

LATAM Businesses Hit by XWorm via Fake Financial Receipts: Full Campaign Analysis

By Moises Cerqueira (0xOlympus)

Published: 2026-02-17 · Archived: 2026-05-22 02:00:49 UTC

Editor's note: The current article is authored by Moises Cerqueira, malware researcher and threat hunter. You can find [Moises on LinkedIn](#).

Malware campaigns targeting Latin America (LATAM) are evolving. While the final payloads, often commodity RATs like [XWorm](#), remain consistent, delivery mechanisms are becoming increasingly sophisticated to bypass region-specific defenses and increase the chance of reaching real business users.

In this analysis, we dissect a recent campaign targeting Brazilian users. What starts as a **deceptive “banking receipt”** quickly turns into a multi-stage infection chain that leverages steganography, Cloudinary abuse, and a dedicated .NET persistence module designed to bypass traditional schtasks monitoring, reducing early visibility for security teams and prolonging dwell time.



Complete infection chain from WScript execution to CasPol injection

Key Takeaways

Built to blend into finance workflows: A “receipt” lure is optimized for real corporate inboxes and shared drives across LATAM.

High click potential in real operations: Payment and receipt themes map to everyday processes, which raises the chance of execution on work machines.

The chain is designed to stay quiet: WMI execution, fileless loading, and .NET-based persistence reduce early detection signals and increase dwell time.

One endpoint can become an identity problem: XWorm access can lead to credential/session theft and downstream compromise of email, SaaS, and finance systems.

Trusted services and binaries are part of the evasion: Cloud-hosted payload delivery and CasPol.exe abuse help the activity blend in.

Early detection is an operational advantage: Better monitoring + faster triage + [regional hunting](#) can keep his attack from escalating into fraud, data exposure, or ransomware.

Stage 1: The Deceptive Delivery

This campaign begins with a classic but effective technique aimed at Brazilian users: a malicious file masquerading as a bank receipt (“Comprovante-Bradesco...”). While it abuses the double-extension trick (.pdf.js) to look like a document, it is, in reality, a Windows Script Host (WSH) dropper designed for direct execution



The file tries to masquerade as a PDF document

The file tries to masquerade as a PDF document with a fake extension to deceive the user.

Although the file size is unusually large (~1.2MB) for a simple script, this is intentional. The attackers padded it with junk data to inflate entropy and evade static analysis scanners that may skip larger files, helping the lure pass

through initial controls and delaying detection.

Analyzing the Obfuscated JavaScript

Upon opening the file, there's no readable code. Instead, the script uses heavy obfuscation via Unicode "junk injection." The malicious logic is buried inside massive string variables packed with emojis, homoglyphs, and other non-ASCII characters

 Heavily obfuscated code using Unicode characters and emojis
Heavily obfuscated code using Unicode characters and emojis.

As seen above, the script uses a delimiter-based reconstruction method. Rather than relying on complex cryptography, it applies a simple `.replace()` function at runtime to strip away the injected Unicode noise (the delimiters) and reconstruct the payload

Deobfuscation and Payload Extraction

To understand the dropper's intent, we replicated the deobfuscation logic using CyberChef. By stripping the specific Unicode delimiters and decoding the 16LE text, we revealed the core payload.

 Using CyberChef to strip Unicode delimiters
Using CyberChef to strip Unicode delimiters and reveal the PowerShell command

The deobfuscated payload confirms that this is a pure dropper. It constructs a PowerShell command responsible for downloading the next stage.

Evasion via WMI Execution

An interesting aspect of this sample is how it executes the payload. Instead of using the noisier `WScript.Shell.Run`, it leverages WMI (Windows Management Instrumentation) via `GetObject("winmgmts:root\cimv2")` and `Win32_Process`.

 ANY.RUN process graph
The execution flow in ANY.RUN confirms the use of WMI to spawn PowerShell

This technique allows the attacker to set `ShowWindow = 0`, spawning the PowerShell process in a hidden window to avoid alerting the user. The script also implements a hardcoded `Sleep(5000)` delay, likely to ensure the system is ready and to bypass simplistic sandbox heuristics that expect immediate malicious behavior.

