Evolve: Quietly Creeping into Remote Hosts

April 28, 2017 | 10 minutes read | James Antonakos

Introduction

I recently engaged in an investigation involving two new Carbanak campaigns targeting the hospitality sector. In each campaign a malicious Word or RTF document was sent to specific employees, claiming the sender had trouble with the online ordering system, or was filing a lawsuit because a member of his group got sick after dining at one of the organization's restaurants. This social engineering scam is augmented with a personal phone call from the attacker, encouraging the intended victim to open the email attachment and click inside it. The attacker then calls back 30 minutes later to check if the document was opened and hangs up as soon as the employee says yes.

Phishing continues to be an effective attack vector, and a number of systems at the organization became infected as a result of opening the attachment. This time around, however, there are several twists added to the usual credit-card scraping and screen-shot grabbing malware previously seen in the November 2016 Carbanak campaign published in the Trustwave SpiderLabs blog (https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/new-carbanak-anunak-attack-methodology/).

I want to provide my sincere thanks to Rodel Mendrez,Trustwave SpiderLabs Senior Security Researcher, for all his assistance and knowledge. From malware reversing, to writing custom Python scripts, to historical knowledge of exploits, his help was invaluable to unraveling the many layers of trickery employed by the Carbanak malware writers. Thanks also to Brian Hussey,Trustwave SpiderLabs, VP of Cyber Threat Detection and Response and Mike Wilkinson, Trustwave SpiderLabs IR, North America Managing Consultant, for their review and guidance.

Did You Open The Attachment?

The infected email attachment is a Microsoft Word docx or RTF file. These documents, and their MD5 hashes, have appeared across several campaigns:

Name	MD5 Hash	File Size
2-reservation-1.docx	D5FBFD2D1D71[illegible]	[illegible] bytes
Claim_Dave.docx	[illegible]BF653B89A08[illegible]	[illegible] bytes
Letter 216 micro.docx	FB442B402F73[illegible]	[illegible] bytes
4404_inf04.docx	[illegible]C12B6A6EBFFFDA75E470706	395,857 bytes
Ord2.rtf	ABAB101E4E25A989D2943C3	1,075,043 bytes

Let's take a look at what the most recent document, Ord2.rtf, puts into motion when opened. Note that all of the files listed here are contained within the malicious RTF document and extracted when it is opened:

So that is two VBS and one PS1 file created by the infected RTF document. Persistence is established by creating a scheduled task that runs the main malware file every 25 minutes.

```
Creates folder:
%HOMEPATH%\Intel
Creates:
%HOMEPATH%\Intel\58d2a83f7778d
5.36783181.vbs --- Persistence
script
```

```
%HOMEPATH%\Intel\58d2a83f77794
2.26535794.ps1 --- Malware
downloader
%HOMEPATH%\Intel\58d2a83f77790
8.23270411.vbs --- Command-
and-Control (CnC) malware
creator
Scheduled task "Intel" that
runs the persistence script
every 25 minutes.
Runs:
The malware downloader if it
has not run yet.
The CnC malware creator script
which creates additional
malware and connects to
198.100.119.6
via port 80, 443, or 8080.
Notes:
The persistence script runs
the malware downloader if it
has not run yet.
The malware downloader injects
400+ bytes of code into
process memory.
This code contains the domain
name
aaa.stage.14919005.www1.proslr
3.com and retrieves and
executes shellcode stored in
encoded DNS TXT records.
```

