

Salat Stealer Analysis: Go-Based RAT, C2 Resilience, and Info-Stealing Capabilities

darkatlas.io/blog/salat-stealer-analysis-go-based-rat-c2-resilience-and-info-stealing-capabilities

Dark Atlas Team

May 6, 2026



Blog



May 06, 2026



10 min read



Darkatlas Squad



Share

Executive Overview

Salat Stealer is a sophisticated **Go-based Remote Access Trojan (RAT)** with deep information-stealing capabilities. Rather than acting as a simple stealer, it functions as a **full post-exploitation framework** with features that include WebSocket/QUIC command-and-control (C2), remote shell access, desktop and webcam streaming, browser and crypto-wallet theft, keylogging, clipboard theft, and SOCKS5 pivoting.

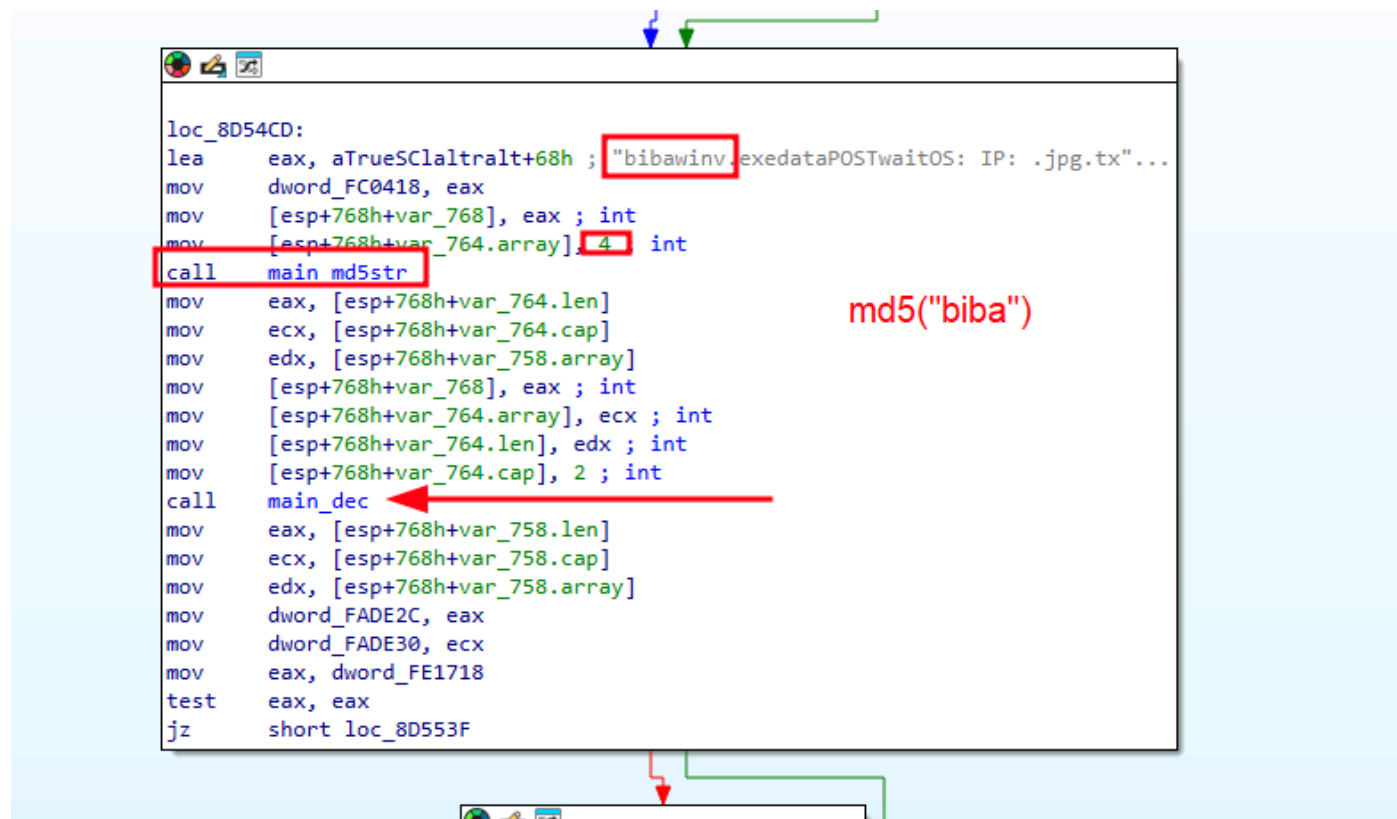
The malware also implements a **six-mode string obfuscation scheme** and a **per-machine key derivation chain** that binds parts of its decryption process to the victim's hostname and hardware profile. This design increases resilience and complicates static analysis.

