VENON: The First Brazilian Banker RAT in Rust

By ZenoX Team

Published: 2026-03-10 · Archived: 2026-04-05 13:05:03 UTC

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 American banking trojans, particularly the use of banking overlays and active window monitoring, characteristics present in established families such as Grandoreiro and Mekotio.

The fundamental difference emerged during static analysis: unlike all known families in the Latin American ecosystem, VENON does not contain a single line of Delphi code. The binary is compiled entirely in Rust, with 88 external dependencies identified from Crates.

This report documents the results of the technical analysis conducted by the ZenoX Research Team, covering the complete infection chain, malware capabilities, command-and-control infrastructure, and attribution indicators identified during the investigation process. The analysis required building custom tooling and reimplementing the Argon2id + XChaCha20-Poly1305 pipeline used to protect the remote configuration.

During analysis, the team raised the hypothesis that VENON may essentially be an AI-assisted Vibecoding refactor of an already-established banking trojan in the region, possibly Grandoreiro itself, rewritten from scratch in Rust. The fidelity with which classic functional patterns from the Delphi ecosystem, such as the overlay logic, window monitoring, and swap mechanisms, were reproduced in a completely different systems language suggests the author did not start from an original conception, but from a known behavioral base, using generative AI to perform the technical translation.

Additional evidence of AI use was identified in the operator’s own infrastructure: the C2 panel code exhibits AI-assisted generation patterns consistent with what the security community has termed vibe coding, reinforcing the hypothesis that the entire operation, from malware to backend, was built with extensive AI tooling assistance.

As of this publication, ZenoX has not identified any other banking trojan in the Brazilian or Latin American ecosystem with this technical profile. VENON represents, possibly, the first appearance of a Brazilian banker RAT developed entirely in Rust, with a level of sophistication comparable to tools used by known APT groups.

Click here to read the full report

Analysis Complexity

VENON presents a level of static analysis difficulty significantly greater than traditional Latin American banking trojans. While Delphi malware like Grandoreiro or Mekotio can be examined with relative ease, with readable strings, exposed RTTI, and identifiable visual components, VENON combines multiple layers of protection that make each step of the reverse engineering process a distinct technical challenge.

For an analyst familiar with Delphi trojans, where an x64dbg session reveals strings like “Banco handler of Brasil” ou “Itau” nos primeiros minutos, o VENON exige um investimento de tempo and ferramental de ordem de magnitude superior.

Barrier	Detail
UPX with modified headers	Prevents automatic decompression; requires manual header reconstruction before processing the binary
Native Rust compilation	Functions with mangled names, 88 crates, no RTTI or debug symbols
XOR encryption with 95 unique functions	Each sensitive string is decrypted by a different key derivation function; there is no reusable global key
Argon2id + XChaCha20-Poly1305	State-of-the-art encryption for the remote config; requires reimplementing the KDF parameters to decrypt
ChaCha20-Poly1305 in C2	C2 traffic encrypted per session via ring-0.17.14; passive inspection is not possible without the session key
9 active anti-analysis techniques	AMSI Bypass, ETW Bypass, ntdll overwrite, indirect syscalls, thread hiding, DACL, anti-sandbox, anti-screenshot, and Defender SID Check

Table 1 – Analysis Barriers

No single tool was sufficient to cover all protection layers. The analysis required building a six-phase pipeline, each feeding the next with the information needed to advance:

Phase	Tool / Technique	Result
Phase 1	DIE/PE Analysis + manual UPX decompression	libcef.dll unpacked (9.3 MB)
Phase 2	FLOSS v3.1.1	130,749 static strings + 41,799 Rust strings extracted; automatic deobfuscation disabled by binary density
Phase 3	Ghidra 12.0.4 Headless	17,765 functions identified; 500 functions of interest decompiled; 143,093 lines of decompiled C generated
Phase 4	Python + Capstone	95 XOR blocks processed; 92 successfully deciphered (96.8% coverage)
Phase 5	Reimplementation of Argon2id KDF + XChaCha20-Poly1305	Remote config decrypted; C2 host confirmed
Phase 6	Categorization of 143,093 decompiled lines	14 functional groups mapped, 70+ features documented, Rust modules reconstructed

Table 2 – Analysis Pipeline Adopted

The level of effort required to analyze VENON is itself a metric of the artifact's sophistication. A trojan that requires building custom analysis tools is not ordinary malware. It is an artifact that reflects advanced technical competence from its author and significantly raises the analysis cost for any incident response or threat intelligence team.

Infection Chain

VENON's infection chain is structured in eleven sequential phases, combining social engineering, multiple evasion layers, and a sophisticated payload delivery mechanism. The campaign demonstrates considerable technical planning, with each phase designed to overcome specific security controls before advancing to the next.

Initial Vector

The confirmed entry vector is DLL sideloading via the legitimate NVIDIANotification.exe installer: the malicious libcef.dll is loaded in place of the legitimate Chromium Embedded Framework by exploiting the Windows DLL search order, which prioritizes the executable's directory. The initial delivery mechanism for the NVIDIANotification.exe + libcef.dll pair to the victim's system was not, however, determined with high confidence during this analysis.