But the malware is not finished yet. The CnC malware creator script &nbsp;creates additional malware and support files in a different folder:

```
Creates folder:
%HOMEPATH%\Intel\{BFF4219E-
C7D1-2880-AE58-9C9CD9701C90}
Creates:
%HOMEPATH%\Intel\
{BFF4...1C90}\58d2a83f777638.6
0220156.ini --- Command/Status
file
%HOMEPATH%\Intel\
{BFF4...1C90}\58d2a83f777688.7
8384945.ps1 --- Screenshot
script
%HOMEPATH%\Intel\
{BFF4...1C90}\58d2a83f7776b5.6
4953395.txt --- Base64
container file
%HOMEPATH%\Intel\
{BFF4...1C90}\58d2a83f7776e0.7
2726761.vbs --- CnC script
%HOMEPATH%\Intel\
{BFF4...1C90}\58d2a83f777716.4
8248237.vbs --- INI processing
script
%HOMEPATH%\Intel\
{BFF4...1C90}\58d2a83f777788.8
6541308.vbs --- script updater
%HOMEPATH%\Intel\
{BFF4...1C90}\58d2a83f7777b0.3
9039198.vbs --- CnC
persistence script
Runs:
CnC script if it has not run
yet.
```

```
The INI processing script.
The CnC persistence script.
Notes:
The Command/Status file is
used to receive commands from
CnC and maintain malware
status.
The screenshot script saves a
screen shot to
"screenshot.png" when run.
The Base64 container file
contains one or more lines of
base64 code used to update
these files:
Line 1: Command/Status file
Line 2: screenshot script
Line 3: CnC script
Line 4: INI processing script
The CnC script provides CnC
communication and command /
status.
Connects to 198.100.119.6 via
port 80, 443, or 8080
The INI processing script
parses and processes INI file.
Commands: ScreenShot, runvbs,
runexe, runps1, update,
delete.
Delete command deletes all
INI, PS1, VBS, TXT files,
deletes Run key, and deletes
scheduled task.
The script updater reads
base64 strings from the Base64
container file and uses them
to:
Create any .ini, ps1, or .vbs
files that do not yet exist.
Then runs the CnC script.
Then runs the INI processing
script.
The CnC persistence script
creates Foxconn.lnk which
points to the script updater.
Adds HKCU Run key CtMgk2y9v0_
for Foxconn.lnk.
Creates scheduled task
"ZSUUvsCJNF" that runs the
script updater every 25
minutes.
Then the CnC persistence
script deletes itself.
```

In addition to another PS1 file and four more VBS scripts, we also see INI and TXT files being created. What are these for? The answer is coming shortly.

So much malware from one little RTF and a simple socially-engineered click to start everything in motion. This is why continuous security awareness training is essential.

When you look at all the functionality built into those VBS and PS1 files you can see why Carbanak continues to be a significant threat. Even though some of the PS1 and VBS code is the same or very similar to that of previous attacks, there are some new twists in these files, and they are dramatic.

Bring Back the INI Files

This new Carbanak campaign also resurrects the INI file, using it to issue commands to the compromised machine as well as to reflect the status of previous commands. INI files are text files used in earlier versions of Windows that contain software configuration details organized into sections, properties, and values. Here is an "in-use" INI file from the investigation:

The "last_cmd_id=2" entry indicates that the malware has transmitted the victim's system information and a list of currently running processes back to the attacker, and is waiting for a new command to process. This is quite clear from the following code snippet:

System info sent to the attacker contains the following items:

- OS Name, Version,&nbsp;&nbsp; Service Pack,&nbsp;&nbsp; OS Manufacturer,&nbsp;&nbsp; Windows Directory,&nbsp;&nbsp; Locale
- Available Physical Memory, Total Virtual Memory,&nbsp;&nbsp; Available Virtual Memory
- OS Name, System Name,&nbsp;&nbsp; System Manufacturer,&nbsp;&nbsp; System Model,&nbsp;&nbsp; Time Zone
- Total Physical Memory, Processor System Type,&nbsp;&nbsp; Processor,&nbsp;&nbsp; BIOS Version
- Microsoft Office Apps, Computer name,&nbsp;&nbsp; Domain,&nbsp;&nbsp; User name

The INI processing script parses and processes the contents of the INI file, providing the following commands:

- Screenshot (save screenshot as screenshot.png)
- Runvbs

- Runexe
- Runps
- Update
- Delete

The Screenshot command takes a screenshot of the entire Desktop and saves it to the folder where the malware is installed. Here is a sample screenshot created by running the screenshot script:

Depending on what a user has opened on their Desktop, the screenshot could end up providing the attacker with valuable information.

The "update" command is particularly interesting, as it is used to update between one and four of the VBS / PS files created by the infected email attachment.