Technical Analysis

At the beginning of execution, the malware calls `os.Executable()` to retrieve its own file path on disk. This path is stored in a global variable and later reused for several purposes:

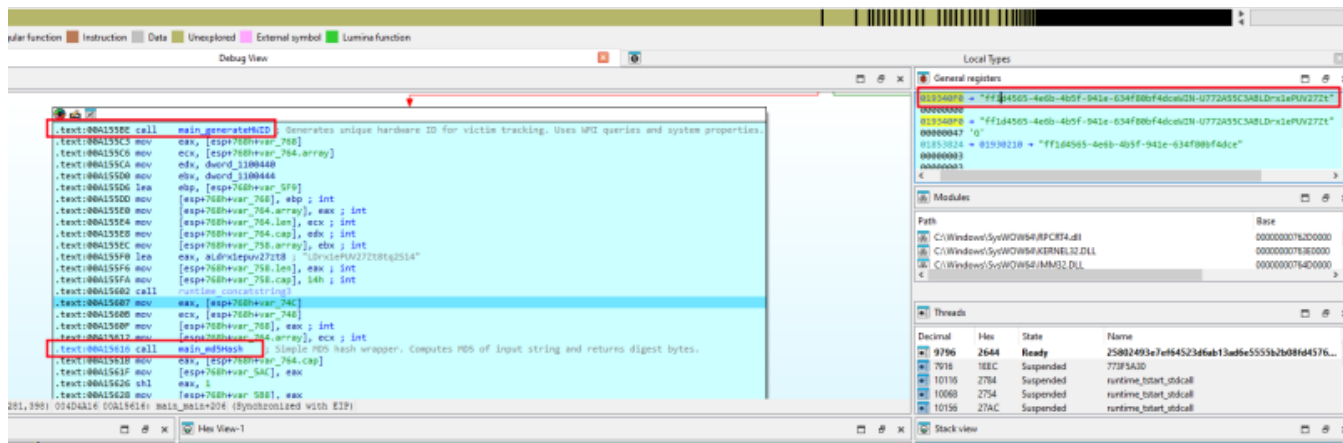
- locating the running process for injection checks
- self-replication during persistence installation
- deriving cryptographic material from the executable's contents

The malware derives its first decryption key by applying an **MD5 hash** to the string `"biba"`.



This key is then used as the **AES-128-GCM key** for **mode 1** of its six decryption modes. Another notable key is the hex representation of the Russian phrase `"Я люблю сосать"`, which is used as the **XOR key** for **mode 3**.

Later, the malware generates a unique victim fingerprint by concatenating the hostname, HWID, and a hardcoded salt string `LDrx1ePUV27Zt8tq2S14`, then hashing them again with MD5. The resulting 16-byte digest is converted into a 32-character hexadecimal string and used as the **agent ID**. This ID uniquely identifies the victim to the C2 server and is embedded in every beacon.



“renamed several functions for clearer identifications”

The malware also attempts privilege escalation by calling `main_Elevate`, aiming to relaunch itself with administrator rights if possible. This enables broader access to the infected system.

Runtime Mode Branching

Salat Stealer checks the length of `os.Args` and branches into multiple runtime modes:

Args	Behavior
<code>argv[1] == "-k"</code>	Keylogger-only mode. Calls <code>main.runKeylogger()</code> and exits.
<code>len(argv) == 3</code>	Command-executor mode. which spawns a subprocess, pipes a JSON-encoded command array to stdin, executes it, and exits
<code>len(argv) == 4</code>	Mutex-wait mode. Which creates a named mutex, starts a background goroutine, then enters a <code>select</code> loop with a 1-second timer waiting on a channel.
Default	Full RAT mode.

This branching shows that Salat Stealer can operate in multiple specialized modes rather than following a single fixed execution path.

