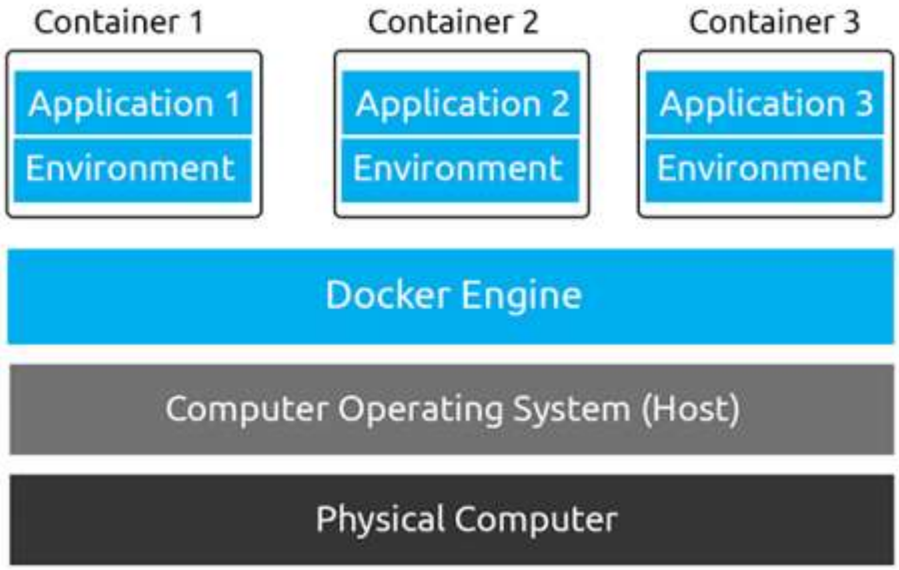
Situational Awareness Docker?

Now that we have gained a shell onto the webserver, we need to perform some situational awareness to figure out where we are. We know from looking through some files when we exploited LFI that this may be a container. We can run some further enumeration and information gathering to identify whether that is true or not and anyway misconfigurations that might allow us to escape the container.

From the Docker documentation, "A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings."



Containers have networking capabilities and their own file storage. They achieve this by using three components of the Linux kernel:

- Namespaces
- Cgroups
- OverlayFS

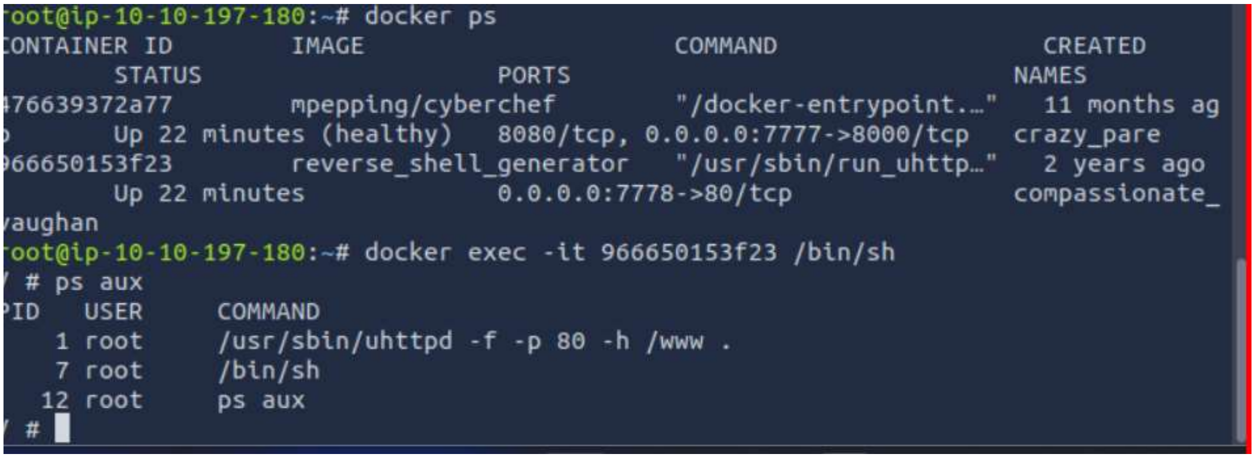
But we're only going to be interested in namespaces here; after all, they lay at the heart of it. Namespaces essentially segregate system resources such as processes, files, and memory away from other namespaces.

Every process running on Linux will be assigned a PID and a namespace.

Namespaces are how containerization is achieved! Processes can only "see" the process that is in the same namespace - no conflicts in theory. Take Docker; for example, every new container will be running as a new namespace, although the container may be running multiple applications (and, in turn, processes).

Let's prove the concept of containerization by comparing the number of processes there are in a Docker container that is running a web server versus the host operating system at the time.

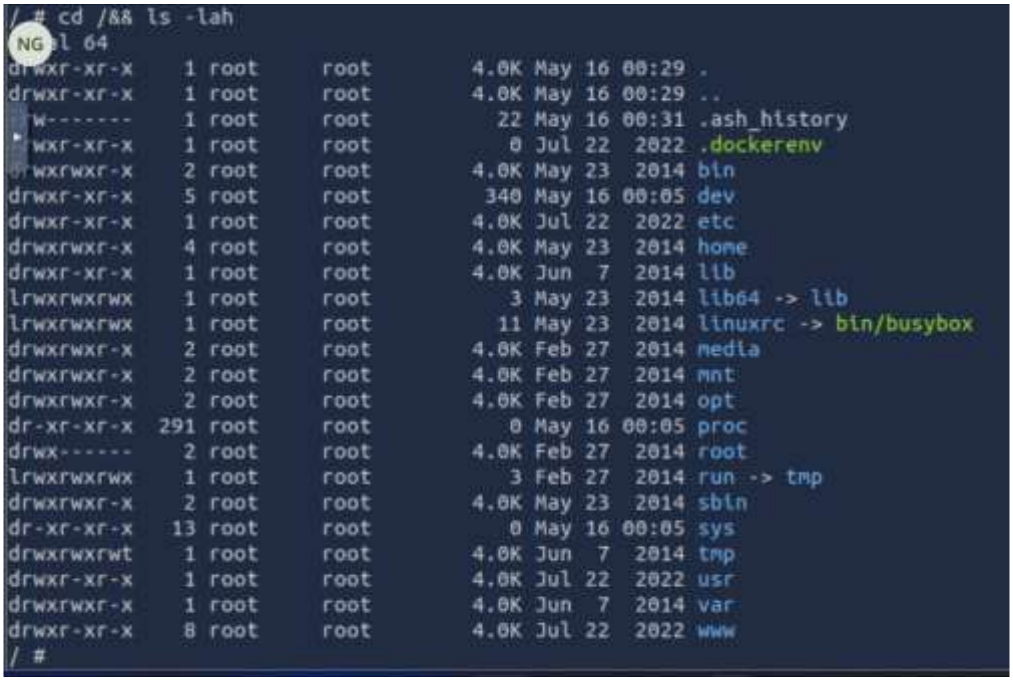
We can look for various indicators that have been placed into a container. Containers, due to their isolated nature, will often have very few processes running in comparison to something such as a virtual machine. We can simply use **ps aux** to print the running processes. Note in the screenshot below that there are very few processes running?



```
oot@ip-10-10-197-180:~# docker ps
ONTAINER ID        IMAGE                        COMMAND                  CREATED
       STATUS                    PORTS                                   NAMES
76639372a77        mpepping/cyberchef         "/docker-entrypoint.…"   11 months ag
o       Up 22 minutes (healthy)   8080/tcp, 0.0.0.0:7777->8000/tcp   crazy_pare
966650153f23        reverse_shell_generator    "/usr/sbin/run_uhttp…"   2 years ago
       Up 22 minutes             0.0.0.0:7778->80/tcp               compassionate_
vaughan
oot@ip-10-10-197-180:~# docker exec -it 966650153f23 /bin/sh
/ # ps aux
PID   USER     COMMAND
    1 root     /usr/sbin/uhttpd -f -p 80 -h /www .
    7 root     /bin/sh
   12 root     ps aux
/ # █
```

Containers allow environment variables to be provided from the host operating system by the use of a .dockerenv file. This file is located in the "/" directory and would exist on a container - even if no environment variables were provided.

