

After disabling callbacks, the malware terminates and deletes files of specific AV/EDR processes at the user level. The malware also creates a scheduled task to run with SYSTEM privileges to execute a batch script on every user logon to ensure the processes are repeatedly killed.

Next, the malware modifies registry keys associated with the 360 Total Security program:

- The `NetCheck` registry value is set to `0` in `HKLM\SOFTWARE\WOW6432Node\360Safe\360Scan` (presumably to disable network checks).
- Adds random data to a null value name under the registry key located at `HKU\360SPDM\CC2FCASH\speedmem2\x\b5e3891842b605bf7917ba84`.

Following these registry changes, the malware re-enables the previously disabled network adapters to restore the system's network connectivity. Thereafter, the first-stage shellcode executes the third-stage shellcode, which functions as a downloader to facilitate the next phase of the attack.

Third stage

The malware retrieves and executes a shellcode file named *2025.bin* from a hardcoded URL by utilizing the EnumDateFormatsA API callback. The shellcode, heavily obfuscated with junk code, downloads a Base64-encoded file named *output.log*, which is decoded to reveal structured data for subsequent attack stages. An example is shown below.

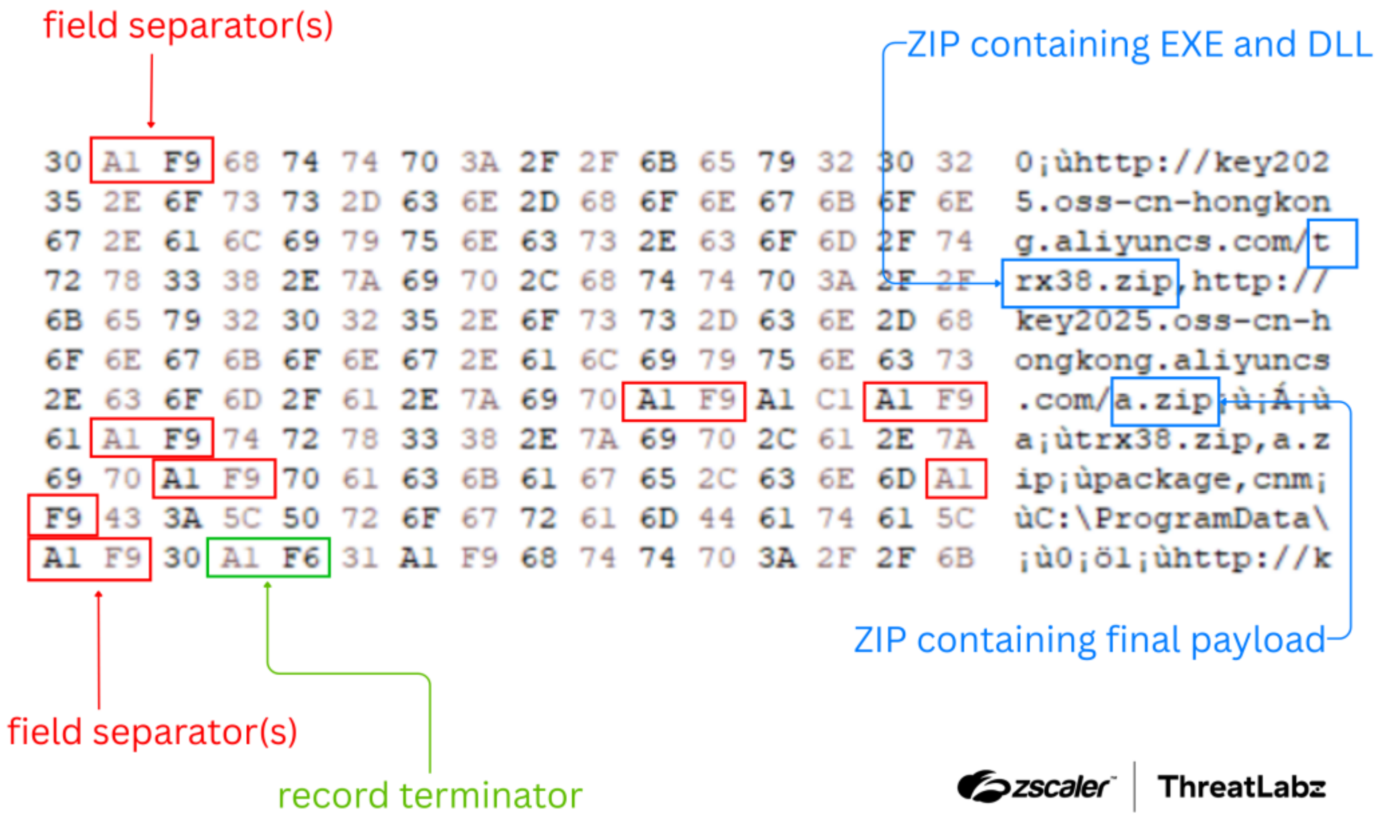

Figure 3: Hexdump of the decoded data used to download various RATs.