It is worth noting that during the period of sample identification, ZenoX observed a significant increase in ClickFix campaigns using NVIDIANotification.exe as the final payload, where the victim is socially engineered into executing a command that downloads and activates the file pair. The correlation between the analyzed artifact and this distribution vector is plausible and is being investigated, but it was not possible to confirm with sufficient confidence to include it as a definitive finding in this report.

The infection chain analysis documented in this report begins from the moment the NVIDIANotification.exe + libcef.dll pair is already present on the victim's system.

Distribution occurs via phishing emails, fake pages mimicking legitimate portals, or sponsored ads. In all scenarios, dropper execution depends entirely on voluntary victim action; no technical exploit is required at this stage.

Install via PowerShell

Obfuscated batch file of ~1.6 KB. Critical strings (URLs, paths, commands) are reconstructed at runtime by concatenating fragmented variables, avoiding static signature detection.

The script relaunches itself with **RunAs** via PowerShell if not running as administrator.

```
@echo off
setlocal EnableDelayedExpansion

net session >nul 2>&1
if %errorLevel% neq 0 (
    powershell -NoP -C "Start-Process -FilePath '%~f0' -Verb RunAs"
    exit /b
)
```

Figure 1 – Privilege Escalation

Adds **C:\ProgramData\USOShared\ NuPLihaOH** via **Add-MpPreference** before the download. The parent directory mimics the Update Session Orchestrator; the space in the subfolder name impedes command-line searches.

```
set "DyzaGp=C:\P"
set "zlVJB=rogram"
set "VCVD=Data"
set "Anfgxy=\US"
set "IbnLk=OSha"
set "sTqg=red"
set "llPybi=!DyzaGp!!zlVJB!!VCVD!!Anfgxy!!IbnLk!!sTqg!\ NuPLihaOH"
if not exist "!llPybi!" mkdir "!llPybi!" >nul 2>&1
```

Figure 2 – Defender exclusion

ZIP obtained from S3 bucket via **Invoke-WebRequest**, with URL dynamically constructed by variable fragmentation.

Figure 3 – Payload download

The ZIP contains **NVIDIANotification.exe** (signed NVIDIA binary) and **libcef.dll** (malware). The executable is renamed to **®mjtgr.exe**; the character ® (U+00AE) in the prefix impedes references via CLI and forensic tools.

Figure 4 – Extraction and Renaming

Run key added at **HKCU\…\Run**; individual exclusion created for **®mjtgr.exe**.

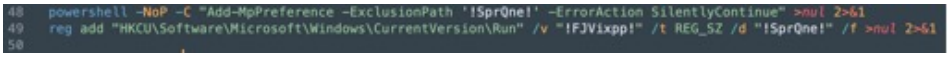


Figure 5 – Persistence + second exclusion

Script self-deletes via **(goto) 2>nul &** and forces reboot in 3 seconds (**shutdown /r /t 3 /f**), activating the run key and eliminating evidence of the entry vector.

Figure 6 – Self-deletion and reboot

DLL Sideloading

After reboot, Windows executes **®mjtgr.exe** via run key. Since the Windows DLL search order prioritizes the executable's directory, the malicious **libcef.dll** is loaded in place of the legitimate Chromium Embedded Framework.

The process appears in Task Manager with the NVIDIA name and digital signature. The DLL exports the standard functions of a COM object (**DllCanUnloadNow, DllGetClassObject, DllMain, DllRegisterServer, DllUnregisterServer**) to mimic a legitimate DLL; all malicious code resides in the **DLL_PROCESS_ATTACH** do **DllMain**.

Initialization and Evasion

Upon loading, the DLL executes nine evasion techniques in sequence before initiating any malicious activity.

The most sophisticated technique in this phase is the overwrite of the **.text** da **ntdll.dll** in memory with the clean version read from disk.

```
// XOR-decrypt "ntdll.dll"
lVar40 = FUN_1800b8a3c(0x1804e25e5, 0x5cd8cb9c);
// ...
pHVar35 = GetModuleHandleA((LPCSTR)&uStack_16a8);  // Get ntdll in memory
if (pHVar35 != (HMODULE)0x0) {
    uStack_890 = "src\\stealth\\indirect_syscall.rs";  // ← Panic path!

    // Open ntdll from DISK (fresh, unhook copy)
    pvVar34 = CreateFileW((LPCWSTR)pHStack_ff0, 0x80000000, 1, ...);
    DVar28 = GetFileSize(pvVar34, (LPDWORD)0x0);
    BVar30 = ReadFile(pvVar34, uStack_890, DVar28, ...);
    CloseHandle(pvVar34);

    // Verify PE signature
    if (*(int *)(pcVar23 + uVar38) == 0x4550) {  // "PE\0\0"
        // Find .text section
        for (lVar40 = 0; ...; lVar40 = lVar40 + 0x28) {
            if (pcVar23[... + 0x18] == '.' &&
                pcVar23[... + 0x19] == 't' &&
                pcVar23[... + 0x1a] == 'e' &&
                pcVar23[... + 0x1b] == 'x' &&
                pcVar23[... + 0x1c] == 't') {   // ← Procura ".text"

                // VirtualProtect → PAGE_EXECUTE_READWRITE (0x40)
                iVar61 = (*pcVar58)(lpAddress, uVar38, 0x40, &DStack_1d38);
                if (iVar61 != 0) {
                    // OVERWRITE memory ntdll .text with disk copy
                    FUN_1804e2550(lpAddress, pcVar23 + offset, uVar38);
                    // Restore original protection
                    VirtualProtect(lpAddress, uVar38, DStack_1d38, &DStack_1d38);
                }
            }
        }
    }
}
```

