Technical Analysis of SnappyClient | ThreatLabz

By Muhammed Irfan V A

Published: 2026-03-18 · Archived: 2026-04-05 23:37:23 UTC

The following sections focus on SnappyClient's attack chain as well as a technical analysis of the core features.

Attack chain

The figure below shows the SnappyClient attack chain observed by ThreatLabz.

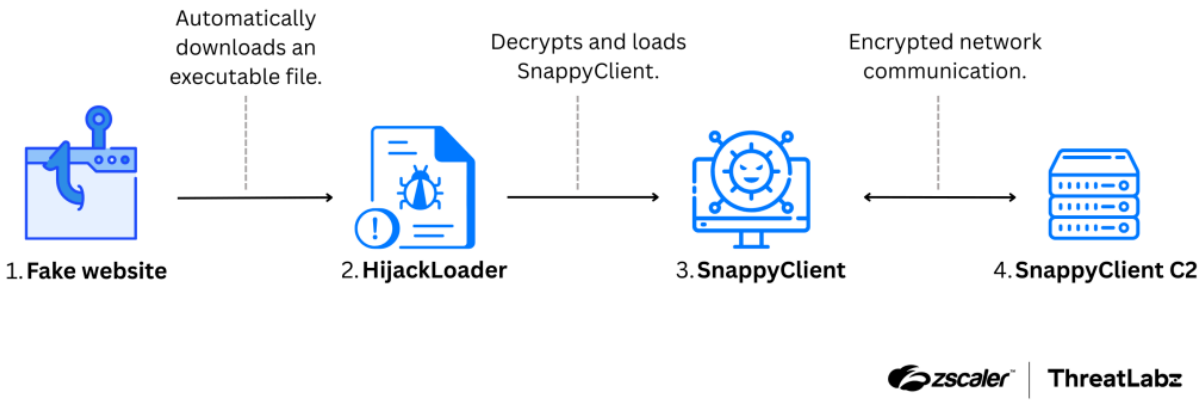

Figure 1: Example attack chain of a campaign delivering SnappyClient.

The attack started with a website that impersonated a telecommunications company and targeted German-speaking users. The page lists product features and branding details. When a victim visits the page, a HijackLoader executable file is automatically downloaded on the victim's system. This HijackLoader sample (if executed by a victim) decrypts and loads SnappyClient.

Additional attack chains have also been observed for SnappyClient delivery. In early February, ThreatLabz observed an X (formerly Twitter) post by @Kostastsale describing a GhostPulse/HijackLoader intrusion via ClickFix, with SnappyClient delivered as the payload.

AMSI bypass

SnappyClient installs a trampoline hook on `LoadLibraryExW` and checks whether the process is loading amsi.dll. If this condition is met, SnappyClient hooks `AmsiScanBuffer` and `AmsiScanString` to always return `AMSI_RESULT_CLEAN`, effectively bypassing AMSI.

SnappyClient configuration

SnappyClient stores the main configuration as a plaintext JSON object embedded in the binary. The table below shows the configuration keys and their usage.

Key Name	**Description**
tag	Malware tag; sent in the registration message.
buildId	Malware build ID; sent in the registration message.
dd	Default directory used by SnappyClient. All folders and files references in the configuration are created under this directory.
ef	Filename of the encrypted EventsDB (described in the next section).
sf	Filename of the encrypted SoftwareDB (described in the next section).
kf	Filename of the encrypted keylogger file capturing the victim's keystrokes.
c	Directory used to check whether the victim's device is banned. SnappyClient retrieves the volume serial number of the root directory and builds a string by concatenating the volume serial number with the string `:BANNED`, then calculates the SHA-1 digest of the string. SnappyClient then checks if a filename with this hash value exists in the directory specified by this c variable under the default directory (`dd`). If the file exists, SnappyClient decrypts the contents by performing an XOR operation with the string `BANNED`. If the decrypted content is the string TRUE, the device is treated as banned and SnappyClient exits.
ab	Directory used to cache the AES-256 master key for Chromium App-Bound Encryption. The file contents are encrypted. The filename is the first 4 bytes of the SHA-256 digest of the app_bound_encrypted_key value stored in the Local State file.
e	If set to True, SnappyClient creates a named shared memory and writes the malware version number at the start of the region. If there is a version already running and the currently running version number is less than or equal, the process exits. If the currently running version is greater,

Key Name	Description
	the version number in the named shared memory is updated and the older instance will terminate. This ensures the latest version of SnappyClient is always running. The shared memory name is generated by creating a string in the format: {COMPUTERNAME}{USERNAME}. The string is then reversed and its FNV-1a 32-bit hash is calculated, which is then XOR'ed with the string length. The result is converted to decimal representation and used as the shared memory name.
m	Contains the mutex name. If a mutex with this name already exists, SnappyClient exits.
s	If set to True, the configuration contains additional fields (si, sm, and sn) used for persistence and installation.
si	Specifies the installation path where SnappyClient copies itself. After the copy completes, SnappyClient launches a new process from the copied file and terminates the original process.
sm	If the value is 1, SnappyClient first attempts to establish persistence using scheduled tasks. If this fails, SnappyClient attempts to establish persistence using the registry. If the value is anything else, SnappyClient only attempts to establish persistence using scheduled tasks. For scheduled tasks, SnappyClient triggers on logon of the current user and the path is set to the current process path. For registry persistence, SnappyClient uses the autorun key Software\Microsoft\Windows\CurrentVersion\Run.
sn	Name of the scheduled task and the registry name.