Command used: cd / && ls -lah

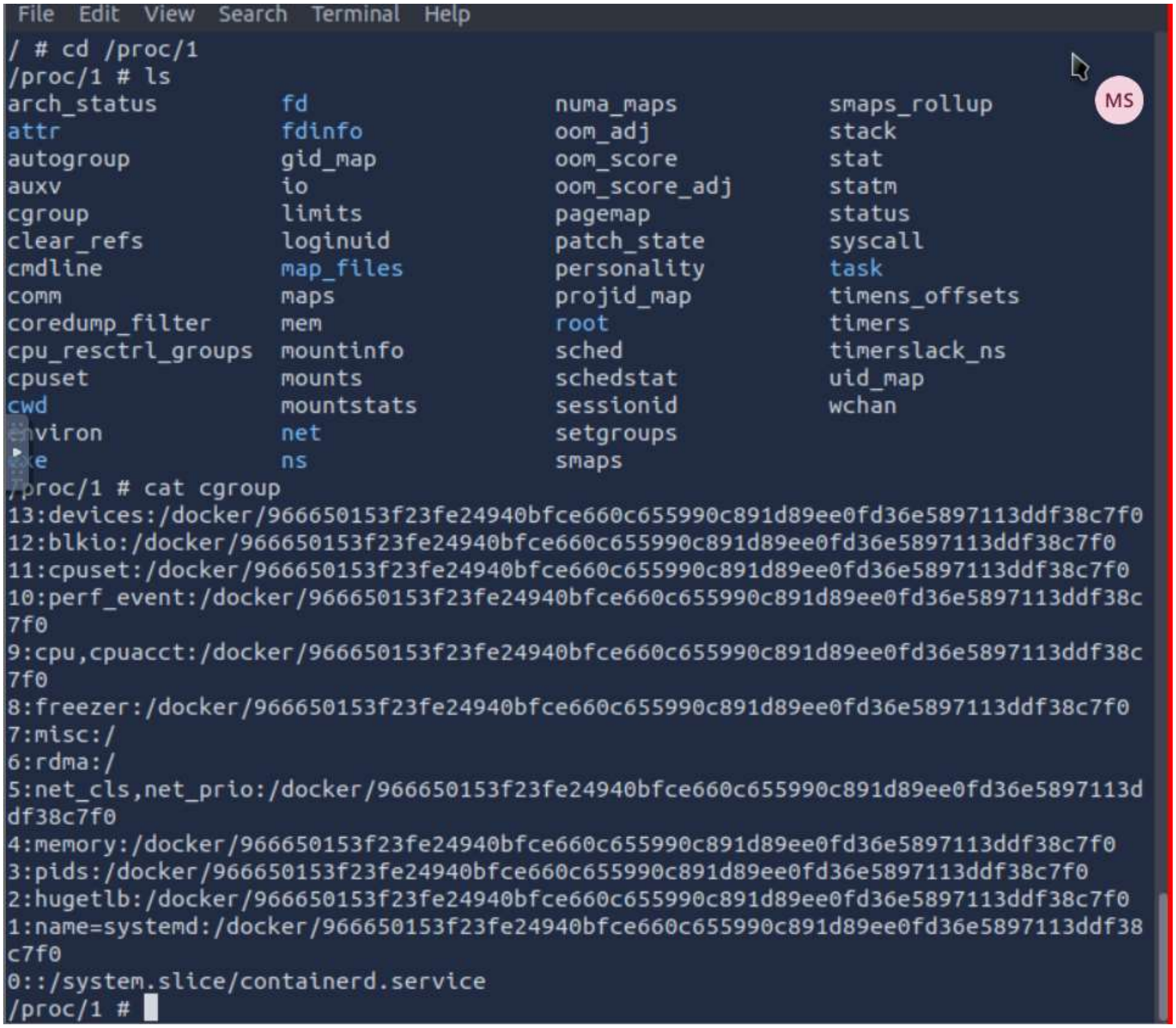


```
/ # cd /&& ls -lah
   l 64
drwxr-xr-x    1 root     root        4.0K May 16 00:29 .
drwxr-xr-x    1 root     root        4.0K May 16 00:29 ..
  w-------    1 root     root          22 May 16 00:31 .ash_history
  wxr-xr-x    1 root     root           0 Jul 22  2022 .dockerenv
  wxrwxr-x    2 root     root        4.0K May 23  2014 bin
drwxr-xr-x    5 root     root         340 May 16 00:05 dev
drwxr-xr-x    1 root     root        4.0K Jul 22  2022 etc
drwxrwxr-x    4 root     root        4.0K May 23  2014 home
drwxr-xr-x    1 root     root        4.0K Jun  7  2014 lib
lrwxrwxrwx    1 root     root           3 May 23  2014 lib64 -> lib
lrwxrwxrwx    1 root     root          11 May 23  2014 linuxrc -> bin/busybox
drwxrwxr-x    2 root     root        4.0K Feb 27  2014 media
drwxrwxr-x    2 root     root        4.0K Feb 27  2014 mnt
drwxrwxr-x    2 root     root        4.0K Feb 27  2014 opt
dr-xr-xr-x  291 root     root           0 May 16 00:05 proc
drwx------    2 root     root        4.0K Feb 27  2014 root
lrwxrwxrwx    1 root     root           3 Feb 27  2014 run -> tmp
drwxrwxr-x    2 root     root        4.0K May 23  2014 sbin
dr-xr-xr-x   13 root     root           0 May 16 00:05 sys
drwxrwxrwt    1 root     root        4.0K Jun  7  2014 tmp
drwxr-xr-x    1 root     root        4.0K Jul 22  2022 usr
drwxrwxr-x    1 root     root        4.0K Jun  7  2014 var
drwxr-xr-x    8 root     root        4.0K Jul 22  2022 www
/ #
```