Figure 7 – ntdll .text Overwrite

For sensitive operations, the malware implements indirect syscalls: syscall numbers are read directly from ntdll on disk and stubs are constructed in memory, bypassing any hook that could be reinstalled. Finally, it applies thread hiding via **NtSetInformationThread** with the flag **ThreadHideFromDebugger**, modifies the process's own DACL to deny external access, and configures **SetWindowDisplayAffinity** nos overlays para que screenshots exibam apenas tela preta.

```
// Thread Hiding
LOCK();
DAT_180915ef8 = '\x01';      // Set "ready" flag
UNLOCK();
if (cVar27 == '\0') {
    pvVar34 = GetCurrentThread();
    NtSetInformationThread(pvVar34, 0x11, 0, 0);  // ThreadHideFromDebugger
    pHStack_80 = GetCurrentProcess();

    // Initialize security descriptor for DACL
    BVar30 = InitializeSecurityDescriptor(pHStack_70, 1);
    if (BVar30 != 0) {
        // XOR-decrypt SID strings...
        // ...
        BVar30 = InitializeAcl((PACL)uStack_890, 0x400, 2);
        if (BVar30 != 0) {
            AddAccessDeniedAce(pAcl, 2, 0x1fffff, pSid);       // DENY all
            AddAccessAllowedAce(pAcl, 2, 0x101000, ppuStack_98); // Allow minimal
            AddAccessAllowedAce(pAcl, 2, 0x101000, plVar39);
            AddAccessAllowedAce(pAcl, 2, 0x1fffff, puStack_b0);
            BVar30 = SetSecurityDescriptorDacl(pHStack_70, 1, pAcl, 0);
            if (BVar30 != 0) {
                SetKernelObjectSecurity(pHStack_80, 4, pHStack_70); // Apply!
            }
        }
    }
}
```

Figure 8 – ThreadHideFromDebugger + DACL Protection

Remote Config Fetch

The config thread makes a GET request to **hxxps://storage.googleapis[.]com/mydns2026/startabril2026**, with fallback to **hxxps://pluginsafeguard[.]help/ipv4/config.enc**. Google Cloud Storage is used as the primary channel because it is rarely blocked by corporate firewalls.

The response goes through three decryption layers: Base64 Decode, followed by key derivation via Argon2id (**m=19456 KiB, t=2, p=1**) with password **L0@D_S3CR3T_K3Y_X9F2_PR0D_2024!** e salt **LOAD_SALT_2024!!**, and finally XChaCha20-Poly1305 decryption with a 24-byte nonce extracted from the beginning of the blob. The result is a JSON containing the C2 address: **{“host”:”brasilmotorsvs14[.]com”}**.

Earlier versions of this RAT used AES-256-CBC with SHA-256 and a zero IV, significantly weaker. The migration to Argon2id and XChaCha20-Poly1305 reflects deliberate technical evolution between versions.

```
// XOR-decrypt secret key: "L0@D_S3CR3T_K3Y_X9F2_PR0D_2024!" (31 bytes)
lVar14 = FUN_1800b8f40(0x1804e2a13, 0xf22d3a3c);   // Unique key derivation fn
for (uVar48 = 0; uVar48 < 0x18; uVar48 = uVar48 + 8) {
    uVar15 = FUN_180050a23(lVar14 + uVar48);
    *(ulonglong *)((longlong)&local_1180 + uVar48) =
         uVar15 ^ *(ulonglong *)((longlong)&DAT_180750b16 + uVar48);
}
//                                    ^^^^^^^^^^^^^^ ,rdata XOR data
for (uVar48 = 0x18; uVar48 < 0x1c; uVar48 = uVar48 + 4) {
    uVar11 = FUN_180050a00(lVar14 + uVar48);
    *(uint *)((longlong)&local_1180 + uVar48) =
         uVar11 ^ *(uint *)((longlong)&DAT_180750b16 + uVar48);
}
uStack_1164 = *(ushort *)(lVar14 + 0x1c) ^ 0xe0ad;   // Last 2 bytes
bStack_1162 = *(byte *)(lVar14 + 0x1e) ^ 100;         // Final byte (0x64 = '!')
FUN_1800bcaec(&local_13a0, &local_1180, 0x1f);        // Create String, len=0x1f(31)

// XOR-decrypt salt: "LOAD_SALT_2024!!" (16 bytes)
lVar14 = FUN_1800b889f(&DAT_1804eed92, 0x6af458f);    // Different key derivation fn
for (uVar48 = 0; uVar48 < 0x10; uVar48 = uVar48 + 8) {
    uVar15 = FUN_180050a23(lVar14 + uVar48);
    *(ulonglong *)((longlong)&local_940 + uVar48) =
         uVar15 ^ *(ulonglong *)((longlong)&DAT_180750cb5 + uVar48);
}
FUN_1800bcaec(&local_1180, &local_940, 0x10);         // Create String, len=0x10(16)
```