Table 1: Description of SnappyClient's embedded configuration.

After parsing the main configuration, SnappyClient parses another configuration. This configuration is also a JSON object with a single key: `pairs`. The `pairs` key contains a list of Class Identifiers (CLSIDs) and Interface Identifiers (IIDs) used in Chromium App-Bound Encryption. Each list item includes three properties, and all property values are Base64-encoded. The properties are:

- `name`: Name of the browser.
- `c`: Elevator CLSID.
- `i`: IID of `IElevator` interface.

There are two configurations that are also retrieved from SnappyClient's C2 that are named `EventsDB` and `SoftwareDB`. These configurations are written to disk encrypted using the ChaCha20 cipher. Each file can contain multiple streams of encrypted data with different keys. To separate multiple streams, SnappyClient adds a header to the start of each encrypted stream. The structure of the header is shown below.

```
 struct Header {
   DWORD MAGIC1;     // Used to find the start of an encrypted stream.
   DWORD MAGIC2;     // Used to find the start of an encrypted stream.
   DWORD MAGIC3;     // Used to find the start of an encrypted stream.
   BYTE  key[0x20];  // Key used in the ChaCha20 cipher.
   BYTE  nonce[0xC]; // Nonce used in the ChaCha20 cipher.
   DWORD crc_value;  // CRC value of the header excluding crc_value.
};
```

The Python function below demonstrates the use of the MAGIC bytes to identify the start of the encrypted stream.

```
 import struct
import binascii

K0 = 0xCEDD9AB7
K1 = 0x7FCBB9E9
HEADER_LEN = 0x3C

def is_header_valid(header_bytes: bytes, k0: int = K0, k1: int = K1) -> bool:
    if len(header_bytes) != HEADER_LEN:
        return False

    data = struct.unpack("15I", header_bytes) #little-endian
    MAGIC1, MAGIC2, MAGIC3, crc_value_stored = data[0], data[1], data[2], data[14]

    condition_1 = ((MAGIC2 ^ k0) & 0xFFFFFFFF) == ((~MAGIC1) & 0xFFFFFFFF)
    if not condition_1:
        return False

    condition_2 = ((k1 ^ MAGIC3) & 0xFFFFFFFF) == MAGIC2
    if not condition_2:
        return False

    crc_value = binascii.crc32(header_bytes[:0x38])
    return crc_value_stored == crc_value
```

The Python script to decrypt these configuration files is available in the ThreatLabz Github repository.

EventsDB

The EventsDB file is a list of events sent by the C2 that perform a particular action when a condition is met. Each event is a JSON object. The table below shows the keys in an event and their usage.

Key Name	**Description**
id	Event ID; sent by the C2.
hash	Event hash; sent by the C2.
f1	Base64-encoded regular expression used to check clipboard content (internal trigger type **4**). If the pattern matches, SnappyClient executes the configured action.
f2	Base64-encoded, action-dependent value (see the following row): for action 64, it contains the replacement clipboard content; for action 8912, it contains the exfiltration URL.
action	Action(s) to perform when the condition is met; multiple values may be combined using bitwise OR. Supported values include: • **64 (0x40):** Replace clipboard content (if it matches f1) with f2. • **384 (0x180):** Take a screenshot of the foreground window, convert it to a JPEG image, and send it to the C2. • **8912 (0x2000):** Exfiltrate clipboard data over HTTP (if it matches f1). In this case, f2 contains the URL used for exfiltration and may include placeholders that are replaced with the appropriate values: `$(name)` is replaced with a string concatenating the computer name and username, `$(systemid)` is replaced with the victim machine's system ID (explained in a later section), and `$(clipboard)` is replaced with the clipboard content.
type	Internal trigger type. There are two supported event trigger types: • **3:** Check filters against the window title. • **4:** Check filters against the clipboard's content.
target	Target type for the event. The event is only registered if the victim device's target type matches the target filter:

Key Name	Description
	• **0:** Only register the event if the target filter matches {COMPUTERNAME} {USERNAME}. • **1:** Only register the event if the target filter matches the computer name. • **2:** Only register the event if the target filter matches the username. • **3:** Register the event regardless of the target filter string. This value is used to register an event for a particular victim.
filter	Target filter used to check against the target type.
condition	Specifies how to match the target filter against the target type. Supported values include: • **0:** Use regex matching. • **1:** Use case-insensitive wildcard string matching (supports * and ? only). • **2:** Use case-insensitive string matching.
wndfilter	Base64-encoded filter evaluated against the window title (internal trigger type **3**) if it matches, the configured action is executed.
wndcondition	Specifies how to match wndfilter against the window title. Supported values include: • **0:** Use case-insensitive wildcard string matching (supports * and ? only). • **2:** Use regex matching.
tags	List of CRC tag hashes compared to CRC of the tag value in the main configuration. The event is registered only if a hash matches, or if tags is 0 (no tag filtering), enabling tag-scoped targeting.