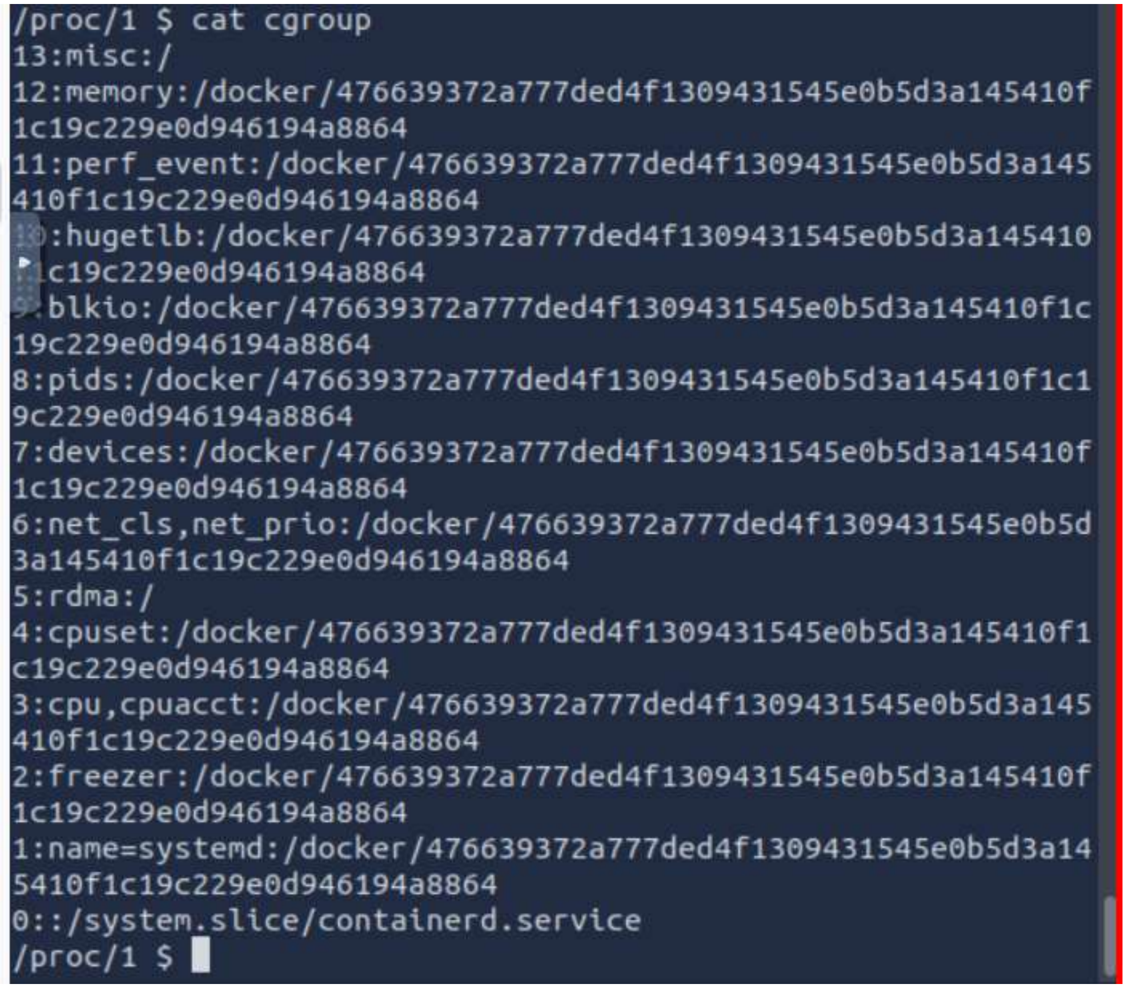
Cgroups are used by containerization software such as LXC or Docker. Let's look for them by navigating to /proc/1 and then catting the "cgroup" file... It is worth mentioning that the "cgroups" file contains paths including the word "docker".

Heading: Utilizing the /proc Filesystem to Understand Container Isolation

"The /proc filesystem in Linux provides a dynamic view of the kernel's data structures, including information about processes. To understand how containerization achieves isolation, we explored the /proc filesystem within our Docker container. Specifically, we examined the /proc/1/ns directory, which lists the namespaces associated with process ID . The screenshot below shows the output of cat /proc/1/, revealing the different types of namespaces and their corresponding inode numbers. This confirms that the web server process operates within a confined set of namespaces, contributing to the overall isolation and security of the containerized environment."



```
/ # cd /proc/1
/proc/1 # ls
arch_status          fd                   numa_maps            smaps_rollup
attr                 fdinfo               oom_adj              stack
autogroup            gid_map              oom_score            stat
auxv                 io                   oom_score_adj        statm
cgroup               limits               pagemap              status
clear_refs           loginuid             patch_state          syscall
cmdline              map_files            personality          task
comm                 maps                 projid_map           timens_offsets
coredump_filter      mem                  root                 timers
cpu_resctrl_groups   mountinfo            sched                timerslack_ns
cpuset               mounts               schedstat            uid_map
cwd                  mountstats           sessionid            wchan
environ              net                  setgroups
exe                  ns                   smaps
/proc/1 # cat cgroup
13:devices:/docker/966650153f23fe24940bfce660c655990c891d89ee0fd36e5897113ddf38c7f0
12:blkio:/docker/966650153f23fe24940bfce660c655990c891d89ee0fd36e5897113ddf38c7f0
11:cpuset:/docker/966650153f23fe24940bfce660c655990c891d89ee0fd36e5897113ddf38c7f0
10:perf_event:/docker/966650153f23fe24940bfce660c655990c891d89ee0fd36e5897113ddf38c
7f0
9:cpu,cpuacct:/docker/966650153f23fe24940bfce660c655990c891d89ee0fd36e5897113ddf38c
7f0
8:freezer:/docker/966650153f23fe24940bfce660c655990c891d89ee0fd36e5897113ddf38c7f0
7:misc:/
6:rdma:/
5:net_cls,net_prio:/docker/966650153f23fe24940bfce660c655990c891d89ee0fd36e5897113d
df38c7f0
4:memory:/docker/966650153f23fe24940bfce660c655990c891d89ee0fd36e5897113ddf38c7f0
3:pids:/docker/966650153f23fe24940bfce660c655990c891d89ee0fd36e5897113ddf38c7f0
2:hugetlb:/docker/966650153f23fe24940bfce660c655990c891d89ee0fd36e5897113ddf38c7f0
1:name=systemd:/docker/966650153f23fe24940bfce660c655990c891d89ee0fd36e5897113ddf38
c7f0
0::/system.slice/containerd.service
/proc/1 # 
```



```
/proc/1 $ cat cgroup
13:misc:/
12:memory:/docker/476639372a777ded4f1309431545e0b5d3a145410f
1c19c229e0d946194a8864
11:perf_event:/docker/476639372a777ded4f1309431545e0b5d3a145
410f1c19c229e0d946194a8864
10:hugetlb:/docker/476639372a777ded4f1309431545e0b5d3a145410
f1c19c229e0d946194a8864
9:blkio:/docker/476639372a777ded4f1309431545e0b5d3a145410f1c
19c229e0d946194a8864
8:pids:/docker/476639372a777ded4f1309431545e0b5d3a145410f1c1
9c229e0d946194a8864
7:devices:/docker/476639372a777ded4f1309431545e0b5d3a145410f
1c19c229e0d946194a8864
6:net_cls,net_prio:/docker/476639372a777ded4f1309431545e0b5d
3a145410f1c19c229e0d946194a8864
5:rdma:/
4:cpuset:/docker/476639372a777ded4f1309431545e0b5d3a145410f1
c19c229e0d946194a8864
3:cpu,cpuacct:/docker/476639372a777ded4f1309431545e0b5d3a145
410f1c19c229e0d946194a8864
2:freezer:/docker/476639372a777ded4f1309431545e0b5d3a145410f
1c19c229e0d946194a8864
1:name=systemd:/docker/476639372a777ded4f1309431545e0b5d3a14
5410f1c19c229e0d946194a8864
0::/system.slice/containerd.service
/proc/1 $
```

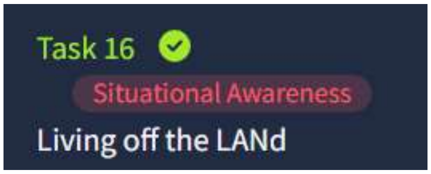
Answer the questions below

Read the above section and familiarize yourself with your new environment

Completed

Submit the flag on L-SRV02

Completed



Task 16

Situational Awareness

Living off the LANd

Introductory Section – Docker Container Network Enumeration & Port Scanning

As the engagement progressed and deeper access was achieved, it became evident that the environment was operating within a **Docker container**. This discovery significantly impacts the attack surface, as containers often reside in isolated network segments with access to internal services not exposed externally. Therefore, the next step in our **situational awareness** phase focused on **enumerating the container's internal network** to uncover services and possible lateral movement paths.