The files are updated as illustrated in the following figure:

After the files are updated, the new versions of the VBS files are executed to provide the new functionality.

The INI processing script communicates with the attacker's CnC server after checking the victim's proxy settings via examination of this Registry key:

`HKEY_CURRENT_USER\Software\Microsoft\Windows\Current\ Settings\ProxyEnable`

The attacker's CnC IP address is hardcoded into the VBS file:

```
hxxp://198.100.119.6:80/cd
hxxp://198.100.119.6:443/cd
hxxp://198.100.119.6:8080/cd
```

This IP address is managed by GorillaServers, Inc.

For the exfiltration method, the malware uses "http POST" to send data that has been retrieved/stolen from the infected system to the CnC server.

Information is sent using randomized variable names and simple encryption on the text-based data using base64 encoding and substitution encryption with these alphabets:

```
alfIn =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabc
defghijklmnopqrstuvwxyz0123456
789"
alfOut =
"860vFjbqCRty2BUrpAGkEMLVKYJ97
1eNDThfilxugP5WHOSsnm3QIcdZwX4
oza"
```

For example, to exfiltrate the string "Windows7" the following occurs:

Step [1]: Convert "Windows7" to base64: "V2luZG93czc="

Step [2]: Add "V2luZG93czc=" to exfiltration message: "info%SEPR%V2luZG93czc="

Step [3]: Convert "info%SEPR%V2luZG93czc=" to base64: "aW5mbyVTRVBSJVYybHVaRzkzY3pjPQ=="

Step [4]: Encrypt "" using alfIn and alfOut:

&nbsp;&nbsp;&nbsp; <a>&nbsp; Get symbol to convert. Example: "a"

&nbsp;&nbsp;&nbsp; <b> Look up index value of symbol in alfIn. Example: alfIn[26] = "a" (see red input symbol above)

&nbsp;&nbsp;&nbsp; <c> Substitute symbol from alfOut at same index. Example: alfOut[26] = "J" (see red symbol above)

&nbsp;&nbsp;&nbsp; <d> For any input symbol not found in alfIn, just add symbol to output as is. Example: "=" is unchanged.

&nbsp;&nbsp;&nbsp;
"aW5mbyVTRVBSJVYybHVaRzkzY3pjPQ=="
encrypts to
"JLXx93MkAM6GRMK39qMJAQiQKZPfrp=="

Step [5]: URL encode
"JLXx93MkAM6GRMK39qMJAQiQKZPfrp=="
into
&nbsp;"JLXx93MkAM6GRMK39qMJAQiQKZPfrp%3D%3D"

This information, along with other items, such as encoded and encrypted IDs based on the hard-drive serial number and the software version of the malware, are bundled into various "parameter=value" strings and sent to the CnC sever.

The parameter names are randomly generated and match the regular expression **[a-z]{1,2}[a-z0-9]{1,6}** and there are also random data strings generated and transmitted (presumably to confuse anyone decoding them) that correspond to the regular expression **[a-z0-9]{1,8}**.

Here is information from my own analysis system that is gathered by the malware and transmitted:

```
CUID: sn = 907316415
Encoded sn = OTA3MzE2NDE1
id_b = 1-1-OTA3MzE2NDE1
CompInfo():outData =
OS Name: Microsoft Windows 10
Pro|C:\WINDOWS|\Device\Harddis
k1\Partition1
Version: 10.0.14393
```

```
   Service Pack: 0.0
   OS Manufacturer: Microsoft
   Corporation
   Windows Directory: C:\WINDOWS
   Locale: 0409
   Available Physical Memory:
   30120564
   Total Virtual Memory: 38472844
   Available Virtual Memory:
   34617616
   OS Name: 4980736
   System Name: JLA-FORENSICS
   System Manufacturer: Dell Inc.
   System Model: Precision M4800
   Time Zone: -300
   Total Physical Memory:
   34295918592
   Processor System Type: 9
   Processor: Intel64 Family 6
   Model 60 Stepping 3
   BIOS Version: DELL - 1072009
   Perform : 64-bit
   %UsrNmCmpNmDomNm%
   Networking information
   Computer name : JLA-FORENSICS
   Domain : jla-forensics
   User name : jantonakos
```