Table 2: Description of the SnappyClient EventsDB configuration file.

SoftwareDB

The SoftwareDB file is a list of software entries and their properties that the C2 sends to SnappyClient, which the malware then uses to steal data. Each software entry is stored as a JSON object. SnappyClient targets software

applications for data theft, which are listed in the table below. All values, except `engine` and `ids`, are Base64-encoded.

Software Type	Key Name	Description
Browser	name	Name of the browser.
	data_path	Base directory where the browser stores per-user data.
	engine	Browser engine type indicating whether the browser is Chromium-based or Mozilla-based.
	profile_regex	Regex used to identify profile subfolders under data_path.
	process_name	Executable filename for the browser.
Extension	name	Name of the browser extension.
	ids	ID of the browser extension.
	engine	Browser engine type.
Other Application	name	Name of the application.
	search_path	Base application directory; SnappyClient steals files located in this directory.

Table 3: Description of the SnappyClient SoftwareDB configuration file.

An example of the decrypted EventsDB and SoftwareDB is available in the ThreatLabz Github repository.

Network configuration decryption

SnappyClient's also contains an encrypted network configuration. The network configuration decryption is a convoluted process that uses a combination of ChaCha20-Poly1305, SHA1, SHA256, modified RIPEMD-160 (Compared to standard RIPEMD-160, it inserts one additional compression step per block after step 31), Snappy compression, and Base58 encoding. The complete network configuration decryption script, including all helper functions, is available in the ThreatLabz GitHub repository.

The Python function below replicates the algorithm to decrypt the SnappyClient network configuration.

```
def decrypt_config(filename: str) -> tuple[bytes, bytes]:
   with open(filename, 'rb') as f:
       compressed_data = f.read()
   try:
       uncompressed_data = snappy.uncompress(compressed_data)
   except:
       print("decompression failed")
       exit()
   enc_data_size = struct.unpack('H',uncompressed_data[0:2])[0] #little-endian
   context_size =  struct.unpack('H',uncompressed_data[2:4])[0] #little-endian
   enc_content_outputtag = uncompressed_data[4:enc_data_size+4]
   enc_content = uncompressed_data[4:enc_data_size+4-16]
   output_tag = uncompressed_data[enc_data_size+4-16:enc_data_size+4]
   context_content = uncompressed_data[enc_data_size+4:enc_data_size+4+context_size]
   keya, noncea = generatekey_nonce(context_content)
   noncea = noncea[:0xc]
   context_content_compressed = snappy.compress(context_content)
   ct_and_tag, ciphertext, tag = ChaCha20Poly1305encrypt(keya, noncea, context_content_compressed)
   ct_and_tagb58 = base58.b58encode(ct_and_tag)
   ct_and_tagb58_ripemdmodb58 = base58.b58encode(ripemd160_modified(ct_and_tagb58))
   tag_ripemdmodb58 = base58.b58encode(ripemd160_modified(tag))
   keyb = hashlib.sha256(ct_and_tagb58_ripemdmodb58).digest()
   nonceb = hashlib.sha256(tag_ripemdmodb58).digest()
   nonceb = nonceb[:0xc]
   key_seedcompressed = ChaCha20Poly1305decrypt(keyb, nonceb, enc_content, output_tag)
   key_seed = snappy.uncompress(key_seedcompressed)
   keyc, noncec = generatekey_nonce(key_seed)
   enc_content2 = key_seed[:-16]
   tag2 = key_seed[-16:]
   k7z_sig = b'\x37\x7A\xBC\xAF\x27\x1C\x00\x04'
   k7z_data = k7z_sig + context_content[8:]
   key_seedripedmdmod_b58 = base58.b58encode(ripemd160_modified(key_seed))
   key_seedripedmdmod_b58_keyseed=key_seedripedmdmod_b58+key_seed
   archive_path_ip = base58.b58encode(ripemd160_modified(key_seedripedmdmod_b58_keyseed))
   keyd_nonced = extract_7z_memory(k7z_data, key_seed, archive_path_ip)
   keyd = keyd_nonced[:0x20]
   nonced =  keyd_nonced[0x20:]
   enc_tag = base58.b58decode(key_seed)
```

```
    encd = enc_tag[:0x20]
    tagd = enc_tag[0x20:]
    ip = ChaCha20Poly1305decrypt(keyd, nonced, encd, tagd)
    ip = snappy.uncompress(ip)

    iphashobject = hashlib.sha1(ip)
    iphash = iphashobject.digest().hex().encode("utf-8")
    ipmod = base58.b58encode(ripemd160_modified(ip))
    ipmod_keyseed = ipmod+key_seed
    archive_path_port = base58.b58encode(ripemd160_modified(ipmod_keyseed))
    port_string = extract_7z_memory(k7z_data, key_seed, archive_path_port)

    return ip, port_string
```

The decrypted network configuration contains one or more C2 IP addresses separated by semi-colons and a JSON object with two ports for communication:

- **p:** Control port. This port is used for the control session, which is the first session created by SnappyClient. Victim registration occurs over this session, and SnappyClient receives initial commands through it.
- **dp:** Data port. This port is used for data sessions. For example, when SnappyClient sends a file to the C2, it establishes a data session using this port and transfers the data. The C2 can also instruct SnappyClient to create a data session by sending the respective command through the control session.