A critical aspect of this phase was identifying the **default gateway** of the container. This gateway typically represents the Docker bridge network interface, often connected to other containers or the host system itself. By inspecting the routing table, the **default gateway** was identified as:

192.168.100.1

With the gateway known, we began enumerating open ports using multiple **"living off the land"** techniques—relying on common tools and scripting languages available within most Linux environments. These included:

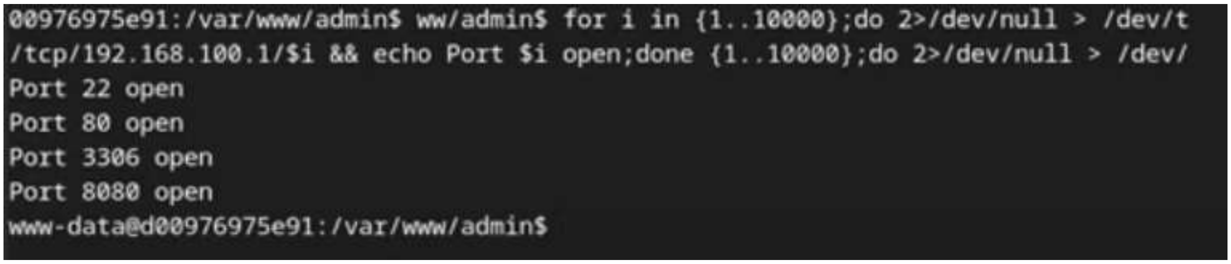
- A **Bash-based port scanner** leveraging the /dev/tcp/ pseudo-device to check port responsiveness.
- A **Python-based port scanner** utilizing the socket library to probe known service ports.
- Use of **Netcat (nc)** in scanning mode to quickly check ranges of ports across the gateway.

Through these methods, we identified two significant open ports on the container's gateway:

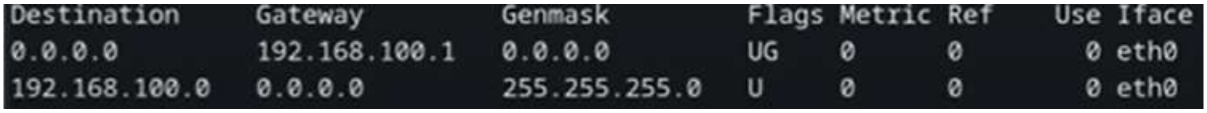
- **Port 8080** – A high-numbered port commonly used for alternative HTTP services or administrative panels.
- **Port 3306** – The default port used by MySQL databases, supporting earlier findings that the system hosts a MySQL backend.

This enumeration phase confirmed the presence of critical services running on the internal network, which may be further leveraged during the exploitation or post-exploitation stages. The results gathered here will inform targeted interactions with backend systems and guide the subsequent phases of the engagement.

Let me know if you want to continue with the next section of the report (e.g., exploitation of the MySQL service on port 3306).



```
00976975e91:/var/www/admin$ ww/admin$ for i in {1..10000};do 2>/dev/null > /dev/t
/tcp/192.168.100.1/$i && echo Port $i open;done {1..10000};do 2>/dev/null > /dev/
Port 22 open
Port 80 open
Port 3306 open
Port 8080 open
www-data@d00976975e91:/var/www/admin$
```



```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.100.1   0.0.0.0         UG    0      0        0 eth0
192.168.100.0   0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

What is the Default Gateway for the Docker Container?

Answer: 192.168.100.1
This is typically obtained using:

What is the high web port open in the container gateway?

Answer: 8080
This is a common alternative HTTP port, often used for web applications.

What is the low database port open in the container gateway?

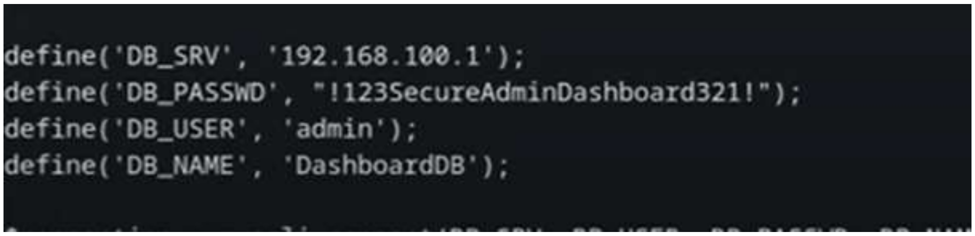
Answer: 3306
Port 3306 is the default port for MySQL, which aligns with the earlier context regarding a MySQL backend.

Task 17 :

proceeded with passive enumeration and information gathering to identify potentially sensitive configuration files and exposed services, particularly those related to backend infrastructure such as databases.

Given that the target system is a confirmed **web server**, it was a logical assumption that it hosts a backend database, likely **MySQL**, to support dynamic content delivery. While databases are often protected from remote access, they tend to be poorly secured when accessed locally, often exposing sensitive configuration files readable by users on the system.

One such file is typically located at the root of the web directory (commonly `/var/www/`) and is named `db_connect.php`. This file is critical for PHP-to-MySQL connectivity and often contains hardcoded database credentials. During our enumeration, we successfully identified and accessed the `db_connect.php` file, which included cleartext connection parameter