Keylogging Capabilities

The malware includes embedded **keylogging functionality** to capture keystrokes from the victim system. This allows operators to collect credentials, messages, and other sensitive input beyond what is stored in browsers or applications

```

4      (int)&off_D3190C,
5      1);
6      p_4_uintptr = (_4_uintptr *)runtime_newobject((int)&RTYPE__4_uintptr);
7      (*p_4_uintptr)[0] = 13;          // 13 -> WH_KEYBOARD_LL
8      (*p_4_uintptr)[1] = v18;
9      (*p_4_uintptr)[2] = 0;
10     (*p_4_uintptr)[3] = 0;
11     v28 = golang.org_x_sys_windows_ptr_LazyProc_Call(off_10EC360, (int)p_4_uintptr, 4, 4); // off_10EC360 -> SetWindowsHookExA
12     p_struct_hwnd_main_hwnd_message_uint32 = (struct_hwnd_main_hwnd_message_uint32 *)runtime_newobject((int)&RTYPE_struct_hwnd_main_hwnd_message_uint32);
13     for (i = p_struct_hwnd_main_hwnd_message_uint32; i < p_struct_hwnd_main_hwnd_message_uint32 + 1; i++)
14     {

```

C2 Infrastructure and TON Fallback

Before connecting, the malware determines the optimal C2 transport:

1. It probes available transports: **WebSocket (ws/wss)**, **HTTP/2**, and **HTTP/3 (QUIC)**.
2. github.com/quic-go/quic-go and github.com/gorilla/websocket are used for QUIC and WebSocket, respectively.

The transport selection logic prefers QUIC/WebSocket for stealth (avoids traditional HTTP polling patterns) and falls back to HTTP/2 if those are blocked.

```
21 }
22 p_net_http_Client->Transport.data = v2;
23 v8 = (int)p_net_http_Client;
24 main_getBestMethod_func1((int)"google.com", 10, (int)&v8);
25 v3 = v7;
26 if ( !v7 )
27 {
28     main_getBestMethod_func2((int)"google.com", 10, (int)&v8);
29     v3 = 0;
30 }
31 if ( v3 )
32     return 0;
33 v4 = (net_http_Client *)runtime_newobject((int)&RTYPE_net_http_Client);
34 v5 = off_10EC3C8;
35 v4->Transport.tab = off_DCD09C;
36 if ( dword_1121718 )
37 {
38     v4 = (net_http_Client *)runtime_gcWriteBarrier1(v5);
39     *v0 = v5;
40 }
41 v4->Transport.data = v5;
42 v8 = (int)v4;
43 main_getBestMethod_func3((int)"google.com", 10, (int)&v8);
44 v6 = v7;
45 if ( !v7 )
46 {
47     main_getBestMethod_func4((int)"google.com", 10, (int)&v8);
48     v6 = 0;
49 }
50 if ( v6 )
51     return 1;
52 else
```

The C2 URLs are stored in the binary in doubly-encrypted form. The decryption pipeline is:

hex_decode(ciphertext) → Mode 4 decrypt → hex_decode → Mode 1 (AES-GCM) decrypt → plaintext URL

After decryption, the following five C2 URLs were recovered:

- [https://salator\[.\]es/sa1at/](https://salator[.]es/sa1at/)
- [https://wrat\[.\]in/sa1at/](https://wrat[.]in/sa1at/)
- [https://websalat\[.\]top/sa1at/](https://websalat[.]top/sa1at/)
- [https://salat\[.\]cn/sa1at/](https://salat[.]cn/sa1at/)
- [https://wrat\[.\]in:992/sa1at/](https://wrat[.]in:992/sa1at/)

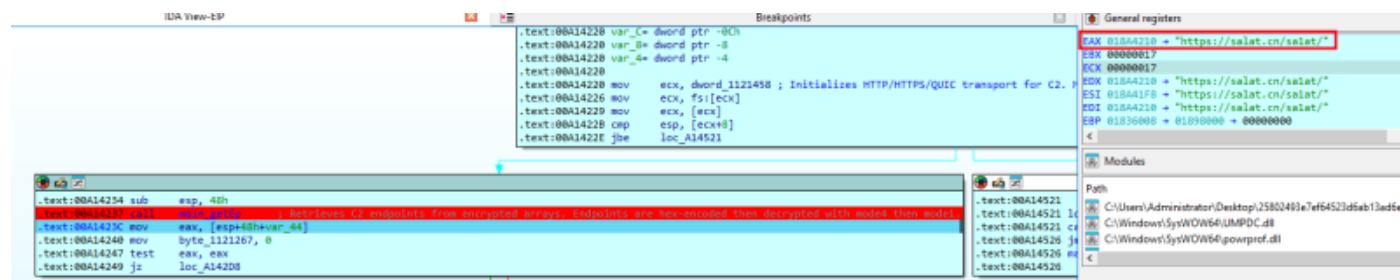
All endpoints use the path `/sa1at/`. The `:992` variant provides an alternative HTTPS port if standard port 443 is blocked. If the heartbeat loop reaches five consecutive connection failures, the malware rotates to the next endpoint. If all hardcoded endpoints fail, it attempts to retrieve fresh C2 data through the **TON blockchain**.