Figure 9 – XOR Decrypt Secret Key + Salt

Persistence and C2

After obtaining the configuration, the malware installs a Scheduled Task named **“NVIDIA Notification Service”** with trigger **AtLogOn** and maximum run level, replacing the WMI Event Subscription mechanism used in earlier versions. The WebSocket connection to the C2 is established under TLS 1.3 with ChaCha20-Poly1305 cipher via **tungstenite** e **rustls**. Each victim is identified by a HardwareID calculated as the SHA-256 of the computer name concatenated with the volume serial.

Itaú Swap

In addition to the attack mechanisms via banking overlay and clipboard manipulation, VENON deploys two VBScript code blocks extracted directly from **libcef.dll**. These blocks implement a shortcut hijacking mechanism targeting the Itaú Application, replacing legitimate system shortcuts with tampered versions that redirect the victim to a web page under operator control, preserving the bank’s original icon to avoid suspicion.

This is a VB module exclusive to Itaú; no custom scripts like this were found for other banks and targets.

The attack is operated in two distinct stages: install, which performs shortcut substitution, and uninstall, which reverts the modifications. The presence of the second block indicates the mechanism is controllable via C2, allowing the operator to restore shortcuts before ending the session or upon detecting signs of investigation.

Block 1: Install

The install script is responsible for locating and tampering with all Itaú Application shortcuts present on the victim's machine. Execution follows four main steps:

- **Microsoft Edge Path Resolution**: the script queries the registry at **HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\msedge.exe.** If the key is absent, it tries the WOW6432Node alternative path, then directly checks the standard installation paths at **Program Files (x8**6) e **Program Files**. If Edge is not found by any method, the script writes **-2** to the result file and exits without making modifications.
- **Target Location Enumeration**: o script define um array com oito locais handler of sistema de arquivos onde atalhos handler of Itau podem existir.
- **Itaú Shortcut Identification**: for each **.lnk** encontrado nos locais alvo, o script abre o atalho via **WScript.Shell.CreateShortcut** e verifica se o **TargetPath** contains any of the strings **itauaplicativo.exe, aplicativo itau** ou **itauaplicativo**. The comparison is done in lowercase to ensure case-insensitivity.

```
\\On Error Resume Next
Set objShell = CreateObject("WScript.Shell")
Set objFSO = CreateObject("Scripting.FileSystemObject")

strUserProfile = objShell.ExpandEnvironmentStrings("%USERPROFILE%")
strPublicDesktop = objShell.ExpandEnvironmentStrings("%PUBLIC%") & "\Desktop"
strCaminhoOriginal = strUserProfile & "\AppData\Local\Aplicativo Itau\itauaplicativo.exe"
strCaminhoOriginal2 = strUserProfile & "\AppData\Local\ItauAplicativo\itauaplicativo.exe"
strURL = "https://www.itau.com.br/empresas"
strResultFile = ""
```

Figure 10 – Itaú Shortcut Identification

- **Shortcut Substitution and Icon Preservation**: shortcuts identified as Itaú have their **TargetPath** substituído pelo caminho handler of **msedge.exe** resolvido anteriormente, and os **Arguments** definidos como **hxxps://www.itau.com[.]br/empresas**. The **WorkingDirectory** e limpo. Criticamente, o **IconLocation** is preserved if present, making the shortcut visually identical to the original. The number of modified shortcuts is written to the result file.

Variable / Folder	**Expanded Path (example)**
Desktop (user)	**%USERPROFILE%\Desktop**
Desktop (public)	**%PUBLIC%\Desktop**
StartMenu\Programs	**%APPDATA%\Microsoft\Windows\Start Menu\Programs**
StartMenu\Programs\Itau	**…\Programs\Itau**
StartMenu\Programs\Aplicativo Itau	**…\Programs\Aplicativo Itau**

AllUsersPrograms	**%ALLUSERSPROFILE%\Microsoft\Windows\Start Menu\Programs**
AllUsersPrograms\Itau	**…\Programs\Itau**
AllUsersPrograms\Aplicativo Itau	**…\Programs\Aplicativo Itau**