SnappyClient has only one control session, but it can create multiple data sessions as required.

Network communication protocol

SnappyClient uses a custom network communication protocol over TCP for its control session. SnappyClient first establishes a connection to the C2 server and receives a packet from the C2, which contains a ChaCha20 key and nonce used for encryption. The packet has the following structure:

```
 struct first_packet {
   BYTE  key[0x20];          // Key used in ChaCha20-Poly1305 to encrypt messages.
   BYTE  nonce[0xC];         // Nonce used in ChaCha20-Poly1305.
   DWORD controlsession_id;  // ID of the control session.
};
```

SnappyClient replies by encrypting the key using the same key and nonce sent by the C2, and appending the output tag after the encrypted bytes. This ensures the C2 can successfully decrypt communications from SnappyClient.

All plaintext messages exchanged between the C2 and SnappyClient are JSON objects. Before transmission, messages are compressed using Snappy and then encrypted using ChaCha20-Poly1305. Each message is preceded by a message header with the following structure:

```
struct message_header {
  WORD  command_ID;       // Command ID of the message.
  DWORD message_id;       // Unique ID for each message; generated using Mersenne Twister.
  DWORD unknown;          // Always set to 1 by SnappyClient.
  DWORD message_length;   // Message length before compression.
  BYTE  zero_bytes[0x8];  // Set to zero at the end of all message headers.
};
```

The message header is also encrypted using ChaCha20-Poly1305. The output tag is appended after the encrypted message header and sent to the C2.

After sending the message header, SnappyClient sends the message. Three bytes are added to the start of the encrypted message:

- The first two bytes represent the message size after compression (plus the output tag size if the third byte is set to 1).
- The third byte is a flag. If set to 1, the output tag is appended after the encrypted message.

The image below shows an example of SnappyClient's network communication that occurs over the control session.

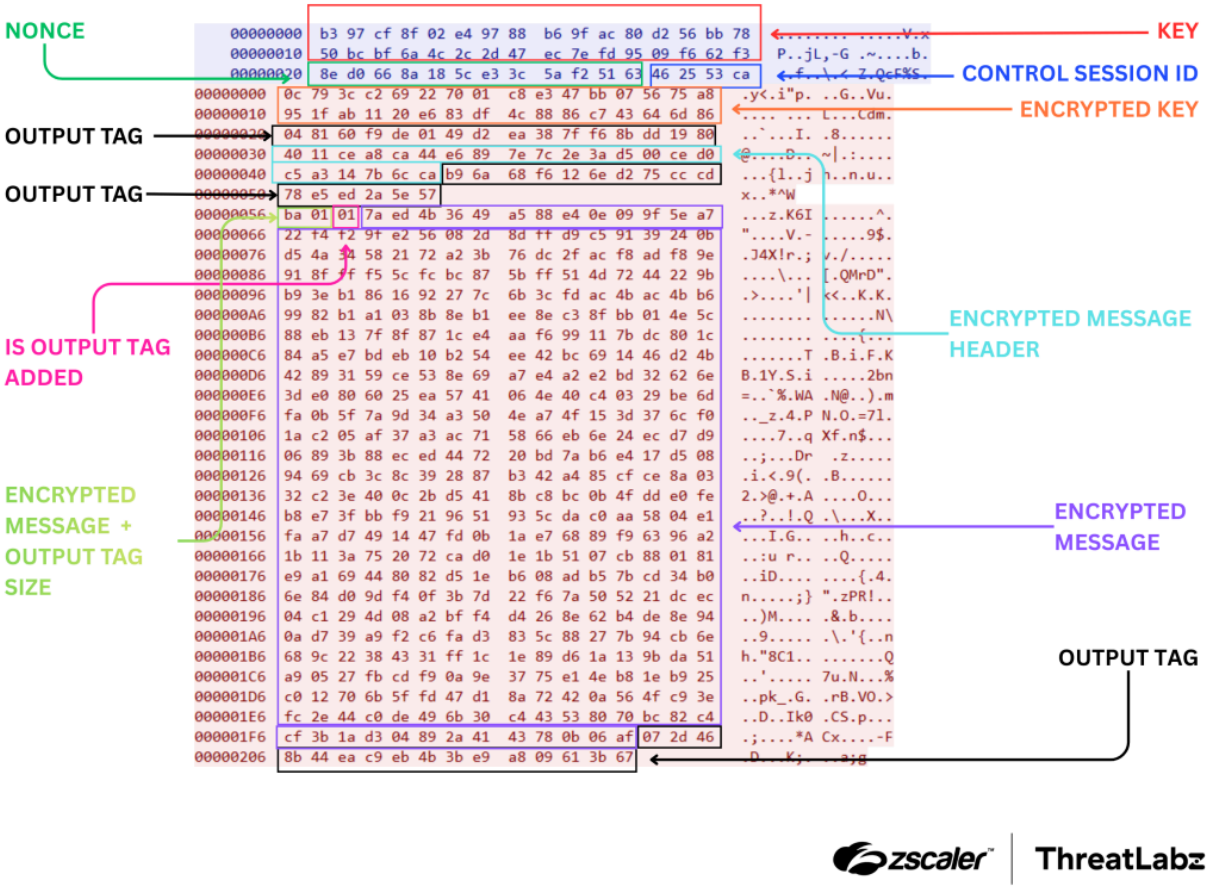

Figure 2: Example SnappyClient network communication in the control session.