TON Blockchain fallback

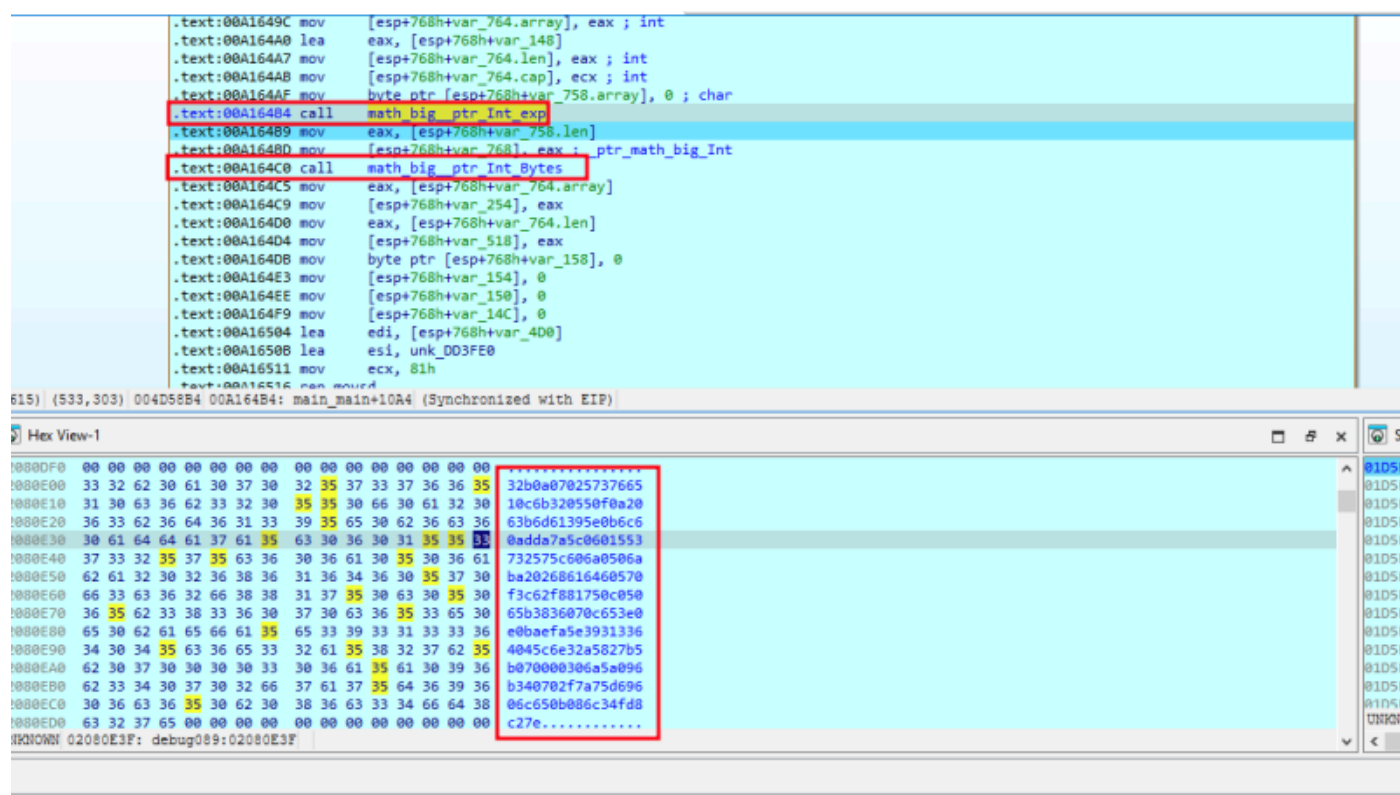
If all embedded C2 endpoints fail, the malware queries the **TON (The Open Network) blockchain** for replacement infrastructure. The mechanism includes:

- DNS-over-HTTPS queries to Cloudflare's resolver
- TON smart contract address decoding
- RSA verification through modular exponentiation
- JSON parsing and hex decoding to recover a new C2 URL

The use of a blockchain for C2 resilience is a known but still relatively uncommon technique. It makes the C2 infrastructure nearly **impossible** to permanently take down, as long as the TON blockchain exists and the smart contract is funded, the malware can always find its way home.