Table 1 – Locations enumerated by the VBS script

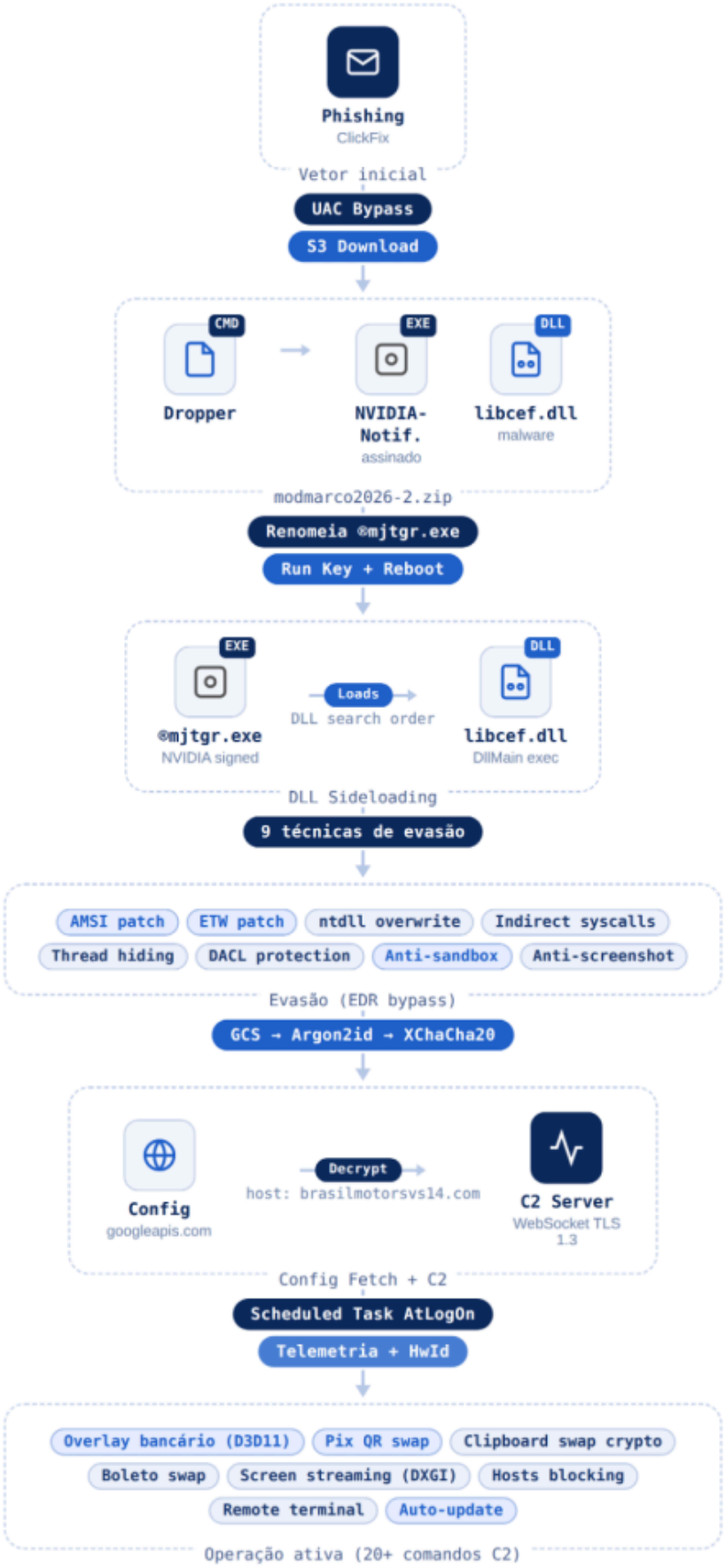
Block 2: Uninstall / Restore

The second block implements the inverse operation: it identifies previously tampered shortcuts and restores them to the original Itaú Application executable. The identification logic is distinct from Block 1 and is based on artifacts left by the modification, not the original shortcut attributes.

The script resolves the path to the legitimate Itaú executable by checking two possible installation locations: **%USERPROFILE%\AppData\Local\Aplicativo Itau\itauaplicativo.exe** e **%USERPROFILE%\AppData\Local\ItauAplicativo\itauaplicativo.exe**. If neither is found, it uses the first path as a fallback. This suggests the operator can trigger the uninstall even on machines where the application is not installed, possibly to cover tracks before the victim notices the tampering.

The **IsModifiedItauShortcut** function identifies tampered shortcuts by checking whether the **TargetPath** aponta para **msedge.exe, microsoft-edge, chrome.exe** ou **firefox.exe**, in combination with a **Arguments** contendo a string **itau.com.br**. The inclusion of Chrome and Firefox as detection criteria indicates the operator may have attack variants using other browsers, or that the criterion was designed to be robust against future variations of Block 1.

Identified shortcuts have their **TargetPath** restored to the Itaú executable, the **Arguments** limpos, and o **IconLocation** redefined to **itauaplicativo.exe,0** if the file exists on disk. The count of restored shortcuts is written to the result file, suggesting the C2 monitors the operation’s success.