This information is processed using the 5-step sequence just described, giving:

```
   JLXx93MkAM6GREAA7j6MeERrLM7m9F
   aPpiXTMIXXKfBU1lPSEL1L2xmcLi7X
   2dBXpVTBpIRAKdIXUjj47bBL2bmr
```

```
EiEXLjEQebBGAcK3KM1U9jTjJbTf9M
RWKMTU7icLejjYLFgnKMTG7bC3BqTF
EV6VLlTtexjVUVMrJEjwkE2I1Im5
Ak6BexO4AjjnMjPKGfRTMIXHGEY0Jj
i37QYRAFjckEFnGcpmkL1EMIYcYj1J
JjiQEfjf9MYXkdl0kxjVkulh2IXd
LxXAYcF3UVlfAQlXLMTG7bC3BFXv9b
AnKxcG1xpQkL1GAdmXLl1U2bCQGfMr
JERFkdmwLjBMBEME2LAEAjjnkLC3
kxThAcEdGEA62FcFJIXvJIK3LM1H7c
lVGuBJEIRAKETHexjVkxThpIRULl7m
1xBSJQYRAFcZkMAR1IXELk6FEV6M
KfBGJbRvpl1TLFgnYj1b7Ilj2LmhMQ
lXYMAsYIc4YQ6UeiiIkiAAkiBWAfRY
MdmQLM1t7cPkpl1TLFgnYj1b7Ilj
2LmhMQlXYMAsYIc4EkRBMb23kMAYki
BWUMARAkMgKl1MBilFEkMrAFFQkVPY
kiBHkfMf2cRHKlB0kclV2LmrJERy
MFMj1jRWUMBGMkMEEcMUMFAA7jAlLF
wnLl7nYcAVAuMiMcPgLkBG2LBOMulr
JERjLl1w7Ilj9qMYekAUpdmUBL2Q
ExmhEIRUKfRG9bRF9d1MGFPHLkRHex
jVUVMRAk8nkIA61IAA7jMTMQjHGEYn
1xROMkYRpQ64kEA6kiBHEuYiAIYQ
GEY09dMKku6Y2iYQGEEm9bRVUVllMb
aukVPAeEaEMkMBMb7mkcARkiBHpulh
2iXHKQBU1xBPplAlLFwnLl7nYcYC
9q1JMbaukcFnGcMCGuYY2lY4KQCXeE
aPpiPh9lRHKiAY2FljLxThMdmQYMB6
2ilj2VYJAcYQGEAY1Ilbkf6JLFRZ
KM7c9ilFkEXvJIPtMvjBYcYOMulf2x
mdKxPsYcRjMicEpIjuGE2nYIcEpkBB
JijZkcFnGcMqMulJ9klXKlB6BilF
Lk62MIPnYFFnGIPLMuPfJQMEkCm1cA
O2EMh2fjrKlBMkiBWBLmiGbAdKdcI7
bROKd1TMQMOKfBt1jlKEu6h2fAUp
```

```
d7nGcF3UVAfGjKnLlTRYdROAuAJEIF
dGEMnkMjk2E1E2EPbMbmUGlFmkEXvJ
cRdKl1b7bRPpkYRAZ6QLM2m9LCQG
xmh9iXnLkBBkiBHMuPJLFluKxcb1jP
kpkYRAZ6gKxXG1xROAuRh2IcUpd7ar
p%3D%3D
```

And here we see similar information from a different system on its way to the attacker:

Knowing the format of the messages sent to the attacker may assist in developing alerts for your organizations SIEM, firewall, or other security monitoring tools.

Lateral Movement Without Passwords

Unlike previous campaigns, where Mimikatz or some other credential stealer was used, lateral movement between systems is accomplished without user accounts and passwords. Instead, the malware bypasses authentication on the remote system and uses SMB commands (including TreeConnect and Open/Write AndX) to locate a vulnerable host by checking the ability to write data to the C:\Windows\Temp folder on a potential victim system. If the write operation is performed, the attacking system writes a short batch file to the victim and runs it remotely via another SMB command (DCERPC). The batch file creates a log file whose contents are either "x86" or "amd64" depending on the bit-size of the operating system on the victim.