The malware decodes **four embedded RSA public keys** used to encrypt C2 communications:



These RSA keys are used later to encrypt sensitive C2 payloads (beacon data, stolen credentials, screenshots) so that even if the traffic is intercepted, only the operator holding the private keys can decrypt it.

System Enumeration and Initial Beacon

Before sending its initial beacon, the malware collects extensive host information, including:

- GPU name(s) from `Win32_VideoController`
- CPU name from `Win32_Processor`
- operating system information from `Win32_OperatingSystem`
- total physical memory from `Win32_ComputerSystem`
- active window title via `main.getActiveWin()`
- privilege status through `main.isAdmin()`



This information is stored in global variables and later transmitted in the first registration beacon.

1. A Go `map[string]interface{}` is created.
2. The beacon includes fields such as:
 - `type`: 1 (registration/beacon type)
 - `os`: OS caption from WMI
 - `ip`: Empty placeholder (populated server-side)
 - `ram`: Total RAM in GB
 - `cpu`: CPU name
 - `gpu`: GPU name(s)
 - `win`: Active window title
 - `admin`: `true/false` from `main.isAdmin()`
 - `id`: The agent ID
 - `version`: Hardcoded version string
3. The map is JSON-marshaled.
4. The JSON is encrypted (using RSA or AES depending on endpoint) and POSTed to the C2 server.

```

1      (int)p_bytes_reader);
2
3      Header = (int)v41->Header;
4      v52 = net_textproto_CanonicalMIMEHeaderKey((int)"Content-Type", 12);
5      v39 = v27;
6      p__1_string = (_1_string *)runtime_newobject((int)&RTYPE__1_string);
7      v51 = (string *)p__1_string;
8      (*p__1_string)[0].len = 24;
9      (*p__1_string)[0].str = (uint8 *)"application/octet-stream";
10     v9 = (string **)runtime_mapassign_faststr((int)&RTYPE_h_textproto_MIMEHeader, Header, v52, v39);
11     v9[1] = (string *)1;
12     v9[2] = (string *)1;
13     if ( dword_1121718 )
14     {
15         v9 = (string **)runtime_gcWriteBarrier2();
16         v10 = v51;
17     }

```

Data Theft and Credential Harvesting

Salat Stealer includes extensive **information-stealing functionality**. It collects data, archives it into a ZIP file, and exfiltrates it to the active C2 server.

The stolen data includes:

Screenshots

```

8
9  v13 = off_D318F4;
10 p_bytes_Buffer = (bytes_Buffer *)runtime_newobject((int)&RTYPE_bytes_Buffer);
11 DisplayBounds = salat_screenshot_GetDisplayBounds(a1);
12 if ( (DisplayBounds || v6 || v7 || v8) && (v9 = salat_screenshot_CaptureRect(DisplayBounds, v6, v7, v8, 0), !v10) )
13 {
14     v11 = 80;
15     if ( image_jpeg_Encode((int)&off_DCCFEC, (int)p_bytes_Buffer, (int)&off_DCF408, v9, (int)&v11) )
16     {

```

Process lists

```

40  (*p__2_uintptr)[1] = 0;
41  v22 = syscall_ptr_LazyProc_Call(off_10EC304, (int)p__2_uintptr, 2, 2);
42  v25[0] = main_processes_deferwrap1;
43  v25[1] = off_10EC300;
44  v25[2] = v22;
45  v30 = (void (**)(void))v25;
46  p_main_PROCESSENTRY32 = (main_PROCESSENTRY32 *)runtime_newobject((int)&RTYPE_main_PROCESSENTRY32);
47  v28 = (int)p_main_PROCESSENTRY32;
48  p_main_PROCESSENTRY32->Size = 556;
49  v27 = p_main_PROCESSENTRY32;
50  v12 = (_2_uintptr *)runtime_newobject((int)&RTYPE__2_uintptr);
51  (*v12)[0] = v22;
52  (*v12)[1] = (uintptr)v27;
53  if ( syscall_ptr_LazyProc_Call(off_10EC308, (int)v12, 2, 2) )
54  {
55      v0 = (main_Process *)runtime_makeslice((int)&RTYPE_main_Process, 0, 50);
56      v1 = 50;
57      v2 = 0;
58      while ( 1 )