Phishing
ClickFix
Vetor inicial
UAC Bypass
S3 Download
CMD
Dropper
EXE
NVIDIA-Notif.
assinado
DLL
libcef.dll
malware
modmarco2026-2.zip
Renomeia ®mjtgr.exe
Run Key + Reboot
EXE
®mjtgr.exe
NVIDIA signed
Loads
DLL search order
DLL
libcef.dll
DllMain exec
DLL Sideloading
9 técnicas de evasão
AMSI patch
ETW patch
ntdll overwrite
Indirect syscalls
Thread hiding
DACL protection
Anti-sandbox
Anti-screenshot
Evasão (EDR bypass)
GCS → Argon2id → XChaCha20
Config
googleapis.com
Decrypt
host: brasilmotorsvs14.com
C2 Server
WebSocket TLS 1.3
Config Fetch + C2
Scheduled Task AtLogOn
Telemetria + HwId
Overlay bancário (D3D11)
Pix QR swap
Clipboard swap crypto
Boleto swap
Screen streaming (DXGI)
Hosts blocking
Remote terminal
Auto-update
Operação ativa (20+ comandos C2)

Monitored Targets

VENON monitors 33 financial institutions and digital asset platforms, distributed across six categories. Monitoring is performed via window title and active browser domain checks, activating attack mechanisms upon detecting any of the targets below.

#	Institution	Monitored Domain
1	Itaú Unibanco	itau.com.br
3	Santander Brasil	santander.com.br
4	Caixa Econômica Federal	caixa.gov.br
5	Banco handler of Brasil	bb.com.br
6	Nubank	nubank.com.br
7	Banco Inter	bancointer.com.br
9	Sicoob	(string parcialmente decodificada; confirmado via titulo “- sicoob”)
10	Sicredi	sicredi.com.br
11	Banco Original	original.com.br
12	Banco Safra	safra.com.br

Table 1 – Traditional Banks

#	Institution	Monitored Domain
13	BTG Pactual	btgpactual.com

Table 2 – Investment Bank

#	Institution	Domain / Identifier
14	PagBank / PagSeguro	pagseguro.uol.com.br
15	PicPay	picpay.com
16	Mercado Pago	mercadopago.com.br
17	Bling ERP	(título de janela: “bling erp”)

Table 3 – Fintech / Payments

#	Institution	Monitored Domain
18	Receita Federal	receita.fazenda.gov.br, gov.br/receitafederal

Table 4 – Government

#	Platform	Monitored Domain
19	Binance	binance.com
20	Coinbase	coinbase.com
21	Kraken	kraken.com
22	Bybit	bybit.com
23	Mercado Bitcoin	mercadobitcoin.com
24	Foxbit	foxbit.com
25	Gemini	gemini.com
26	Nexo	nexo.com
27	Ripio	ripio.com

Table 5 – Exchanges and Crypto

#	Platform	Identificador
28	MetaMask	(título de janela: “metamask”)
29	Trust Wallet	(título de janela: “trust wallet”)
30	Phantom	phantom.app
31	Ledger Live	ledger.com
32	Rabby Wallet	rabby.io
33	Cake DeFi	app.cakedefi

Table 6 – Crypto Wallets

During the initial analysis phases, the ZenoX team identified numerous behavioral similarities with established families in the Latin American ecosystem, particularly Grandoreiro: use of banking overlays for visual interception, active window monitoring, registry-based persistence mechanisms, and a command-and-control structure with remote operation capability. At first glance, the artifact appeared to be yet another representative of the classic LATAM banking trojan paradigm.

The fundamental difference emerged during static analysis: unlike all known families in the region, VENON does not contain a single line of Delphi code. The entire binary is compiled from Rust, with 88 crates identified in Cargo.lock and 17,765 functions. This is not merely a Rust loader delivering a Delphi payload, as experimentally observed in Casbaneiro. The final payload, with all attack logic, encryption, evasion, and C2 communication, is native Rust end-to-end.

As of this publication, ZenoX has not identified any other banking trojan in the Latin American or Brazilian ecosystem with this profile. VENON represents, possibly, the first discovery of a Brazilian banker RAT developed entirely in Rust, with a level of technical sophistication comparable to APT group tools.

The Latin American Banking Trojan Ecosystem

Latin America, and Brazil in particular, is the global epicenter of banking trojans. Of the 30 most detected families worldwide, 11 are of Brazilian origin, representing 22% of all detections in 2024. Brazil alone accounts for 61% of banking trojan detections in the region (ESET, 2024).

The ecosystem has historically been dominated by Delphi-written families, a language that offers self-sufficient binaries and ease of GUI development for banking overlays. The table below lists the main active families and their language characteristics:

Family	Language	Profile
Grandoreiro	Delphi	The largest LATAM banker: 1,700 banks, 45 countries, partial MaaS model
Mekotio	Delphi	Europa and LATAM, uso extensivo de PowerShell no dropper
Casbaneiro	Delphi + Rust*	Experimental Rust use only in the downloader; final payload in Delphi
Guildma	Delphi	Also known as Astaroth; XOR string obfuscation
Mispadu	Delphi	SAMBA SPIDER; dropper via HTA/VBScript
Coyote	.NET + Nim	Emerged in 2024; Squirrel installer; 61 Brazilian banks
Kiron	Rust	Rust downloader with DGA and browser credential theft (2024); no banking attack logic in Rust
VENON	Rust nativo	Full banking RAT in Rust: 88 crates, active evasion, state-of-the-art encryption

Table 1 – Profile Comparison Among Latin American Malware

* Casbaneiro used Rust experimentally only in the download component in 2023; the core malware handler remains in Delphi.