The batch file looks like this:

The attacking system reads the log file and uses its contents to then transfer a 32-bit or 64-bit executable file to the victim's C:\Windows\Temp folder and run it. The executable acts as a stager and enables the transfer of the meterpreter DLL to the victim system. The meterpreter DLL is made operational via reflective DLL injection, which copies the meterpreter DLL into the memory space of a chosen victim process (such as explorer.exe) and runs it as a thread.

The entire sequence of operations looks like this:

The {~random} file names are not really random, but instead a base64 combination of the victim's system volume serial number and MAC address, which are obtained by queries to the system's WMI Service. The IP address 5.149.251.167 is used to enable a reverse_TCP meterpreter shell back to the attacker' system, allowing for complete control over the victim computer. The attacker system at 5.149.251.167 is most likely running a meterpreter listener that transfers an XOR-encrypted meterpreter DLL to the victim and provides the quiet and authentication-free lateral movement between systems the attacker wants.

This WHOIS lookup of 5.149.251.167 indicates that this IP address should raise red flags.

A total of 606 SMB messages back and forth between the attacking and victim systems are needed to process the 9 infection steps previously described, and takes 105 seconds to complete, as the stager EXE is 59 KB. The stager is one of many that have been created to enable victim-CnC communication using metasploit. The following Wireshark screenshot shows a small portion of the SMB messages that transfer the stager to the victim:

Malware Transfer Via DNS TXT Records

A PowerShell script created by the infected email attachment contains a base64-encoded block of text that is decoded, written into memory as a thread, and executed. This uses the same technique as the metasploit 'DNS TXT Record Payload Download and Execution' module, as described below (with slightly modified code):

This code performs DNS queries to the attacker's malware domain, as illustrated here:

A 255-character TXT record is received in response to each query. For example, a query to:

aaa.stage.3553299.s1.rescsovwe.com

returns the TXT record:

```
WYIIIIIIIIIIIIIII7QZjAXP0A0Ak
AAQ2AB2BB0BBABXP8ABuJIIljHs0wp
wpuPjK0NaMlK1UePLycWrOU72ONcyU
va
4yhHd5iaLCKuWsaE6QzVP1ioTqKkCV
nkClduUPNY51U1gpUP5i8KbTVCTq9P
eakbLHhZk0XjBaK9zpiQ9p7t4qIBk8
Ij
K0ZJW17q8PnhaTEmc0os9WEQosIVfb
hkhM68ioYp8hnMyoYoIoAAgogoENFK
OIAAAAAAAAF
```

The next query is to subdomain baa, which returns:

```
LFCEFFFIJOFIBMDFCHDAAAAPPNDIJM
DFHGIAEAAAAAAFAPPNAGIPALFKCFGG
IAFAAAAAAFAPPNDAAAAAAAAAAAAAAA
AA
AAAAAAAAAAAAAPAAAAAAAAAOBPLKAOA
ALEAJMNCBLIABEMMNCBFEGIGJHDCAH
AHCGPGHHCGBGNCAGDGBGOGOGPHECAG
CG
```

```
    FCAHCHFGOCAGJGOCAEEEPFDCAGNGPG
    EGFCOANANAKCEAAAAAAAAAAAAAADID
    ICCBAHMFJEM
```

Note that frequency analysis of the symbol distribution in the records may be helpful in generating a rule for detection, as well as also noting if additional systems are performing the same queries. From a detection perspective, it is important to note that the TXT records are specifically requested in the DNS query, as shown here:

This is not typical DNS behavior.

As you refer back to the first TXT record you'll see the majority of the symbols are colored differently. You may also notice there are alphanumeric symbols in this portion of the TXT record that do not appear in the remainder of the record, or in fact, in any of the remaining 1,542 TXT records (as the DNS queries cycle through caa, daa, and so on up to zaa, then rolls over to aba, bba, caa, etc. until all the TXT records have been obtained).