```
define('DB_SRV', '192.168.100.1');
define('DB_PASSWD', "!123SecureAdminDashboard321!");
define('DB_USER', 'admin');
define('DB_NAME', 'DashboardDB');
```

This discovery provided valuable information such as:

- **Database host address**
- **Username and password credentials**
- **Target database name**

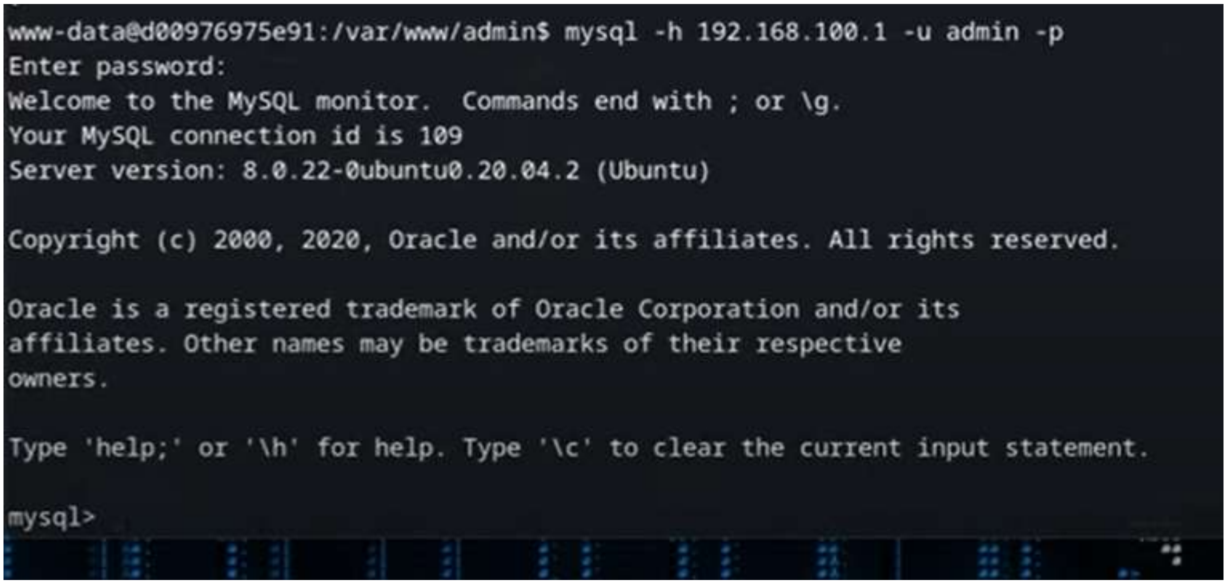
In coordination with the engagement's scope and pre-approved rules of engagement with **Hololive**, we were authorized to attempt access and exfiltration of sensitive data (specifically **names and passwords**) from the DashboardDB database.

We then attempted to access the MySQL service locally using the credentials obtained from the configuration file, leveraging the following syntax:

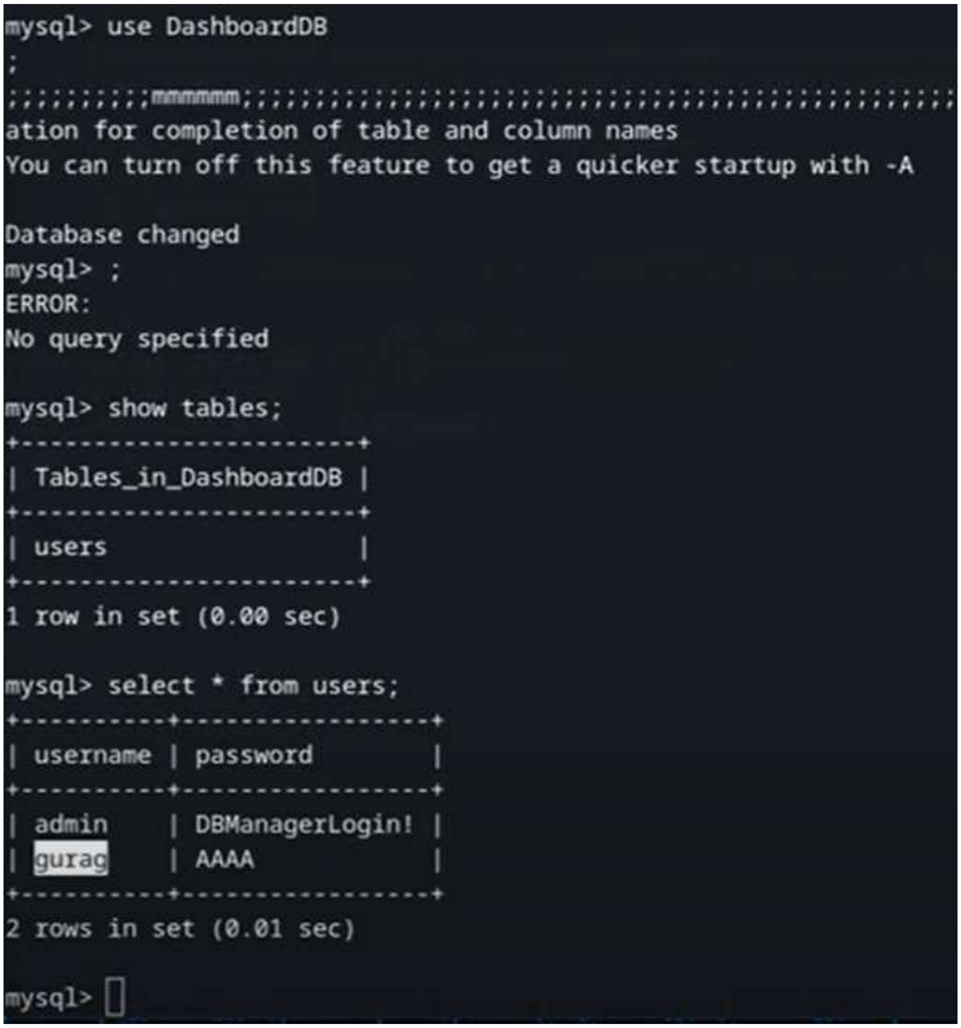
we used basic SQL commands to enumerate the structure and contents of the target database:

- SHOW DATABASES; – to list available databases
- USE DashboardDB; – to select the target database
- SHOW TABLES; – to enumerate tables
- SHOW COLUMNS FROM <table>; – to view table schema
- SELECT * FROM <table>; – to extract data from specific tables

This phase illustrates the importance of internal configuration security and the risk of hardcoded credentials in web-accessible locations. The findings here will be detailed further in the technical findings section, including any credentials or data successfully retrieved.



```
www-data@d00976975e91:/var/www/admin$ mysql -h 192.168.100.1 -u admin -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 109
Server version: 8.0.22-0ubuntu0.20.04.2 (Ubuntu)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```



```
mysql> use DashboardDB
;
;;;;;;;;;;mmmmmm;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ation for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> ;
ERROR:
No query specified

mysql> show tables;
+-----------------------+
| Tables_in_DashboardDB |
+-----------------------+
| users                 |
+-----------------------+
1 row in set (0.00 sec)

mysql> select * from users;
+----------+----------------+
| username | password       |
+----------+----------------+
| admin    | DBManagerLogin! |
| gurag    | AAAA           |
+----------+----------------+
2 rows in set (0.01 sec)

mysql> 
```

Task 39

Situational Awareness: Identifying Defensive Measures on PC-FILESRV01

After gaining a user account on **PC-FILESRV01** and securing a directory we can execute from, the next critical phase is **situational awareness**. This process involves understanding the system we've accessed, its security posture, and what resources or tools are available for further exploitation.

What is Situational Awareness in Windows?

Just like in Linux, situational awareness in Windows aims to gather detailed insights about:

- Active security products (AV/EDR)
- System and user information
- Privilege escalation opportunities
- Possible lateral movement or persistence paths

Tool Spotlight: Seatbelt

One of the primary tools used in this phase is **Seatbelt**, a C#-based utility designed to perform a wide range of host-recon and security checks.

From the GitHub Description:
"Seatbelt is a C# project that performs a number of security-oriented host-survey 'safety checks' relevant from both offensive and defensive security perspectives."

After building Seatbelt using Visual Studio (as described in Task 7), the output will include:

- .exe (Executable)
- .xml
- .pdb
  Only the .exe file is required for execution.