The following table compares VENON's technical attributes with the three most representative families in the ecosystem: Grandoreiro (volume and reach), Coyote (modern language, Brazil), and Mekotio (European presence).

Characteristic	VENON	Grandoreiro	Coyote	Mekotio
Language	**Rust nativo**	Delphi	.NET + Nim	Delphi
Active Period	2024-2026	2017-2026	2024-2026	2015-2026
Binary Size	9,3 MB (UPX)	390-414 MB	~50 MB	~20-30 MB
Targets (Banks)	36+	1.700+	61+	50+
Targets (Crypto)	21 plataformas	276 wallets	Não	Não
Geographic Reach	Brasil	45 países	Brasil	LATAM + Europa
C2 Protocol	**WebSocket TLS**	RealThinClient	SSL mutual	TCP custom

C2 Encryption	**ChaCha20-Poly1305**	AES-CTS	AES	XOR + custom
Config Encrypt	**Argon2 + XChaCha20**	AES-256	AES	XOR
AMSI Bypass	Sim	Não	Não	Não
ETW Bypass	Sim	Não	Não	Não
ntdll Unhook	Sim	Não	Não	Não
Indirect Syscalls	Sim	Não	Não	Não
Pix QR Intercept	Sim	Não	Não	Não
Boleto Swap	Sim	Não	Não	Parcial
Screen Streaming	**DXGI (GPU)**	Screenshots	Screenshots	Screenshots
Operational Model	Solo / artisanal	MaaS (partial)	Solo	Solo

Table 2 – Comparative Analysis: VENON vs. Established Families

Attribution

Attribution of VENON to a known operator or group presented low confidence throughout the analysis. Although the malware shares several behavioral characteristics with established Latin American families such as Grandoreiro, Mekotio, and Coyote, the structural technical differences are sufficiently deep to prevent high-confidence attribution to any previously documented group or campaign in the region.

Hypothesis: AI-Assisted Development

An element that complicated both the analysis and attribution is the hypothesis that VENON may have been developed with extensive artificial intelligence assistance, which the security community has termed “vibe coding”. The Rust code structure presents patterns suggesting a developer familiar with the capabilities of existing Latin American banking trojans, but who used generative AI to rewrite and expand these functionalities in Rust, a language that requires significant technical experience to use at the observed level of sophistication.

This hypothesis would explain some asymmetries observed in the code: the coexistence of state-of-the-art cryptographic implementations alongside relatively straightforward control structures, and the reproduction in Rust of functional patterns common in Delphi, such as swap logic and window enumeration, with greater technical fidelity than would be expected from a first-time Rust developer. If confirmed, this would be one of the first documented instances of AI use for banking trojan development in Latin America.

Developer Exposure via Compilation Artifacts

During the string extraction process from an early DLL version (January 2026), identified during hunting, the ZenoX team located local compilation paths exposed in the binary. Unlike the version analyzed as the main subject of this report, this earlier sample had not removed the full paths from the author’s development environment.

The exposed paths repeatedly contain the username **byst4**, revealing the local machine username where the malware was compiled. The sequence of strings present in the binary includes paths such as **C:\Users\byst4\.cargo\registry\src\...**, a

pattern consistent across dozens of entries corresponding to the Rust crates used in the project.