The alphanumeric portion of the first TXT record is alphanumeric shellcode. The advantage of using alphanumeric shellcode is that it flies under the radar of AV and IDS/IPS, even though the ASCII codes associated with the alphanumeric characters represent actual 80×86 opcodes. Here are a few examples:

Here is a portion of the alphanumeric shellcode decoding loop (original alphanumeric shellcode and decoded shellcode):

The shellcode decodes itself to create more useful and efficient instructions needed to decode the remaining TXT records that contain the payload. The payload is undergoing detailed analysis and additional information will be forthcoming.

Note: a previous Carbanak campaign used this base domain:

aaa.stage.14919005.www1.proslr3.com

Both of these DNS malware domains are

registered in Russia. As with many malware delivery domains, they operate for a short period of time and then are shut down as the attackers move to a different domain.

Now, consider this: the number of DNS queries to the three-letter subdomains for the malware analyzed in this new campaign is 1,543. That is a very significant number of 255-character TXT records and over 385 KB of encoded characters. In one test the time to perform the 1,543 queries is 214 seconds, or 7 queries per second.

This is an example of "program behavior" rather than "human behavior." Security analysts would agree that this clearly indicates a human was not sitting at a computer rapidly clicking links for over three minutes straight.

Recognizing this type of behavior is a valuable addition to an organization's malware alerting controls. A continuous stream of DNS queries from the same system to the same TLD exceeding 420 queries per minute or some similar threshold would be a suitable alerting mechanism (not for a busy server perhaps but certainly for a user workstation).

Layers Upon Layers

The Carbanak malware writers have used several methods to hide the functionality of their malware. Consider all of these steps required to fully deliver just one of the malware files in this campaign:

1. PowerShell script file created from base64 encoded string hidden in

malicious Word document.

2. Decoding the final PowerShell script requires three levels of decoding.
3. The final PowerShell file contains a base64 encoded string representing 80×86 machine code that implements a DNS TXT record exploit, and is loaded into RAM and executed.
4. Alphanumeric shellcode and encoded payload are obtained via DNS TXT records.
5. Alphanumeric shellcode contains an initial decoding stub that XOR decrypts a portion of the shellcode into the payload decoder.
6. The payload decoder uses the remaining TXT record data to create an in-memory DLL file, thus ensuring that this attack does not leave malware traces on the system itself, only in memory.

The operation of the payload decoder is very simple: Each pair of ASCII characters become a single byte of code in the final payload. As shown in the following figure, each ASCII code in a pair has 'A' (0×41) subtracted from them. The resulting difference is always between 0 and F and becomes the high-order or low-order nibble of the final payload byte.

In addition to the base64 encoding, other files are also encoded and obfuscated. For example, the script updater is initially created by decoding a base64 string. But looking into the file shows these two encoded and obfuscated sections:

Take a moment to examine the symbols in the string that represents the encoded VBS statements. Then look at the symbols in the decoding key, and perhaps you'll see where I am headed.

Did you notice anything strange? If you missed it, take a look here:

Only the first 16 symbols in the decoding key are used to decode the VBS statements string. The second group of 16 symbols do not appear anywhere in the VBS statements string. In fact, the only place all those symbols appear is in a second encoded string that is at the end of the encoded and obfuscated VBS file, and this string is not referenced or processed. Decoding the string results in binary mumbo-jumbo and the purpose of this unused string is not known at this time.