The data session follows a similar communication protocol to the control session, with two differences:

- SnappyClient sends a unique `datasession_id` as the first packet for each session to inform the C2 that the session corresponds to the specified data session.
- The `controlsession_id` sent with the key and nonce is set to zero for the data session since it is not a control session.

Registration message

The first message sent by SnappyClient is a registration message, which contains victim information. The `command_ID` for the registration message is 0xFFFF. The table below shows the keys used in the registration message JSON object.

Key Name	**Description**
computer	Victim's computer name; Base64-encoded.
username	Victim's username; Base64-encoded.
window	Foreground window title; Base64-encoded.
wv	Operating system Windows version.
rc	Set to 1 if the control session was reset. If the reset count is 0, set rc to 0; otherwise, set it to 1. The reset count is incremented when the TCP session is reset.
tt	Time elapsed (in milliseconds) since the system was started.
ut	Time elapsed (in milliseconds) from the start of SnappyClient's execution until registration.
it	Time elapsed (in milliseconds) since the last input event.

Key Name	Description
ram	Victim system total physical memory.
uac	TokenElevationType of the process.
cpu	Victim system processor count.
ver	Malware version number as a string. The version analyzed was 0.1.11.
iver	Malware version number in decimal representation.
ts	Current system time calculated using _Xtime_get_ticks.
cp	Control port used by SnappyClient.
dp	Data port used by SnappyClient.
sync_events	Combined hash of events in EventsDB. SnappyClient serializes event fields into a single contiguous buffer, computes a SHA-1 hash over the buffer, and uses the first 4 bytes of the digest as the combined hash. If no events are in EventsDB, the value is 0. This allows the C2 to quickly identify which events are currently in EventsDB configuration.
sync_software	Combined hash of software applications in SoftwareDB. SnappyClient calculates a hash for each software type (browser, extension, and other application) using CRC32 over the respective fields. These three values are then XOR’ed to produce the combined hash. If no software is in SoftwareDB, the value is 0. This allows the C2 to quickly identify which software is currently in the SoftwareDB configuration.
software	Base64-encoded list of installed applications on the system from the SoftwareDB configuration.

Key Name	Description
sid	The system ID of the victim's machine. This unique ID is generated using the volume serial number of the root directory, the CPU signature (collected using CPUID with EAX set to 1), the computer name, and the username. The function to generate the system ID is available in the ThreatLabz Github repository.
tag	Malware tag label from the main configuration.
buildId	Malware build ID from the main configuration.
av	Base64-encoded list of installed antivirus products on the victim's system.
monitors	List of display monitors on the system. For each monitor, a JSON object is created with the following keys: • **name:** Name of the monitor (Base64-encoded). • **width:** Width of the monitor. • **height:** Height of the monitor. • **rate:** Display refresh rate of the monitor.

Table 4: Data collected by SnappyClient as part of the registration message.

Command messages

After SnappyClient sends the registration message, the C2 will respond with command messages. Command messages before encryption are also JSON objects and include a message header.

Depending on the command ID, the following keys may be present in the message:

- **id:** The `controlsession_id` of the network communication.
- **sid:** The `datasession_id` of the network communication.
- **frameid:** This value is not currently parsed on the client side, but the value is the same across multiple data sessions.

The `command_ID` in the header determines which commands to process. Each command message may include additional arguments. The table below lists the commands supported by SnappyClient and their additional

arguments. The **Type** column contains the value of the type key in the command message, which is used as a sub-command ID.

Command ID	Type/Sub-Command ID	Additional Arguments and Description
0xCCCE	1: Screenshot Grabber	**monitor:** Monitor to capture the screenshot from. **quality:** Quality of the JPEG image (as an integer).

	2: Process Manager	action: Type of action to perform.
		• **0** or **2**: Get a list of all processes currently running on the system, along with their process IDs (PIDs).
		• **3**: Perform a process action on the processes with the specified PIDs.
		  ◦ **1**: Does nothing.
		  ◦ **2** or **3**: Suspend the process (suspend all threads).
		  ◦ **4**: Resume the process (resume all threads).
		  ◦ **5** or **6**: Terminate the process.