The decoded data is structured using the delimiters `0xA1 0xF9` that act as a field separator, dividing individual fields within a record, while `0xA1 0xF6` serves as a record terminator, marking the end of each record. The decoded data consists of 62 records, each record starts with an index ranging from 0 to 61. In each record, the second field contains two URLs, and these URLs are used to download two archive files:

- **trx38.zip:** When unzipped, trx38.zip includes a legitimate executable file and a malicious DLL.
- ***.zip:** (Where * represents a wildcard) This ZIP archive contains a file named *longlq.cl*, which holds the encrypted final payload.

The malware selects a record based on the last letter of the current process's filename. For example, if the filename was setu**p**.exe, the file **p.zip** would be downloaded. The malware then will create a shortcut for the legitimate executable extracted from trx38.zip, add this shortcut to the startup folder for persistence, and execute the legitimate executable to sideload the malicious DLL.

The malicious DLL decrypts and executes the final payload from the file longlq.cl using a 6-byte XOR key at offset 0xD3000, with encrypted data at 0xD3006. The final payload of the campaign varies based on the second ZIP archive that is downloaded. This campaign delivers three different RATs: ValleyRAT, FatalRAT, and kkRAT.

Final payload

Since ValleyRAT and FatalRAT are already extensively documented, they will not be analyzed in this section. However, kkRAT is a previously unknown malware family that incorporates elements from both Ghost RAT and Big Bad Wolf. These shared similarities are outlined below:

- **Ghost RAT:** kkRAT shares similarities with Ghost RAT's network communication protocols, but introduces an added layer of encryption applied after data compression. kkRAT also borrows several network commands from Ghost RAT, such as COMMAND_ACTIVED, COMMAND_KEYBOARD, and COMMAND_LIST_DRIVE.
- **Big Bad Wolf:** kkRAT adopts specific DLL exports from Big Bad Wolf's primary plugin DLL, including DllShell and DllScreen.

Encrypted configuration

kkRAT's configuration, such as the C2 server IP and port, version, and group identifier, are stored as encrypted strings and sent in the registration message. A Python script for decrypting this configuration is available in the ThreatLabz GitHub repository.

Device fingerprinting

After establishing a socket connection, kkRAT gathers system information for device fingerprinting. The collected data is sent to the C2 server in a registration message with the structure below.

```
struct REGISTRATIONINFO
{
BYTE Token; // 0x66 hardcoded value
OSVERSIONINFOEXA OsVerInfoEx; // OS version information
DWORD CPUClockMhz; // CPU frequency
int CPUNumber; // Number of processors
IN_ADDR IPAddress; // Host local IP
char HostName[50]; // Host name
bool IsWebCam; // Is there a web camera connected?
DWORD socketTime; // Time since the socket was established
DWORD Speed; // Internet speed in mbps
DWORD MemSize; // Total physical memory size
DWORD DriverSize; // Hard disk capacity
char Group[50]; // RAT Group - set to Default
char UpTime[32]; // System uptime
char Version[32]; // RAT Version - set to Enterprise
BOOL Is64; // 32-bit or 64-bit; 1 is 64 while 0 is 32
char AV[80]; // List of AV's installed
DWORD isIdle; // Is idle for more than 3 min?
char TG[40]; // Is Telegram present on the system?
```

```
char WC[40]; // Is WeChat present on the system?
char QQ[80]; // QQ number
BOOL IsAdmin;// Is Administrator
char UserName[50]; // Account username
};
```

Network communication protocol

kkRAT's network communication protocol closely resembles that of Ghost RAT, with an added layer of encryption applied after data compression. Each packet exchanged between kkRAT and the C2 server is sent via TCP and follows a specific structure, as illustrated in the figure below.