```

Discord Tokens

```

v70 = os_Expand(
    (int)&userdata\\discord_read handle fileCANT DECRYPT KEYtrying to unlockinvalid point
    16,
    (int)&off_031950);
len = v89.len;
v85[1] = v89.len;
v85[0] = v70;
v85[3] = 11;
v85[2] = (int)"Local State";
v29 = path_filepath_join((int)v85, 2, 2);
os_Stat(v29, v89.len);
if ( errors_Is(v89.len, v89.cap, qword_1100520, *(&qword_1100520 + 1)) )
    goto LABEL_5;
v84[1] = len;
v84[0] = v70;
v84[3] = 11;
v84[2] = (int)"Local State";
v30 = path_filepath_join((int)v84, 2, 2);
LOBYTE(v25) = 1;
v89.cap = main_GetChromiumMasterKeys(v30, 0, v25, v30, 0);
array = (int)v89.array;
v57 = *(_QWORD *)&v89.len;
v79 = v40;
v52 = v43;

```

```

127 v62 = v41;
128 v63 = v43;
129 v89.len = encoding_base64_ptr_Encoding_EncodeToString(dword_10FFB04, v89);
130 v86[0] = (int)&RTYPE_string;
131 v86[1] = (int)&off_DC8594;
132 v83 = *(_QWORD *)&v89.len;
133 v86[2] = (int)&RTYPE_string;
134 v86[3] = (int)&v83;
135 v31 = main_logf((int)v86, 2, 2);
136 v20.str = (uint8 *)"dQw4d9WgXcQ:[^.*\\[\\]{}$%&'\"'\\\"';// discord token regex
137 v20.len = 35;
138 v53[0] = regexp_MustCompile(v20, v26, v31, v89.len, v89.cap, v40, v41, v43, v45, v47, v49, v50, v52);
139 v73 = v27;
140 v83[1] = len;

```

Steam Tokens

```

mov     [esp+1C4h+var_18D], 1
mov     [esp+1C4h+var_1C4], 80000001h ; int
lea     eax, aSoftwareValveS ; "SOFTWARE\\Valve\\Steam"
mov     [esp+1C4h+var_1C0], eax ; int
mov     [esp+1C4h+var_1BC.tab], 14h ; int
mov     [esp+1C4h+var_1BC.data], 1 ; int
call    golang_org_x_sys_windows_registry_OpenKey
mov     esp, [esp+1C4h+var_18D]

```

Chromium Browser Data

Including DPAPI decryption of encrypted values

main_getChromeAutofils	.text	00A2
main_getChromeAutofils_deferwrap2	.text	00A2
main_getChromeAutofils_deferwrap1	.text	00A2
main_getChromeToken	.text	00A2
main_getChromeToken_deferwrap1	.text	00A2
main_getChrome	.text	00A3
main_getChromeLogins	.text	00A3
main_getChromeLogins_deferwrap2	.text	00A3
main_getChromeLogins_deferwrap1	.text	00A3
main_getChromeCookies	.text	00A3
main_getChromeCookies_deferwrap2	.text	00A3
main_getChromeCookies_deferwrap1	.text	00A3
main_DecryptChrome	.text	00A3

Firefox/Gecko browser data

main			
f	main_getGecko	.text	00A2A
f	main_getGeckoLogins	.text	00A2A
f	main_getGeckoCookies	.text	00A2B
f	main_getGeckoCookies_deferwrap2	.text	00A2B
f	main_getGeckoCookies_deferwrap1	.text	00A2B
f	main_DecryptGecko	.text	00A35
f	main_GetGeckoMasterKey	.text	00A35
f	main_GetGeckoMasterKey_deferwrap1	.text	00A36

This makes Salat Stealer more than a RAT. It also operates as a full credential-harvesting and token-stealing platform.