If you are a "code talker" and want to give the second string a go, here it is:

```
"h8u74y828mcd89t7o3n5rr2i8uyya
i1o5wih80suwavtfzu2aszcqsnltmi
jv22sa0ti/"_
&"9fhpd9ccjslgo3i2zgwrio8ot7w0
ouarnauyh5ornfflamdyrmm7q1r2sy
pvm220rgv/"_
&"c9o90ivp2o3qeo45jyinmzc3rp3f
zp1p9nrpsncd4og2tl0yt1s3f2o7s5
n3y9i4c/"_
&"514wg5owsscyjp3mr1an2h33yg5z
fgp7rssejd3nmy787m8gh011inoe4h
o7tgl4ypzm9/"_
&"u9w9yrnu1z158l59ozrryln0wdes
lmj7mc1ugjwc32t1c5d0j1u4tr1v2a
24zuvivg/"_
&"5solssu4qr8woavp3wyz4hrwjfoi
```

```
su3zrr5lph4vftul2zi5mtmnh0ssl9
maa4pmd7oyya/"_
&"q0rhm1mnrhzde2rmulasifdnho7q
zf0l4vp0pa392j439ljcfwg5wd2fai
9f40ohgd/"_
&"i33qhaqmoc1n8svjswzuw4had8cw
uocg3eqo2vrwv832o17jn7wyzp0oap
lntjlwpgayu/"_
&"icle7mrmr0cgyz4cyyr7poodefea
m2rflfjzhjznstuyan0v1ojf7og1nn
up5m2dltst3s5r/"_
&"w111uv0wqz0h770devyygv895wjg
z4vlc25l4lp4v0m47jcru28n808moi
l9g2iyajc/"_
&"gy9j9vj0jjdy1yvhmynruqh1j29r
fugw3hwdyrqinai8tvavzmqidhwo3c
4zluypo3u/"_
&"l8vg470m1hyp7dhjyqamqrc9owg9
sjup99w9rnvgggpfhpvhfaileyuoq4
wr0mozdc/"_
&"jjdg830wiafscumvcw3flyfj3nsl
iwahtu8a3y4pugpd9aq95r4lc7lg1a
zl9fgf3umu0s28i2u4/"_
&"g32pdv8lzhs4ny3rt5i3ir8vc22z
o3iifyil77qncc3pthigrac01ge352
29jy88dmh8je/"_
&"f2ohgefdy7g9pthifu2ia0vgh31d
zpq9mi598ras14cf745y8jp19rpoof
y2wd2dn12vltwwv/"_
&"syf27jq7ttlpleafv340n8imonfh
1himrprjg9gjozluscrghgzouuh9td
91cclrtr0v3/"_
&""
```

The deobfuscated decoder looks like this:

The obsPayload1 variable is set equal to the long encoded string representing the actual VBS statements. Leaving the VBS files encoded like this while they reside on disk is an attempt to prevent detection by AV scans looking for suspicious keywords. Note that the last statement in the DeObs1() subroutine should be

WScript.Run DecodedPayload1

but was changed to WScript.Echo to simply view, rather than execute, the code.

Watch The Compromise Unfold

If you have access to CarbonBlack, the chain of events and spawned processes can be traced from the moment the malicious Word document is opened. Here is a timeline of an actual compromise:

and/or prevent actions we do not want taking place.

Organizational Preparedness

There are several steps an organization can take to prevent or limit the severity of a Carbanak attack. All of the following steps apply and can be achieved via internal security engineering or by engaging an MSSP:

- Regular security awareness training for all employees, paying particular attention to spear phishing.
- Spear phishing exercises where employees are sent a 'phishing' email that points to a site controlled by IT (Trustwave SpiderLabs also offers this service).
- An email server or appliance that can assist with malware detection, such as scanning incoming email attachments for base64 strings.
- Macros disabled by default on all Office applications (although a user can still re-enable them).
- A SIEM or other log-and-event aggregation system that allows aggregated network traffic to be examined by an expert security team before, during, and after an attack.
- Ensuring that IDS rules are able to detect metasploit modules.
- Threat intelligence driven software restriction policies, such as preventing program execution from C:\Windows\Temp.
- Whitelist PowerShell scripts and VBS

scripts used by the organization and blacklist all others.
- Continuous DNS monitoring with threshold alerts for systems issuing excessive DNS queries in a given period of time.
- Restrict DNS traffic so that internal systems are only able to query your DNS servers.

Conclusion

The Carbanak campaign is not standing still and neither is Trustwave SpiderLabs. Look for another update shortly.

Latest SpiderLabs Blogs

Trustwave Rapid Response: CrowdStrike Falcon Outage Update

Trustwave is proactively

Using AWS Secrets Manager and Lambda Function to Store, Rotate

Facebook Malvertising Epidemic – Unraveling a Persistent Threat: SYS01

The Trustwave SpiderLabs Threat