Stage 2: PowerShell, Steganography, and Argument Decoding

Upon decoding the PowerShell command launched by the JavaScript dropper, we find a script designed to act as a stealthy bridge. It performs three critical tasks: downloading a disguised resource, extracting a fileless loader (Stage 3), and preparing the configuration for the final infection.

Abusing Cloudinary for Evasion

The script initializes a `System.Net.WebClient` and sets a specific User-Agent to mimic a legitimate browser. It then reaches out to a hardcoded URL hosted on Cloudinary, a popular image hosting service.

 malware abuses Cloudinary
The malware abuses legitimate infrastructure (Cloudinary) to bypass domain reputation filters

The URL is constructed at runtime using a simple replace function (`.Replace('#', 'h')`) to evade static string detection. To the network perimeter, this traffic looks like a user downloading a standard JPEG image.

Steganography and In-Memory Loading

The downloaded file (`optimized_MSI_lpsd9p.jpg`) carries a hidden payload. The PowerShell script does not save this file to disk as an image. Instead, it reads the data stream and searches for specific markers: `BaseStart-` and `BaseEnd-`.

 The Stage 3 loader is embedded within the image file boundaries
The Stage 3 loader is embedded within the image file boundaries

The data between these markers is a Base64-encoded .NET assembly (Stage 3). The script extracts this blob and loads it directly into memory using `[Reflection.Assembly]::Load()`. This "fileless" technique ensures that the Stage 3 loader never hits the disk, evading traditional antivirus scans.

Deciphering the Configuration Arguments

Before invoking the loaded assembly, the PowerShell script prepares a massive argument string (``$argsBase64``). This is where the malware's true intent is revealed.

Deobfuscating this string (Base64 → UTF-16LE) yields a comma-separated list of parameters that control the behavior of the next stages. Most notably, the first argument appears to be a random string:

'0hHduAjMxQjNwYTMxAjNyAjMf9mdpVXcyF2LyJmLt92YuM3byZXasJXZsV3b29yL6MHc0RHa'

 Reversing and decoding the argument reveals the final payload URL
Reversing and decoding the argument reveals the final payload URL

Upon closer inspection, this string is actually Reversed Base64. By reversing the string order and decoding it, we uncover the URL for the final XWorm payload (Stage 4): https://voulerlivros.com.br/arquivo_20260116064120.txt

The other arguments confirm the injection target and installation paths:

- **Injection Target:** CasPol (defined twice in the arguments)
- **Install Directory:** C:\Users\Public\Downloads\
- **Fallback URL:** .../bkp

With these arguments prepared, the script invokes the Main method of the in-memory assembly, passing the configuration that drives the final phase of the attack.

Stage 3: The Persistence Module (A Dedicated .NET DLL)

Contrary to what one might expect in a simple infection chain, the payload extracted from the image file is not the XWorm RAT itself. Instead, it is a specialized VB.NET DLL designed with a single purpose: Survival.

This stage acts as a dedicated persistence module. It does not communicate with a C2, nor does it download files. Its job is to ensure that the infection survives a reboot by registering a Scheduled Task.

Evading Detection via .NET APIs

Most commodity malware takes the easy route: spawning `cmd.exe /c schtasks /create....` This is “noisy” and easily flagged by EDRs monitoring child processes.

This sample takes a stealthier approach. It abuses the Task Scheduler Managed Wrapper, interacting directly with the Windows Task Scheduler via COM interfaces (TaskService, TaskDefinition) within the .NET framework.

 DLL bypasses schtasks.exe
The DLL bypasses schtasks.exe by using .NET APIs to register persistence directly

By doing this, the malware leaves no command-line artifacts. To a defender looking at process logs, the task appears to “materialize” without a corresponding execution command.

The Infection Loop

The persistence mechanism reveals the modular nature of this campaign. The scheduled task created by this DLL does not launch XWorm directly. Instead, it is configured to re-execute the Stage 2 PowerShell loader.