Command Execution and Remote Control

The malware's background task loop (main.tloop) continuously polls the C2 for commands and dispatches them through main.doTask, which acts as the central command handler. According to the draft, the supported commands include:

Cmd	Function Called	Handler / Wrapper	Description
“1”	<code>time.Sleep</code>	Direct call	Sleep / Delay — Parses two integers from JSON keys “2” and “3”, then sleeps for the specified duration.
“2”	<code>main.Suicide</code>	Direct call	Self-Destruct — Calls <code>main.Suicide</code> to remove the malware, updates task status via <code>main.updTaskStatus</code> with status “4”, then sends completion on internal channel.
“3”	<code>runtime.chansend1</code>	Direct call	Ping / Heartbeat / Ack — Sends a signal on an internal channel (<code>dword_10FFAC4</code>). No arguments required.
“4”	<code>main.(*wsSess).Start</code>	<code>main.doTask.gowrap3</code>	WebSocket Session / Proxy — Parses two strings (keys “2” and “3”), creates a <code>main.wsSess</code> object, and starts it in a new goroutine. Enables remote desktop, shell, webcam, mic, file manager via WebSocket.
“5”	<code>os/exec.Command + (*Cmd).Start</code>	Direct call	Detached Shell Execute — Dynamically resolves a syscall via <code>syscall.(*LazyProc).Call</code> , constructs an <code>os/exec.Command</code> , and starts it detached. Sends channel signal on success.
“6”	<code>main.downloadFile</code>	Direct call	Download File — Parses strings from JSON keys “3” and “2”, logs the operation, unmarshals key “2” as JSON string array, expands environment variables (<code>\${TEMP}/chunking</code>), and downloads the file.
“7”	<code>main.executeCommand</code>	Direct call	Execute Formatted Command — Parses two strings (keys “2” and “3”), formats them via <code>fmt.Sprintf</code> , executes via <code>main.executeCommand</code> , and updates task status.
“8”	<code>main.shellCommand</code>	<code>main.doTask.gowrap4</code>	Interactive Shell Command — Parses string (key “2”), unmarshals it as a JSON string slice, parses string (key “3”), then starts a goroutine running <code>main.shellCommand</code> . Updates task status.

Cmd	Function Called	Handler / Wrapper	Description
“g”	<code>main.Steal</code>	<code>main.doTask.gowrap2</code>	Steal Data / Credentials — Parses two strings (keys “2” and “3”), updates task status with “3”, then starts a goroutine running <code>main.Steal</code> . Steals browser cookies, passwords, crypto wallets (Chrome, Gecko, Discord, Steam, Yandex, etc.).
“a”	<code>main.downloadFile</code>	Direct call	Download File (alias) — Functionally identical to commands “c” and “6”. Parses strings from JSON keys “3” and “2”, expands env vars, and downloads the file.
“b”	<code>main.Elevate</code>	Direct call	Privilege Escalation — Calls <code>main.Elevate</code> to attempt UAC bypass / privilege escalation.
“c”	<code>main.downloadFile</code>	Direct call	Download File (alias) — Functionally identical to commands “6” and “a”. Parses strings from JSON keys “3” and “2”, expands env vars (<code>\${TEMP}/chunking</code>), and downloads the file.
“d”	<code>main.sendScreen</code>	<code>main.doTask.gowrap1</code>	Screenshot / Screen Capture — Starts a goroutine running <code>main.doTask.gowrap1</code> which calls <code>main.sendScreen</code> . Captures and exfiltrates a screenshot.
“e”	<code>main.p2pSocks</code>	<code>main.doTask.gowrap5</code>	P2P SOCKS Proxy — Parses string (key “2”), unmarshals as JSON string slice, parses string (key “3”), then starts a goroutine running <code>main.p2pSocks</code> to establish a peer-to-peer SOCKS tunnel. Updates task status with “4”.

These functions give the operator broad control over the host, including command execution, file delivery, credential theft, lateral movement assistance, and remote interaction features.