	3: File / Directory Operation	**subaction:** Name of the action to perform. Supported values include:
		• **browseabs:** Read a directory path from the **path** key's value and list files under that directory.
		• **archive:** Archive files based on the **data** key's value. The data key contains:
		  ◦ archivetype (archive format, it uses the 7-Zip library for compression and supports numerous archive formats)
		  ◦ base (base directory prepended to each item's name)
		  ◦ items[] (an array of entries to include in the archive)
		  ◦ items[].name (relative path of the files to compress in the base directory)
		  ◦ outname (output archive path)
		  ◦ p (password string used to encrypt the archive)

Command ID	Type/Sub-Command ID	Additional Arguments and Description

		• **extract:** Extract a file based on the **data** key. The **data** key contains:
		  ◦ path (full path to the archive)
		  ◦ p (password)
		• **recursive:** Recursively copy contents under a directory to a new directory. The source (items[].source) and destination (**dest**) are provided in the **data** key.
		• **recursivedel:** Recursively delete contents under a directory. The path (items[].path) to delete is provided in the **data** key.
		• **rename:** Rename a file or folder on disk from the **target** key to the **new** key.
		• **xs:** Validate shortcuts provided in the **shortcuts** argument. If a shortcut does not exist, report it back to the C2. The shortcut path to check is in shortcuts[].path.
		• **quick:** Execute a file specified in the data.path argument using ShellExecuteW.
		• **newfolder:** Create a new folder. The folder path is provided in the **name** argument.

	4: Exfiltrate Keylogger File	Send the keylogger file path and size to the C2.
	5: Browser Password Stealer	Send saved browser passwords to the C2. Additional arguments include the keys described in SoftwareDB, plus **logins_file**, which contains a regex used to match the login database file (Login Data).
	6: Browser Cookies Stealer	Send browser cookies to the C2. Additional arguments include the keys described in SoftwareDB, plus **cookies_file**, which contains a regex used to match the cookies database file (Cookies).
	7: Clone Browser	Clone browser profile artifacts such as History, Preferences, Bookmarks/Favicons, Top Sites/Visited Links, Web Data, and

Command ID	Type/Sub-Command ID	Additional Arguments and Description
		Shortcuts. It also clones other data such as Local Storage, Session Storage, Sessions, Network, Sync Data, Extension Rules, and Local Extension Scripts. Additional arguments include the keys in SoftwareDB, plus **cookies_file** and **logins_file**.
	8: Steal Browser Extension Data	Additional arguments include the keys described in SoftwareDB.
	9: Steal Other Application Data	Additional arguments include the keys described in SoftwareDB, plus **include_filter** and **exclude_filter**. • **include_filter:** Regex used to select files under **search_path** for collection. • **exclude_filter:** Regex used to exclude files under **search_path** from collection.
	10: Execute File	**a:** Execution type. Supported values include: • **0:** Execute the file directly. Additional arguments include path (path of the file to execute). • **1:** Execute the file as a DLL using rundll32.exe. • **2:** Extract an archive and execute the file inside it. Additional arguments include **path** (archive path), **arche** (name of the executable to run after extraction), and **archp** (archive password). Additional arguments applicable to all execution types (used with CreateProcessW): • **cmd:** lpCommandLine for the created process. For DLL execution, cmd includes the path to the DLL. • **cd:** lpCurrentDirectory for the process. • **flags:** dwCreationFlags for the process. • **d:** Desktop name set using STARTUPINFOW.lpDesktop.

Command ID	Type/Sub-Command ID	Additional Arguments and Description
		• **taskFlags:** If set to 1, bypass UAC using the CMSTPLUA COM interface with the elevation moniker Elevation:Administrator!new:{3E5FC7F9-9A51-4367-9063-A120244FBEC7} and the ShellExec function.
	12: Hidden VNC Browser	**action:** Type of action to perform. • **1:** Launch HvncBrowser. Additional arguments include keys in SoftwareDB. • **0x6E (110):** Migrate browser profile data from the source key to the dst key.
	14: Remote File Browser	**action:** Type of action to perform. • **0:** Start a remote file browser for each drive and list the contents of each directory.
	15: Remote Shell	**action:** Type of action to perform. • **0:** Initialize the shell. • **2:** Execute the command in the data key. • **4:** Terminate the shell.
0xCCCC	0: Set up a reverse FTP proxy which forwards requests to an internal hidden FTP server on the victim machine controlled by the malware, allowing the C2 to exfiltrate files from the local filesystem.	• **controlport:** Port used to control the proxy. • **tunnelport:** Port through which proxied data is sent.
	1: Set up a reverse VNC proxy which forwards	

Command ID	Type/Sub-Command ID	Additional Arguments and Description
	requests to an internal hidden VNC server on the victim machine controlled by the malware, providing the C2 with graphical remote control.	
	2: Set up a reverse RLOGIN proxy which forwards requests to an internal hidden RLOGIN server on the victim machine controlled by the malware, granting the C2 command-line access..	
	4: Set up a reverse SOCKS5 proxy which forwards requests to an internal hidden SOCKS5 server on the victim's machine controlled by the malware, enabling the C2 to relay traffic through the victim machine.	
0xDDDD	N/A	No type value for this command_ID. Used to start and stop data sessions. **action:** Type of action to perform. • **0:** Start a data session. Additional arguments include sid (datasession_id of the new data session). • **1:** Stop a data session. Additional arguments include sid (datasession_id of the session to stop).