 ANY.RUN scheduler
The created task ensures the PowerShell loader runs at logon, restarting the cycle

Stage 4: The XWorm Payload & CasPol Abuse

Following the configuration passed by the PowerShell loader, the final payload is retrieved from the URL https://voulerlivros.com.br/arquivo_20260116064120.txt.

Despite the .txt extension, the content is not plain text. It is a reversed Base64 string. This lightweight obfuscation technique can still be effective against content scanners that expect standard Base64 patterns. Once reversed and decoded, the resulting binary is a .NET executable identified as XWorm v5.6.



Reversing the text file reveals the valid PE header of the XWorm payload

Living off the Land: CasPol.exe Injection

The malware does not execute as a standalone process. Instead, it injects itself into CasPol.exe (Code Access Security Policy Tool), a legitimate binary located at C:\Windows\Microsoft.NET\Framework\v4.0.30319\CasPol.exe.



CasPol.exe binary is hollowed out to host the malicious payload

The legitimate CasPol.exe binary is hollowed out to host the malicious payload

By abusing this [“Living off the Land” binary](#) (LOLBIN), the malware attempts to blend in with trusted system processes. However, in the [ANY.RUN sandbox](#), this anomaly is immediately flagged due to the suspicious network activity originating from a trusted utility.

Cracking the Crypto (Static Analysis)

A deep dive into the payload using dnSpy reveals a critical flaw in the malware’s design. The configuration is encrypted using AES, but the implementation is weak.

- **Key derivation:** The AES key is generated by taking the MD5 hash of the Mutex string.
- **Mode of operation:** It uses AES-ECB (Electronic Codebook) mode.



dnSpy Crypto

The encryption key is derived directly from the Mutex string using MD5

Because the Mutex is hardcoded in the binary (or passed via arguments), the encryption is deterministic. This allows us to decrypt the configuration offline without needing to run the malware.

Decrypted configuration:

- **C2 Server:** jholycf100.ddns.com.br (152.249.17.145)
- **Port:** 7000
- **Mutex:** V2r1vDNFXE1YLV0A
- **Splitter:** <Xwormmm> (A unique fingerprint for XWorm)

Behavioral Confirmation (Dynamic Analysis)

The static findings are fully corroborated by the runtime behavior observed in [ANY.RUN](#).

- **Mutex Creation:** The sandbox logs show the creation of the mutex V2r1vDNFXE1YlWoA, confirming the exact seed used for our decryption.
- **C2 Traffic:** The process CasPol.exe initiates a TCP connection to jholycf100.ddns.com.br on port 7000.
- **Protocol:** The traffic stream contains the <Xwormmm> delimiter, matching the decrypted configuration.



Network traffic confirms the C2 destination and the custom XWorm protocol delimiter

Business Impact: What This Means for Companies

This isn't "just another XWorm." The risk comes from how reliably the chain can reach corporate endpoints and how quietly it can stay there. A fake receipt is the kind of lure that fits normal finance and ops workflows, and the delivery stack (WMI-spawned PowerShell, cloud-hosted content, fileless loading, and task-based persistence via .NET APIs) is built to reduce the early signals many teams depend on.

- **Credential and session theft → downstream compromise:** Once a workstation is controlled, attackers can harvest browser sessions and credentials and pivot into email, SaaS, and finance tooling, turning a single click into an identity-driven incident.
- **Higher blast radius, faster:** With persistence in place, the operator can take time, map the environment, and expand access, raising the likelihood of lateral movement and follow-on payloads.
- **Cost of delayed detection:** "Lower-noise" tradecraft tends to inflate MTTR because the initial event looks benign (image download, PowerShell in the background, no obvious dropped binary), while real impact surfaces later.
- **Operational risk, not just endpoint risk:** The outcomes aren't limited to one infected machine. The realistic worst cases are business email compromise, fraudulent payments, data access, or ransomware staging, each with direct financial and reputational consequences.