Useful Seatbelt Commands for AV and Security Enumeration

Command	Description
AMSIProviders	Lists registered AMSI providers

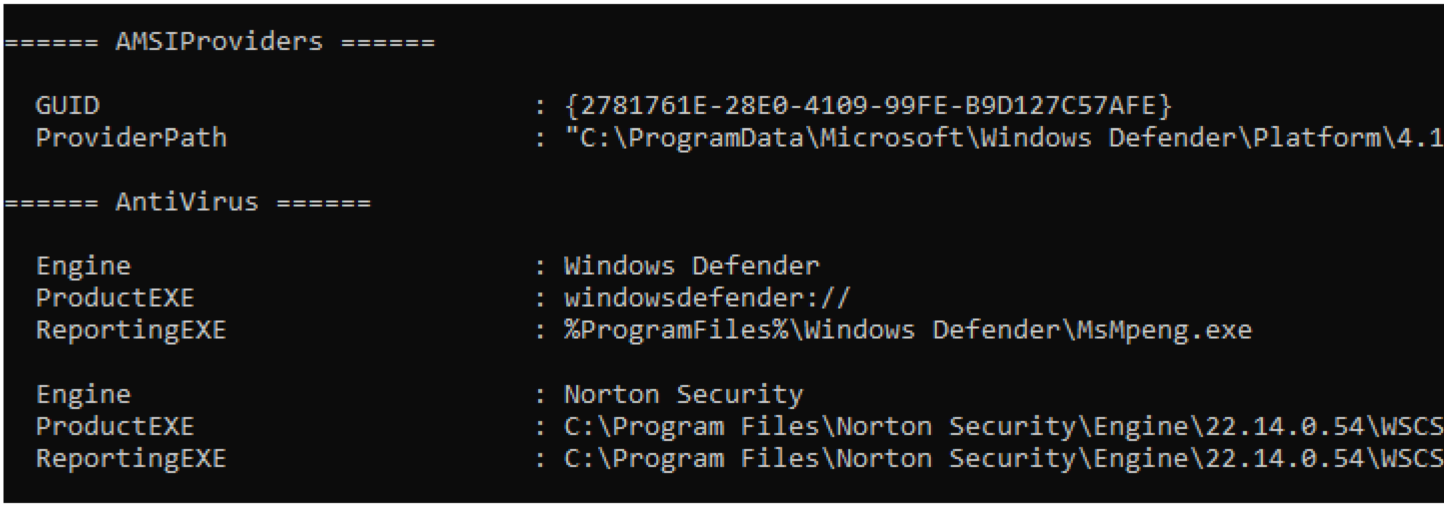
Command	**Description**
AntiVirus	Displays registered antivirus software via WMI
Sysmon	Extracts Sysmon configuration from the registry
WindowsDefender	Shows Windows Defender settings, including exclusions
WindowsEventForwarding	Reveals Windows Event Forwarding (WEF) settings via the registry
McAfeeConfigs	Finds McAfee-related configuration files
InterestingProcesses	Identifies “interesting” processes (e.g., defensive tools, admin utilities)

Syntax: Seatbelt.exe —group=system

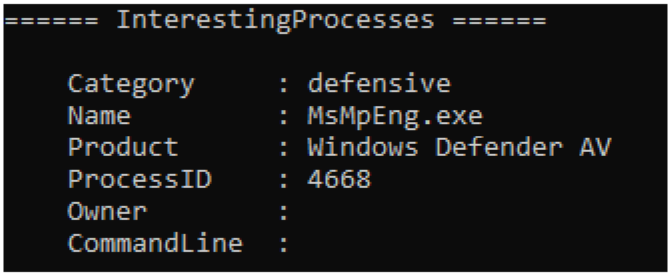
A majority of the commands used above can also be used remotely. This means we will not have to worry about AMSI or Defender as they operate from WMI queries. Find remote syntax for Seatbelt below.

Syntax: Seatbelt.exe -group=remote -computername=<address> -username=<DOMAIN\user> -password=<password

You can find examples of command output indicating AV detections on an endpoint below.



```
====== AMSIProviders ======

  GUID                              : {2781761E-28E0-4109-99FE-B9D127C57AFE}
  ProviderPath                      : "C:\ProgramData\Microsoft\Windows Defender\Platform\4.1

====== AntiVirus ======

  Engine                            : Windows Defender
  ProductEXE                        : windowsdefender://
  ReportingEXE                      : %ProgramFiles%\Windows Defender\MsMpeng.exe

  Engine                            : Norton Security
  ProductEXE                        : C:\Program Files\Norton Security\Engine\22.14.0.54\WSCS
  ReportingEXE                      : C:\Program Files\Norton Security\Engine\22.14.0.54\WSCS
```



```
====== InterestingProcesses ======

    Category     : defensive
    Name         : MsMpEng.exe
    Product      : Windows Defender AV
    ProcessID    : 4668
    Owner        :
    CommandLine  :
```

The second tool we will be looking at is

From the SharpEDRChecker GitHub "SharpEDRChecker, checks running processes, process metadata, DLLs loaded into your current process and the each DLLs metadata, common install directories, installed services and each service binaries metadata, installed drivers and each drivers metadata, all for the presence of known defensive products such as AV's, EDR's and logging tools."

This means that we can identify more advanced forms of anti-virus and detection agents that Seatbelt or other tools may not be able to locate using their methods. For example, Carbon Black, Tanium, or Crowd Strike; these solution platforms can deploy agents onto an endpoint custom to the organization similar to a malicious payload (basically malware, right?)

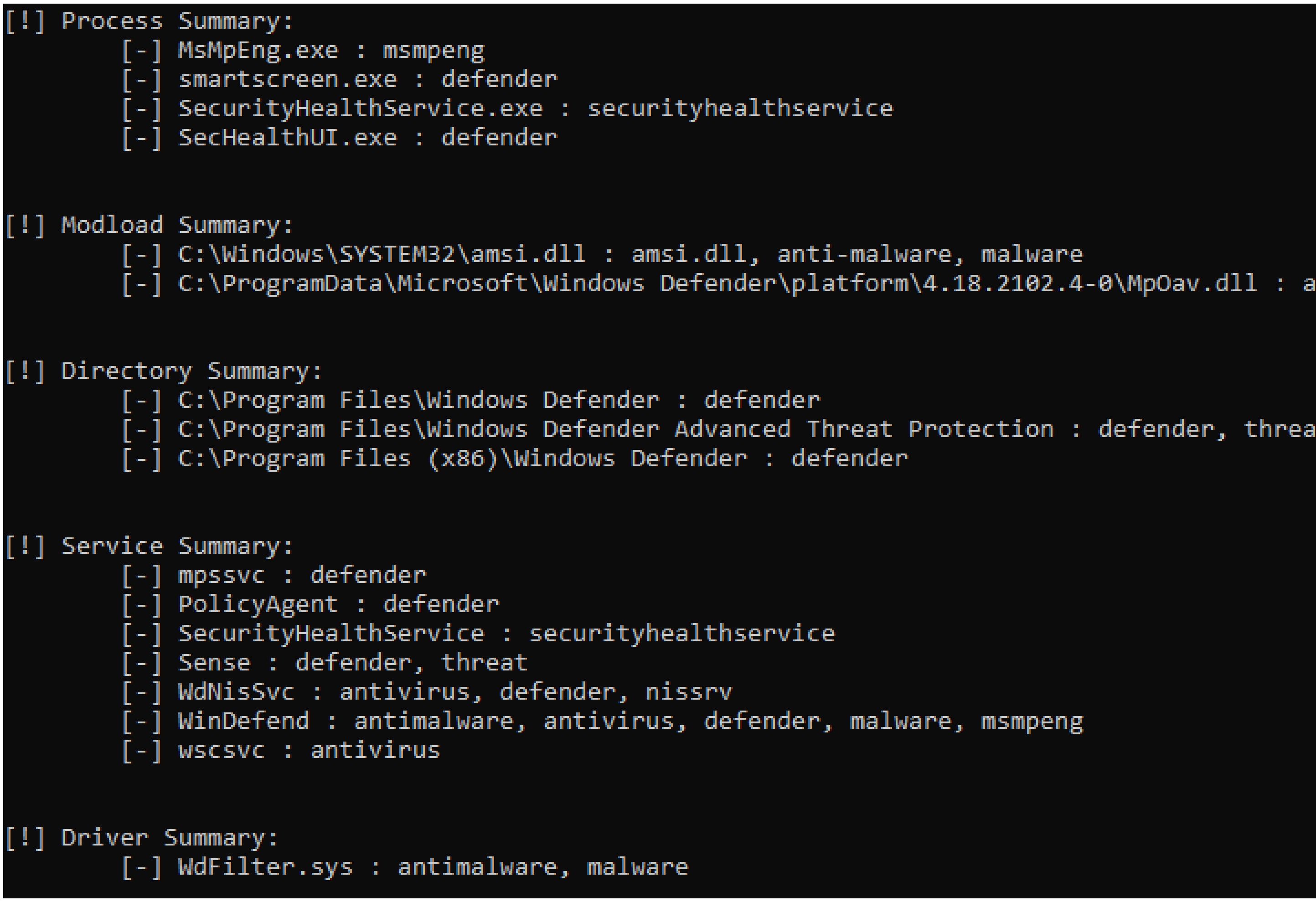
Below we will go into each of the functions of SharpEDRChecker and how they can benefit us in situational awareness.