Command ID	Type/Sub-Command ID	Additional Arguments and Description
0xDCCA	3: Send Files Or Folders	**path:** Path of the file/folder to send. If the path is a folder, all files under the folder are sent.
	N/A	**kl:** If the **kl** key is present in the command message, send the keylogger file.
0xDACC	The type field can contain multiple values combined using bitwise OR operations.	N/A
	0x200: Multi-Browser Credential Stealer	Contains a list of browsers to steal data from. Additional arguments include the keys described in SoftwareDB.
	0x800: Multi-Software Credential Stealer	Contains a list of software to steal data from. Additional arguments include the keys described in SoftwareDB, plus include_filter and exclude_filter. • **include_filter:** Regex used to select files under **search_path** for collection. • **exclude_filter:** Regex used to exclude files under **search_path** from collection.
	0x400: Send Keylogger File	No additional arguments.
	0x7: Download Job	• **url:** URL to download the file from. • **path:** Path to save the file to. • **ua:** User-Agent to use. • **hdrs:** Additional headers to add when downloading the file.
	0x1000: File Search Job	The data key contains a list of items to search for, with the following properties:

Command ID	Type/Sub-Command ID	Additional Arguments and Description
		• **filter:** Filter string to search for. • **condition:** How to use the filter for searching. • **0:** Case-insensitive wildcard string match (supports * and ? only). • **2:** Regex matching. If there is a match, report the file path back to the C2.
	0x40: Replace Clipboard Contents	Replace the clipboard content with the value of the data key.
0xDADA	N/A	Does not depend on type. Same as the download job (command_ID 0xDACC with type 0x7).
0xEECC	0: Sync Software To SoftwareDB	Update the encrypted SoftwareDB on disk based on additional arguments, then process the updated SoftwareDB. Additional arguments include: • **hash:** Combined hash of the software to check whether it has already been registered. • **sync_software:** List of software to sync.
	1: Sync Events To EventsDB	Update the encrypted EventsDB on disk based on additional arguments, then process the updated EventsDB. Additional arguments include: • **hash:** Combined hash of the events to check whether they have already been registered. • **sync_events:** List of events to sync.
0xACCC	1: Exit	Exit SnappyClient. No additional arguments are required.
	3: Ban And Exit	Creates a file on disk that marks the SnappyClient infection as banned (as described in the main configuration section) and

Command ID	Type/Sub-Command ID	Additional Arguments and Description
		exits.

0xADBB	1: Create IWebBrowser Window Or Create A MessageBox	The additional information is provided in the data key. The data properties include:
		
		type: Subtype of the command. Supported values are:
		
		• **0:** Create a MessageBox. Additional arguments include:
		  ◦ **0**: Message box caption.
		  ◦ **1**: Message box text.
		  ◦ **w**: Width of the MessageBox.
		  ◦ **h**: Height of the MessageBox.
		• **1:** Create a window and embed an IWebBrowser control. Additional arguments include:
		  ◦ **0**: Window title.
		  ◦ **4**: Content to render in the window.
		  ◦ **w**: Width of the window.
		  ◦ **h**: Height of the window.

	2: Update Network Configuration	The updated configuration is provided in the data key. The properties inside data are: • **h:** Updated C2 IP. • **p:** Updated control-session port. • **dp:** Updated data-session port.

Table 5: Description of SnappyClient commands.

Process injection

SnappyClient's process injection technique uses code similar to HijackLoader. To evade user-mode API hooks when invoking certain native APIs, SnappyClient uses Heaven's Gate to execute x64 direct system calls. For more details, refer to our earlier previous ThreatLabz blogs: HijackLoader Updates and Analyzing New HijackLoader Evasion Tactics.

To bypass Chromium's App-Bound Encryption, the `IElevator` COM interface must be instantiated from a trusted process. To accomplish this, SnappyClient uses transacted hollowing (with code similar to HijackLoader)

to inject a payload, which retrieves Chromium's AES_256 master key. Therefore, SnappyClient can exfiltrate browser data from Chromium-based browsers.

Post-infection activities

To identify SnappyClient's goal, ThreatLabz decrypted the malware's network communications. The activity indicates a financial motive, with cryptocurrency theft as the primary goal. Below is a list of events and software the malware registers.

Registered events

- If the clipboard content matches the regex `^0x[a-fA-F0-9]{40}$` (an Ethereum wallet address), perform action 384 (takes a screenshot and sends it to the C2).
- If the window title matches the regex `(binance|coinbase|exodus \d{1,2}\.|atomic wallet)`, perform action 384 (takes a screenshot and sends it to the C2).

Registered software

- **Browsers:** 360Browser, Opera, Chrome, CocCoc, Edge, Firefox, Slimjet, Vivaldi, Waterfox, and Brave.
- **Extensions:** Coinbase, Metamask, Phantom, TronLink, and TrustWallet.
- **Other applications:** Atomic, BitcoinCore, Coinomi, Electrum, Exodus, LedgerLive, TrezorSuite, and Wasabi.