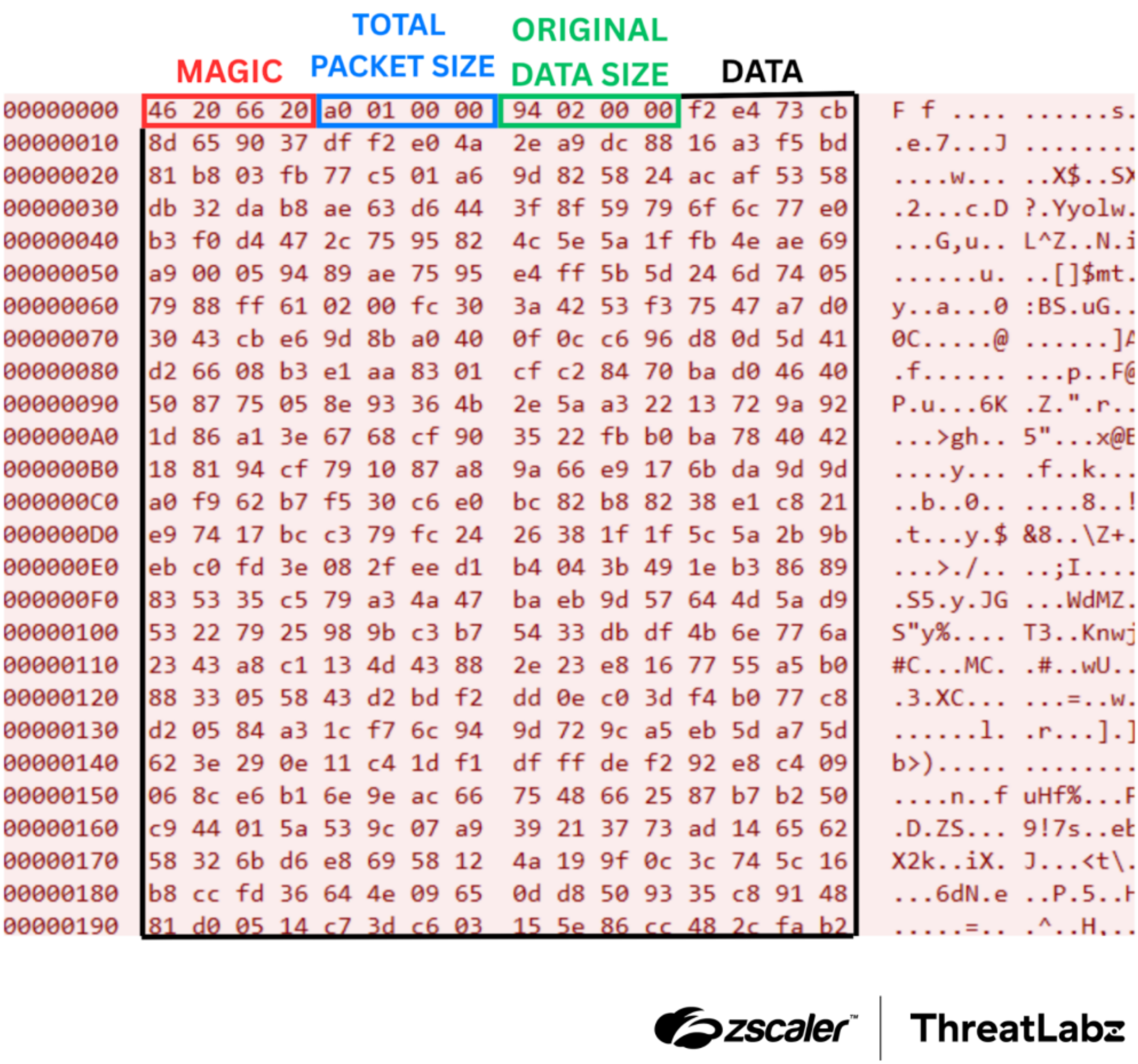

Figure 4: kkRAT packet structure.

The original data is first compressed using *zlib* and then encrypted using an XOR-based algorithm with a key embedded in the malware binary. The Python script provided in the ThreatLabz GitHub repository can be used to decrypt the network data captured.

Plugins

kkRAT retrieves its main plugin and saves it on disk in an encrypted format. When a specific command calls for a plugin export, the encrypted plugin is read from disk, decrypted, loaded into memory, and the requested export is executed. The Python code in the ThreatLabz GitHub repository can be used to decrypt the encrypted plugin. The encryption algorithm is similar to the XOR-based algorithm used to protect network communications.

The table below outlines the plugins and exports for kkRAT.

Plugin Name	Export Name	Description
Main Plugin (`Plugin32.dll`)	`DLLScreen`	Provides basic remote desktop screen management, primarily used for screen capturing and simulating user inputs such as keyboard and mouse actions.
	`DLLScreee`	An extended version of `DLLScreen` that includes additional capabilities, such as retrieving and modifying clipboard data.
	`DLLScreeh`	Enables concealed remote management through virtual desktops, with added functionalities such as launching web browsers and terminating active processes.
	`DllScreer`	Functions as a view-only screen monitor, supporting only screen monitoring without features such as input simulation.
	`DllShell`	Facilitates remote command execution via a shell interface.
	`DllWindows`	Enables management of windows on the screen, offering features such as listing, enabling, disabling, or closing windows.
	`DllProgress`	Provides process management capabilities, including listing active processes and terminating them as needed.
	`DllGetNetState`	Generates a list of active network connections (similar to netstat), along with their associated processes, and allows for the termination of processes based on this data.
	`DllApp`	Offers application management functionalities, including listing installed software and uninstalling selected programs.
	`DllQDXGL`	Enumerates and retrieves the list of values stored in the autorun registry key located at `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`.
	`fnProxy`	Serves as a proxy, facilitating communication between a client and a server by relaying the data.
`PlugProxy.dll`	`ConnSocks`	Functions as a proxy between a client and server, utilizing a Go binary. It implements the SOCKS5 protocol using the go-socks5 library.