- FileChecker This function of the tool is what really separates it from other tools. It will check the metadata of the file that cannot be changed as it will invalidate code signing and break other aspects of the file.
- ProcessChecker Similar function to Seatbelt's InterestingProcesses, The first part of this module will inspect all processes. The second part of the module will check for DLLs loaded by processes, this is important for identifying products such as Cylance and AMSI.
- ServiceChecker Inspects installed services, a similar function to ProcessChecker.
- DriverChecker Performs checks on all drivers using P/Invoke.
- DirectoryChecker Dumps all interesting subdirectories on common directories (Program Files, ProgramData, etc.)

To begin using SharpEDRChecker, you can either download a pre-compiled release from GitHub or compile from source using the solution file. For more information about compiling, return to Task 7. Find releases here, https://github.com/PwnDexter/SharpEDRChecker/releases/tag/1.1.

Find syntax and example output from SharpEDRChecker below.

Syntax: .\SharpEDRChecker.exe



```
[!] Process Summary:
        [-] MsMpEng.exe : msmpeng
        [-] smartscreen.exe : defender
        [-] SecurityHealthService.exe : securityhealthservice
        [-] SecHealthUI.exe : defender


[!] Modload Summary:
        [-] C:\Windows\SYSTEM32\amsi.dll : amsi.dll, anti-malware, malware
        [-] C:\ProgramData\Microsoft\Windows Defender\platform\4.18.2102.4-0\MpOav.dll : a


[!] Directory Summary:
        [-] C:\Program Files\Windows Defender : defender
        [-] C:\Program Files\Windows Defender Advanced Threat Protection : defender, threa
        [-] C:\Program Files (x86)\Windows Defender : defender


[!] Service Summary:
        [-] mpssvc : defender
        [-] PolicyAgent : defender
        [-] SecurityHealthService : securityhealthservice
        [-] Sense : defender, threat
        [-] WdNisSvc : antivirus, defender, nissrv
        [-] WinDefend : antimalware, antivirus, defender, malware, msmpeng
        [-] wscsvc : antivirus


[!] Driver Summary:
        [-] WdFilter.sys : antimalware, malware
```

From the above screenshot, we can see that this tool gives us a much more detailed output than Seatbelt, which is a lot more focused and offers more insight than Seatbelt and other tools.

For more information about SharpEDRChecker,.

Task 40

System Enumeration Using Seatbelt

After understanding the system's detection mechanisms and clarifying what actions are possible within our attack surface, the next step is to perform system enumeration. Enumeration helps us gather detailed information about the endpoint, allowing us to map out the attack surface and identify potential paths for privilege escalation.

For this purpose, we will use **Seatbelt**, a powerful enumeration tool designed to perform multiple system checks and gather valuable data about the endpoint.

What is Seatbelt?

Seatbelt is a comprehensive enumeration tool that runs a variety of checks to collect system information quickly and efficiently. It helps uncover configuration details, user privileges, running processes, and much more. In this phase, we will be leveraging *all* the modules Seatbelt provides to get a full picture of the system's state.

Key Modules We Will Use

Here's a brief overview of some important Seatbelt modules and what they do:

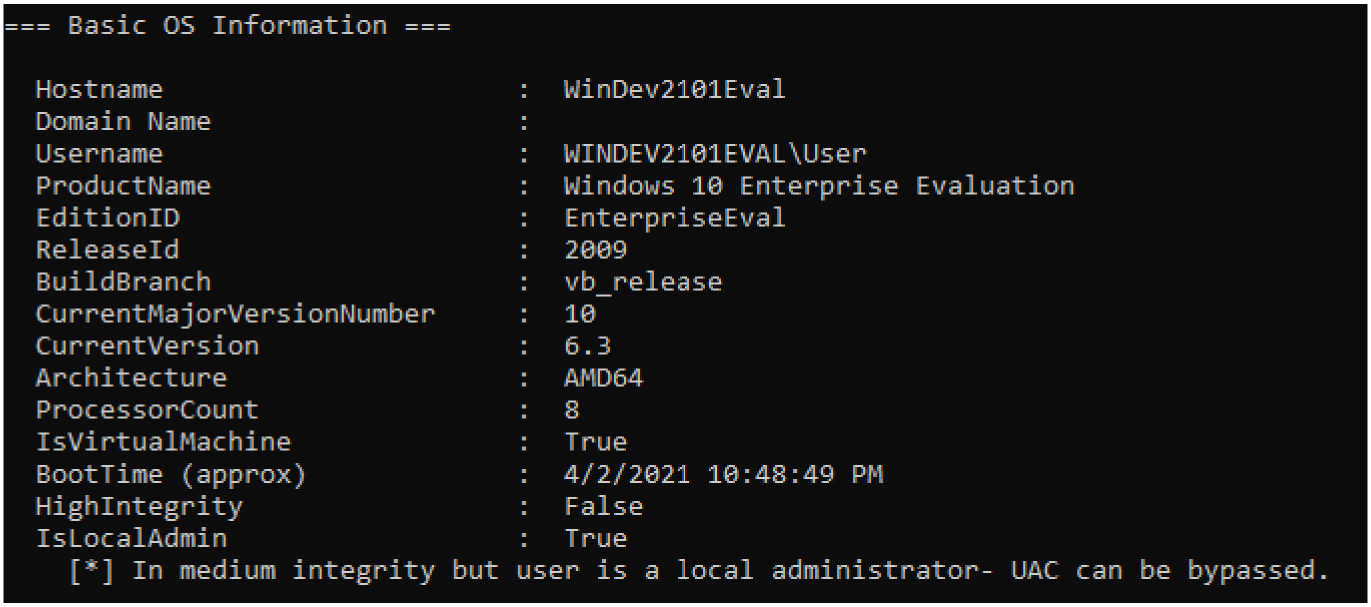
- **DotNet**: Retrieves the installed .NET framework version.
- **LocalGPOs**: Finds any local group policies applied on the machine and to local users.
- **LocalGroups**: Lists all non-empty local user groups.
- **NetworkShares**: Identifies any exposed network shares.
- **PowerShell**: Checks the PowerShell version and security-related settings.
- **Processes**: Lists all running processes on the endpoint.
- **TokenPrivileges**: Shows enabled token privileges such as `SeDebugPrivilege`.
- **CredEnum**: Enumerates saved credentials for the current user.
- **InterestingFiles**: Finds files matching suspicious or interesting patterns in the user's folder.
- **ScheduledTasks**: Lists scheduled tasks that were *not* created by Microsoft, potentially indicating custom or malicious tasks.