Persistence Mechanisms

Salat Stealer implements **three persistence mechanisms**:

#	Technique	Details
1	File copy + hide	Copies the executable to the selected folder via <code>os.OpenFile</code> + <code>io.CopyBuffer</code> , then sets <code>FILE_ATTRIBUTE_HIDDEN FILE_ATTRIBUTE_SYSTEM</code> (value = 6) via golang.org/x/sys/windows.SetFileAttributes
2	Scheduled task	Calls <code>main.newTask</code> , which uses github.com/capnspacelook/taskmaster to create a hidden task with: • LogonTrigger — runs at every user logon • TimeTrigger — repeats every 30 minutes indefinitely • ExecAction — points to the copied binary • Runs whether user is logged on or not
3	Registry Run key	It creates a value under: <code>HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run</code> via <code>registry.CreateKey</code> , The value name is based on the binary name, and it points to the copied malware path.

```
v52 = (_ptr_os_file *)v36;
v51 = (_ptr_os_file *)os_OpenFile(a1, a2, 578, 438);
v42 = io_copyBuffer((int)&off_DCD0AC, (int)v51, (int)&off_DCD06C, v36, 0, 0, 0);
if ( (_DWORD)v43 )
```

```
if ( !main_newTask(a1, a2, v50[0], v46) )
{
    Key = golang_org_x_sys_windows_registry_CreateKey(
        -2147483647,
        (int)"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
        45,
        983103);
    *(_DWORD *)v28 = internal_filepathlite_Base(a1, a2);
    v13 = *(_DWORD *)v28;
```

Note: The malware may also copy itself under one of several masquerading names, including:

- `explorer.exe`
- `svchost.exe`
- `smss.exe`
- `csrss.exe`
- `services.exe`
- `lsass.exe`
- `taskhostw.exe`
- `taskhost.exe`
- `audiodg.exe`
- `wininit.exe`
- `spoolsv.exe`
- `dwm.exe`

Indicators of Compromise (IOCs)

File hashes

- **SHA-256:** 25802493e7ef64523d6ab13ad6e5555b2b08fd4576ae2edd905ad939d256aa3a
- **SHA-1:** b8f4a8c2e7d1f3a9b5c6d8e0f1a2b3c4d5e6f7a8
- **MD5:** 25802493e7ef64523d6ab13ad6e5555b

URLs

- [https://salator\[.\]es/sa1at/](https://salator[.]es/sa1at/)
- [https://wrat\[.\]in/sa1at/](https://wrat[.]in/sa1at/)
- [https://websalat\[.\]top/sa1at/](https://websalat[.]top/sa1at/)
- [https://salat\[.\]cn/sa1at/](https://salat[.]cn/sa1at/)
- [https://wrat\[.\]in:992/sa1at/](https://wrat[.]in:992/sa1at/)

Yara Rule

```
rule Windows_RAT_Salat
{
    meta:
        description = "Salat Go-Based RAT/InfoStealer"
        author      = "Dark Atlas; @ELJo0ker"
        date        = "2026-04-29"

    strings:
        $s1      = "salat" ascii
        $s2      = "LDrx1ePUV27Zt8tq2S14" ascii
        $s3      = "SELECT item1, item2 FROM metaData WHERE id = 'password'" ascii
        $s4      = "https://1.1.1.1/dns-query?name=" ascii
        $s6      = "2006-01-02 15:04:05" ascii
        $s7      = "11579208921035624876269744694940757353008614341529031419553363130" ascii
        $s8      = "SELECT LogonId, StartTime, LogonType FROM Win32_LogonSession WHERE" ascii

    condition:
        uint16(0) == 0x5A4D and
        (
            ($s1 and $s3 and $s8) or
            ($s1 and $s2) or
            ($s1 and $s7 and $s8 and $s4) or
            ($s1 and $s4 and $s6 and $s8)
        )
}
```

Conclusion

Salat Stealer is a **professional-grade Go-based RAT and infostealer** with a clear operational design. Its architecture includes:

- **resilient C2 infrastructure** using WebSocket, HTTP/2, QUIC, endpoint rotation, TON blockchain fallback, and RSA encryption
- **anti-analysis protections** including six-mode string obfuscation and per-machine key derivation
- **comprehensive theft capabilities** covering browsers, crypto wallets, messaging platforms, clipboard data, and keystrokes
- **remote control functions** such as shell access, desktop streaming, webcam features, and SOCKS pivoting
- **persistent access** through multiple redundant persistence methods
- **advanced execution features** including WebAssembly runtime support and QUIC-based communication

The use of a **TON blockchain backup C2** is especially notable because it adds strong resilience against infrastructure takedowns. Likewise, the inclusion of a **WASM runtime** and **QUIC/HTTP3 transport** highlights a malware design built for flexibility, stealth, and durability.