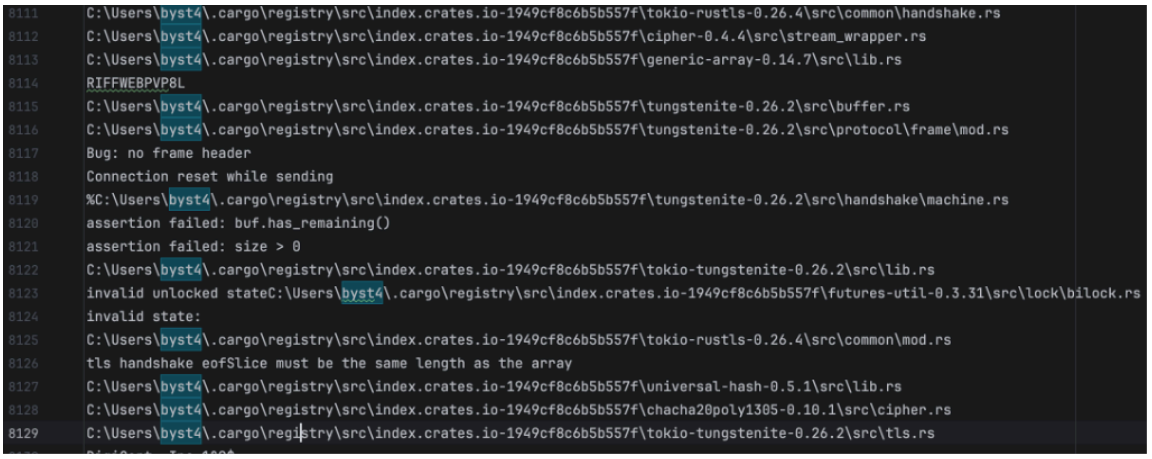


Figure 11 – Username Mention in Rust Crates Exposing the Developer

Indicatores de comprometimento

Type	**Indicator**	**Description / Contexto**
Domains		
Domain	brasilmotorsvs14[.]com	Primary C2 WebSocket (Cloudflare)
Domain	lazybearpottery[.]net	Alternate C2 (Cloudflare)
Domain	digitalmoineyp[.]com	Distribution infrastructure
Domain	portalhondihs[.]com	Distribution infrastructure
Domain	storage.googleapis[.]com	Dead drop – GCS bucket mydns2026
URLs		
URL	https://s3.sa-east-1.amazonaws[.]com/8151218-25.2025.7.12.5178/modmarco2026-2.zip	Payload – AWS S3 (sa-east-1)
URL	https://storage.googleapis[.]com/mydns2026/startabril2026	Dead drop resolver – GCS
URL	https://storage.googleapis[.]com/mydns2026/startmarco2026_1_	Dead drop resolver – GCS
URL	https://storage.googleapis[.]com/mydns2026/startjaneiro_1_	Dead drop resolver – GCS

Type	Indicator	Description / Contexto
URL	https://storage.googleapis[.]com/mydns2026/startabril_2	Dead drop resolver – GCS
URL	https://pastebin.com/raw/2qEMcLsD	Dead drop resolver – Pastebin
URL	https://digitalmoineyp[.]com/v2/cloudflare/avsmail/recive.php	Distribution endpoint
IP Addresses – C2 / Panel		
IP	104.21.7[.]106	brasilmotorsvs14.com – Cloudflare CDN
IP	188.114.96[.]3	lazybearpottery.net – Cloudflare CDN
IP	206.0.29[.]58	VENON Panel – LACNIC
IP	51.222.75[.]250	VENON Panel – OVH Canada
IP	51.222.75[.]248	VENON Panel – OVH Canada
IP	192.99.226[.]117	VENON Panel – OVH Canada
IP	212.69.5[.]84	VENON Panel – Europe
IP	34.227.229[.]85	VENON Panel – AWS EC2
IP Addresses – Abused Legitimate Services		
IP	34.117.59[.]81	ipinfo.io – geolocation fingerprinting
IP	142.251.140[.]187	storage.googleapis.com – dead drop GCS
IP	142.251.141[.]67	c.pki.goog – CRL validation
IP	142.251.140[.]163	o.pki.goog – OCSP validation
File System Paths		

Type	Indicator	Description / Contexto
Path	C:\ProgramData\USOShared\NuPLihaOH\	Implant staging directory
Path	C:\ProgramData\USOShared\NuPLihaOH\NVIDIANotification.exe	Legitimate NVIDIA executable (sideloading)
Path	C:\ProgramData\USOShared\NuPLihaOH\®mjtgr.exe	Main implant (Unicode prefix)
Path	C:\ProgramData\USOShared\NuPLihaOH\qYogBt.zip	Temporary ZIP in staging
Registry Keys		
Registry	HKCU\Software\Microsoft\Windows\CurrentVersion\Run	Persistence – executes ®mjtgr.exe at logon
Processes		
Process	®mjtgr.exe	Main implant (PID 6864)
Process	NVIDIANotification.exe	Legitimate NVIDIA executable – sideloading vehicle
Process	CasPol.exe	LOLBin – sideloading target
Process	wscript.exe	VBS script executor
Files		
File	libcef.dll	Malicious DLL – sideloading via CEF (packed)
File	NVIDIANotification.exe	Legitimate NVIDIA executable used as loader
File	®mjtgr.exe	Renamed implant (Unicode ® prefix)
File	qYogBt.zip	Temporary ZIP in staging

Type	Indicator	Description / Contexto
File	modmarco2026.zip	Versioned payload – hosted on S3
File	DocumentReclamaAQUI_56b2ca9811.cmd.bin	CMD dropper (phase F1)
File	Itau_swap_install.vbs	VBS Script – Itaú shortcut swap
File	startabril2026	Dead drop resolver – GCS
File	startmarco2026_1_	Dead drop resolver – GCS
File	startjaneiro_1_	Dead drop resolver – GCS
File	startabril_2	Dead drop resolver – GCS
Hashes		
MD5	427ccfa456ed27a819aa152708212ff4	libcef.dll (packed)
SHA256	c482286a7fdfb64d308c197a4deabcd773b8b62d9e74d1d08fcfd02568d75d72	libcef.dll (packed)
MD5	2d1c4778094ba0e1a6e13bb67ce1b631	libcef.dll (unpacked)
SHA256	75d1a2560cf93c6a028aa3573febddaf713014d64b0e8904488111772e4cff49	libcef.dll (unpacked)
MD5	a99cb35768489b7aacf2d31d33d8f541	Itau_swap_install.vbs
SHA256	fd5d9effc1ef77a49b0720d2691bc144f513609760c22fa62bc1e8b84dedf879	Itau_swap_install.vbs
SHA256	78b62856878cb09602b14104df18ca2bedac8640e09d74b934ff3ea0e15627f3	Amostra adicional
SHA256	d61be2b21e135726c547a388ecb47552559e5221894f5005ce35bdb24efc0c26	Amostra adicional

---