The takeaway is simple: this kind of campaign rewards **fast, evidence-based validation** at the first suspicious touchpoint (script/PowerShell execution + abnormal cloud-hosted "image" responses) and **strict monitoring of LOLBIN abuse** (e.g., CasPol.exe producing outbound traffic). Catching it early is what keeps a workstation event from becoming a business threat.

How to Set Up Early Detection of XWorm Attacks

Early detection of XWorm usually depends on how well the **SOC operational cycle** is working day to day. When monitoring, triage, and threat hunting are tightly connected, commodity RAT activity is far more likely to be contained before it turns into a real business incident.

1. Monitoring: Strengthen Visibility with TI Feeds

The first signal often appears in external infrastructure or newly observed indicators. [ANY.RUN's TI Feeds](#) help by continuously surfacing fresh XWorm-related domains, hashes, and behavioral patterns, based on telemetry and submissions coming from **15,000+ organizations and 600,000+ security professionals**.

 100% actionable IOCs delivered by TI Feeds
100% actionable IOCs delivered by TI Feeds to your existing stack

This makes it easier to spot suspicious activity earlier and push relevant IOCs directly into SIEM or EDR controls.

2. Triage: Enrich and Validate Alerts in Minutes

Once an alert or suspicious artifact appears, speed becomes critical.

- [TI Lookup](#) provides immediate enrichment, showing reputation, related samples, network relationships, and historical context around a file, hash, or domain.
- [Interactive sandbox analysis](#) allows teams to safely execute suspicious files or URLs and observe real runtime behavior, confirming XWorm activity within minutes rather than hours.



ANY.RUN's sandbox revealing full attack chains in just 1 minute

Fast, evidence-based triage reduces uncertainty and prevents unnecessary escalation while still catching real threats early.

3. Threat Hunting: Track Active Regional Campaigns

The next step in the cycle is proactive visibility. Using structured TI Lookup queries such as: [threatName:"xworm" AND submissionCountry:"br"](#) SOC teams can surface the latest XWorm samples observed in Brazil, review delivery techniques, and pivot into related infrastructure. This makes detection logic more relevant to the **current regional threat landscape**, not just historical global data.



TI Lookup shows analysis sessions related to XWorm attacks observed in Brazil

When these three motions operate as a continuous cycle rather than isolated tasks, XWorm shifts from a late discovery to an **early, manageable security event**, reducing response time, investigation cost, and overall business risk.

Conclusion

This campaign highlights a clear trend in LATAM-focused malware: pairing high-volume delivery vectors with established commodity RATs. While the XWorm payload itself relies on relatively basic cryptography (AES-ECB), the overall delivery chain is built for resilience.

By combining HTML/LNK delivery, Cloudinary abuse, steganography, and modular persistence (via .NET Task Scheduler APIs), the attackers have created a lower-noise infection chain that can bypass superficial defenses.

For defenders, detection opportunities exist at multiple stages:

- **Delivery:** Monitor for LNK/JS files spawning PowerShell.
- **Network:** Flag traffic to image hosting services (Cloudinary) where responses contain non-image headers or BaseStart markers.
- **Endpoint:** Alert on CasPol.exe initiating outbound network connections.

About ANY.RUN

[ANY.RUN](#), a leading provider of interactive malware analysis and threat intelligence solutions, fits naturally into modern SOC workflows, strengthening the day-to-day operational cycle across Tier 1, Tier 2, and Tier 3.

It supports every step of an investigation, from safely detonating suspicious files and links to see real behavior, to enriching indicators with broader context, to delivering fresh intelligence that helps teams act faster and with fewer blind spots.

Today, more than 600,000 security professionals across 15,000+ organizations use ANY.RUN to speed up triage, cut unnecessary escalations, and keep pace with fast-moving phishing and malware campaigns.