Table 1: Plugins supported by kkRAT.

Note that kkRAT's main plugin, *Plugin32.dll*, was uncovered alongside the source code of an older version on [VirusTotal](), which served as the basis for the RAT's name.

After receiving the registration message, the C2 server issues a series of commands for kkRAT to execute. kkRAT supports an extensive range of commands, integrating functionality from its plugin DLL exports. While the known command IDs associated with Ghost RAT are excluded, the table below provides the command IDs for the plugin DLL exports discussed earlier and the new commands introduced in kkRAT.

Command ID	Description
`0x4`	Downloads the main plugin DLL (`Plugin32.dll`).
`0x8`	Removes Internet Explorer browsing data.
`0x9`	Removes Skype local storage data.
`0xA`	Removes Telegram `tdata`.
`0xB`	Removes QQ browser user data.
`0xC`	Removes Firefox profiles data.
`0xD`	Removes Google Chrome user data.
`0xE`	Removes Sogou Explorer cache data.
`0xF`	Removes 360 Speed Browser user data.
`0x10`	Removes 360 Secure Browser user data.
`0x15`	Calls DllScreen export from `Plugin32.dll`.
`0x1F`	Calls DllScreee export from `Plugin32.dll`.
`0x29`	Calls DlScreeh export from `Plugin32.dll`.
`0x2A`	Calls DllScreer export from `Plugin32.dll`.
`0x34`	Calls DllWindows export from `Plugin32.dll`.
`0x35`	Calls DllProgress export from `Plugin32.dll`.
`0x36`	Calls DllGetNetState export from `Plugin32.dll`.
`0x37`	Calls DllApp export from `Plugin32.dll`.
`0x38`	Calls DllQDXGL export from `Plugin32.dll`.
`0x4A`	Establishes persistence on the victim's system. The RAT server provides the sub-command ID and name needed for key/task as parameters to specify the method for persistence. The sub-commands are listed below: • Achieve persistence using the startup folder. • Achieve persistence using autorun key. • Achieve persistence using logon script (`HKCU\Environment\UserInitMprLogonScript`). • Achieve persistence using scheduled tasks.
`0x4B`	Checks for the presence of the GotoHTTP remote monitoring and management (RMM) tool on the victim's system. If GotoHTTP is detected, the command retrieves the `name` and `tmp` values from the *gotohttp.ini* configuration file. If GotoHTTP is not present, the command installs the tool on the system. The GotoHTTP tool (file content) is provided by the C2 as a parameter for the command.

Command ID	Description
`0x4C`	Verifies whether the Sunlogin RMM tool is installed on the victim's system. If Sunlogin is present, the command retrieves the `fastcode` and `password` values from the *config.ini* file. If Sunlogin is not found, the command installs the tool on the system. The Sunlogin RMM tool (file content) is provided by the C2 as a parameter for the command.
`0x4D`	Scans the clipboard for cryptocurrency wallet addresses associated with Tether, Bitcoin, or Ethereum. Identified wallet addresses are replaced with the attacker’s wallet addresses. The attacker’s wallet addresses are provided as parameters for this command.
`0x4E`	Same as `0x4D`.
`0x4F`	Stops the replacement of Tether, Bitcoin, and Ethereum wallet addresses in the clipboard with the attacker’s wallet addresses, effectively disabling the crypto hijacking behavior.
`0x51`	Attempts to elevate privileges on the victim's system using the `runas` verb once.
`0x55`	Invokes the `DllShell` export from the `Plugin32.dll` plugin to execute its associated functionality.
`0x5C`	Calls the `fnProxy` export from the `Plugin32.dll` plugin. This command supports multiple sub-commands, with the first parameter determining the specific operation to be executed. The sub-commands are listed below: • **0x5E:** Establishes a TCP connection to a remote IP and port specified by the attacker. Additional parameters include a unique ID to identify the TCP socket, the target remote IP address, and the target remote port number. • **0x5F:** Terminates the TCP connection associated with the specified ID, which is provided as an additional parameter. • **0x60:** Sends data through the proxy. Additional parameters include the ID of the associated TCP socket and the data to be transmitted.
`0x5D`	Calls the `ConnSocks` export from the `PlugProxy.dll` plugin. Along with this command, the DLL content of `PlugProxy.dll` is provided as a parameter for this command.

Table 2: Commands implemented by kkRAT.