Potential ties to HijackLoader

ThreatLabz observed potential links between HijackLoader and SnappyClient. HijackLoader is commonly used in eCrime campaigns. The code similarities we identified include the following:

- **API structure layout:** SnappyClient's API structure closely matches HijackLoader's, with an almost one-to-one mapping. It also includes placeholder (empty) DWORD values for APIs that SnappyClient does not use. The figure below shows the API structure layout in IDA for both families.

Figure 3: API structure layout of HijackLoader and SnappyClient.

- **Direct system calls and 64-bit ntdll mapping:** Both HijackLoader and SnappyClient use similar code to populate their direct-syscall structures and to map a 64-bit copy of ntdll into memory. The figure below shows the code used by both to populate the syscall structure.

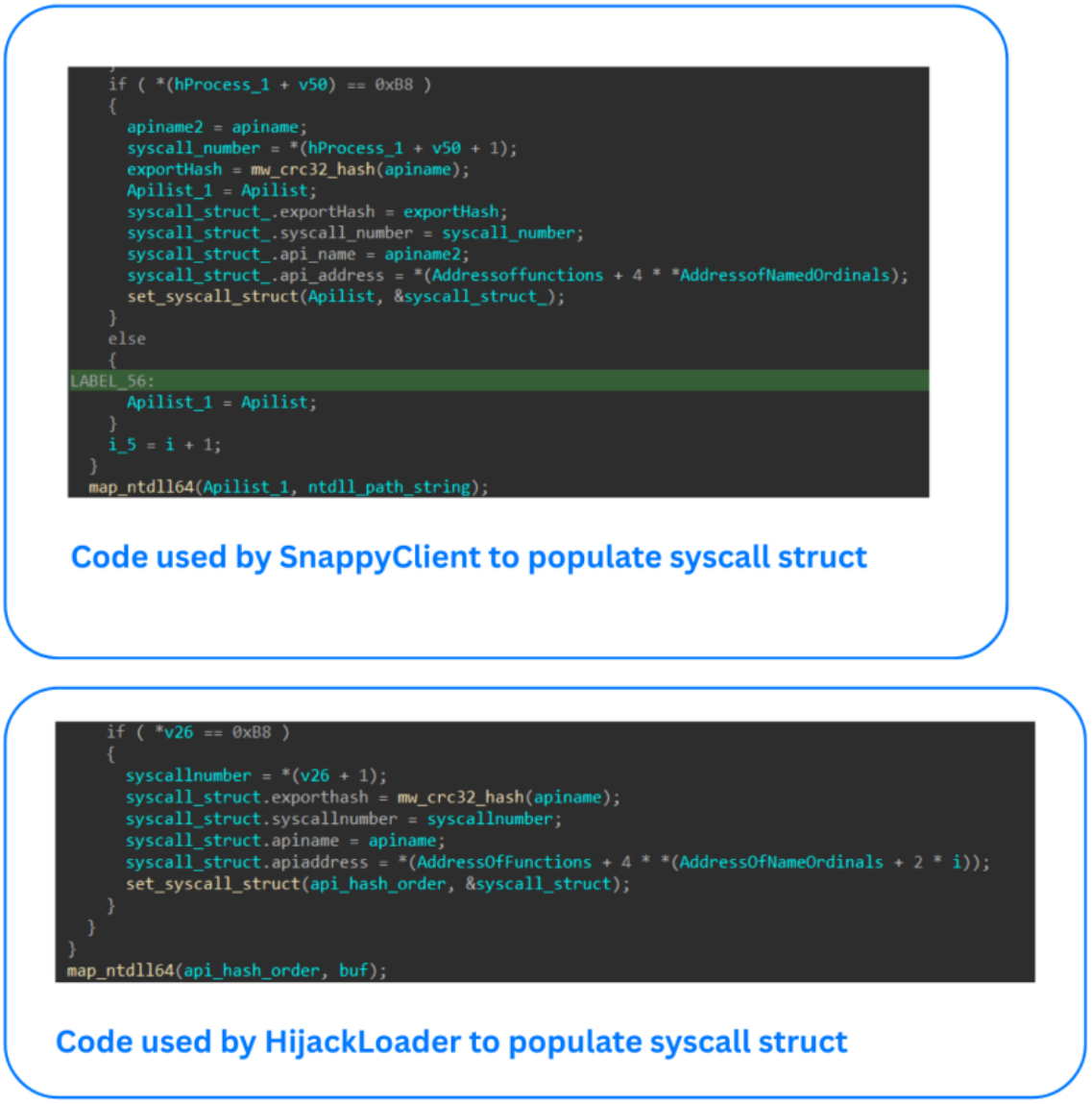

Figure 4: Code used by HijackLoader and SnappyClient to populate a syscall structure.

- **Transacted hollowing:** Both families use similar transacted-hollowing code to inject payloads into a remote process.

In addition, across all campaigns we have observed to date, HijackLoader has been the exclusive loader used to deploy SnappyClient. Based on these overlaps, there may be a connection between the developers of HijackLoader and SnappyClient.

Source: https://www.zscaler.com/blogs/security-research/technical-analysis-snappyclient