[Bring speed and clarity to your SOC with ANY.RUN](#)

Appendix: Indicators of Compromise (IOCs)

Network Indicators

- **C2 Domain:** [jho|ycf100f|.iddnsf|.com|.br](#)
- **C2 IP:** 152[.]249[.]17[.]145
- **Port:** 7000
- **Payload URL 1 (Stego Loader):** res[.]cloudinary[.]com/.../optimized_MSI_lpsd9p.jpg
- **Payload URL 2 (XWorm):** voulerlivros[.]com[.]br/arquivo_20260116064120.txt

Host-Based Indicators

- **Mutex:** [V2r1vDNFXE1YLWoA](#)

- **File Path:** C:\Users\Public\Downloads\
- **Target Process:** CasPol.exe
- **File hash:** 7bfeacf0b3480fb675d0cab7767b5b9697edc9d0e05982025a06ead0054afd5
- **Powershell:** Assembly.Load

Detection Opportunities – YARA Rules

YARA – Javascript Dropper:

This rule is designed as a **medium-to-high confidence hunting rule**, prioritizing behavioral and structural indicators rather than brittle IOCs..

```
rule JS_WSH_Unicode_Padded_Dropper

    meta:

        description = "WSH JavaScript dropper with Unicode padding and repeated assignment patterns"

        author = "0x0lympus"

        confidence = "medium-high"

    strings:

        $assign = "this." ascii

        $pad = {

            74 68 69 73 2E 76 61 74 66 75 6C 20 2B 3D 20 22

            E0 B2 92 E2 9C 96 C8 B7

        }

        $wsh = "Scripting.FileSystemObject" ascii nocase

    condition:

        /* Exclude PE files */

        uint16(0) != 0x5A4D and

        /* Script-sized payloads (not tiny JS snippets) */

        filesize > 1000KB and

        /* Must be WSH-based */

        $wsh and

        /* Obfuscation indicators */

        (

            $pad or

            $assign

        )

}
```

Key detection components:

- **Non-PE filtering**
The check `uint16(0) != 0x5A4D` ensures that only script-based files are evaluated, preventing false positives on executable payloads.

- **File size heuristic**

The condition filesize > 1000KB targets scripts that abuse **entropy padding**. Legitimate JavaScript files are rarely this large, especially when used as WSH droppers.

YARA – XWorm 5.6 Payload:

This rule targets the final XWorm RAT binary, using protocol and cryptographic fingerprints that are stable across XWorm versions.

```
rule XWorm_PE_v56
{
    meta:

        description = "XWorm RAT v5.6 .NET payload"

        author = "0x0lympus"

        family = "XWorm"

        version = "5.6"

        confidence = "very high"

    strings:

        // Protocol splitter (strong family fingerprint)

        $splitter = "<Xwormmm>" ascii

        // Cryptographic implementation

        $crypto1 = "RijndaelManaged" ascii

        $crypto2 = "MD5CryptoServiceProvider" ascii

        $crypto3 = "CipherMode.ECB" ascii

        // Network functionality

        $net1 = "System.Net.Sockets" ascii

        $net2 = "NetworkStream" ascii

    condition:

        uint16(0) == 0x5A4D and

        filesize < 5MB and

        $splitter and

        2 of ($crypto*) and

        1 of ($net*)
}
```

Note: The <Xwormmm> splitter combined with AES-ECB + MD5 key derivation provides a near-unique signature for XWorm, resulting in very low false-positive risk.



Malware Researcher & Threat Hunter with a strong background in Blue Team operations. Specialized in malware analysis and reverse engineering, with hands-on experience dissecting binaries and reconstructing attacker TTPs from initial delivery to command-and-control communication. Driven by a deep interest in adversary tradecraft, bridging low-level technical analysis with strategic threat intelligence and detection engineering.

 moises-cerqueira-0xolympus

Moises Cerqueira (0xOlympus)

Malware Researcher & Threat Hunter with a strong background in Blue Team operations. Specialized in malware analysis and reverse engineering, with hands-on experience dissecting binaries and reconstructing attacker TTPs from initial delivery to command-and-control communication. Driven by a deep interest in adversary tradecraft, bridging low-level technical analysis with strategic threat intelligence and detection engineering.

Follow Moises on:

[LinkedIn](#)

[X](#)

[Website](#)

Source: <https://any.run/cybersecurity-blog/xworm-latam-campaign/>