Note: Some of these modules require elevated privileges or a desktop session to execute fully. However, even with low privileges, Seatbelt can provide valuable insights about the system.

How to Run Seatbelt

To run Seatbelt and collect all available data, the command syntax is:

Syntax: .\Seatbelt.exe all



```
=== Basic OS Information ===

  Hostname                      :  WinDev2101Eval
  Domain Name                   :
  Username                      :  WINDEV2101EVAL\User
  ProductName                   :  Windows 10 Enterprise Evaluation
  EditionID                     :  EnterpriseEval
  ReleaseId                     :  2009
  BuildBranch                   :  vb_release
  CurrentMajorVersionNumber     :  10
  CurrentVersion                :  6.3
  Architecture                  :  AMD64
  ProcessorCount                :  8
  IsVirtualMachine              :  True
  BootTime (approx)             :  4/2/2021 10:48:49 PM
  HighIntegrity                 :  False
  IsLocalAdmin                  :  True
    [*] In medium integrity but user is a local administrator- UAC can be bypassed.
```

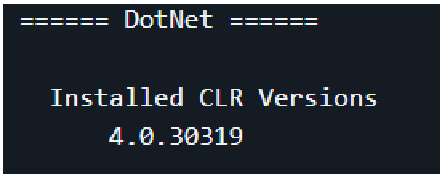
We can also run Seatbelt from Covenant using the Seatbelt module found below.

Module: Seatbelt

You will notice that Seatbelt produces a large amount of output. It can be helpful to save this output to a file to comb through later. You will have to spend a little bit of time searching through the output to get all the information you need on the endpoint.

What CLR version is installed on PC-FILESRV01?

PowerShell module

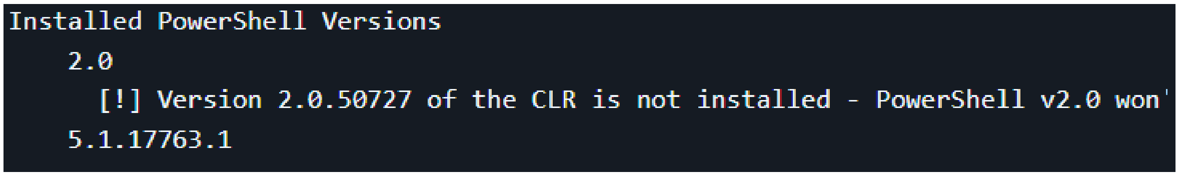


```
 ====== DotNet ======


   Installed CLR Versions
       4.0.30319
```

What PowerShell version is installed on PC-FILESRV01?

PowerShell module

5.1.17763.1

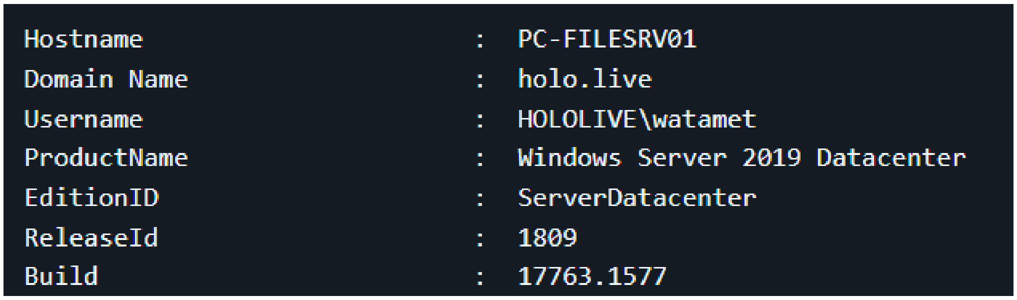


```
Installed PowerShell Versions
    2.0
      [!] Version 2.0.50727 of the CLR is not installed - PowerShell v2.0 won'
    5.1.17763.1
```

What Windows build is PC-FILESRV01 running on?

OSInfo module

17763.1577



```
Hostname                      :  PC-FILESRV01
Domain Name                   :  holo.live
Username                      :  HOLOLIVE\watamet
ProductName                   :  Windows Server 2019 Datacenter
EditionID                     :  ServerDatacenter
ReleaseId                     :  1809
Build                         :  17763.1577
```

Task 41

Situational Awareness ALL THE POWER!

Now that we understand detections and system surface on the endpoint, we can begin looking at the user and groups of the system. This step of situational awareness can allow us to find privileges and user connections for future horizontal movement or privilege escalation.

The first tool we will be looking at is

This tool is no longer supported but is still considered a standard for enumeration. From the PowerSploit GitHub, "PowerView is a PowerShell tool to gain network situational awareness on Windows domains. It contains a set of pure-PowerShell replacements for various windows "net *" commands, which utilize PowerShell AD hooks and underlying Win32 API functions to perform useful Windows domain functionality."

To use the script, we will first need to import it then run the commands that we want to enumerate the endpoint. Find syntax and a few essential commands you can use with PowerView.

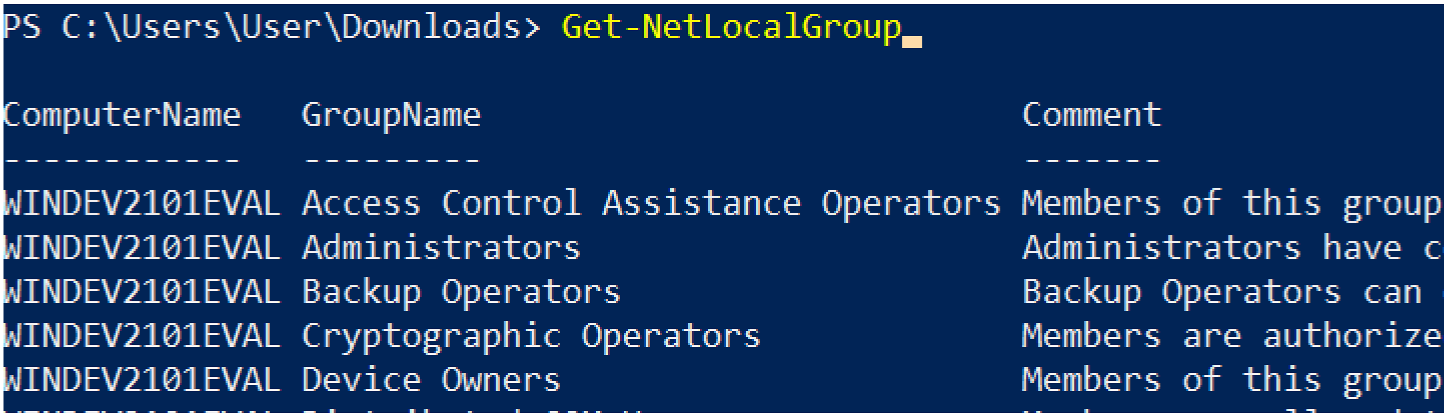
Syntax: Import-Module .\PowerView.ps1

We can now run all of the commands that PowerView offers. In this task, we will be focusing on enumerating the local user and group policy surface. In the next task, we will use native PowerShell to enumerate the active directory surface. Outlined below is a list of commands we will cover in this task.

- Get-NetLocalGroup
- Get-NetLocalGroupMember
- Get-NetLoggedon
- Get-DomainGPO
- Find-LocalAdminAccess

The first PowerView command we will be looking at is Get-NetLocalGroup; this command will enumerate/list all groups present on a local machine/computer. Find the syntax and output for the command below.

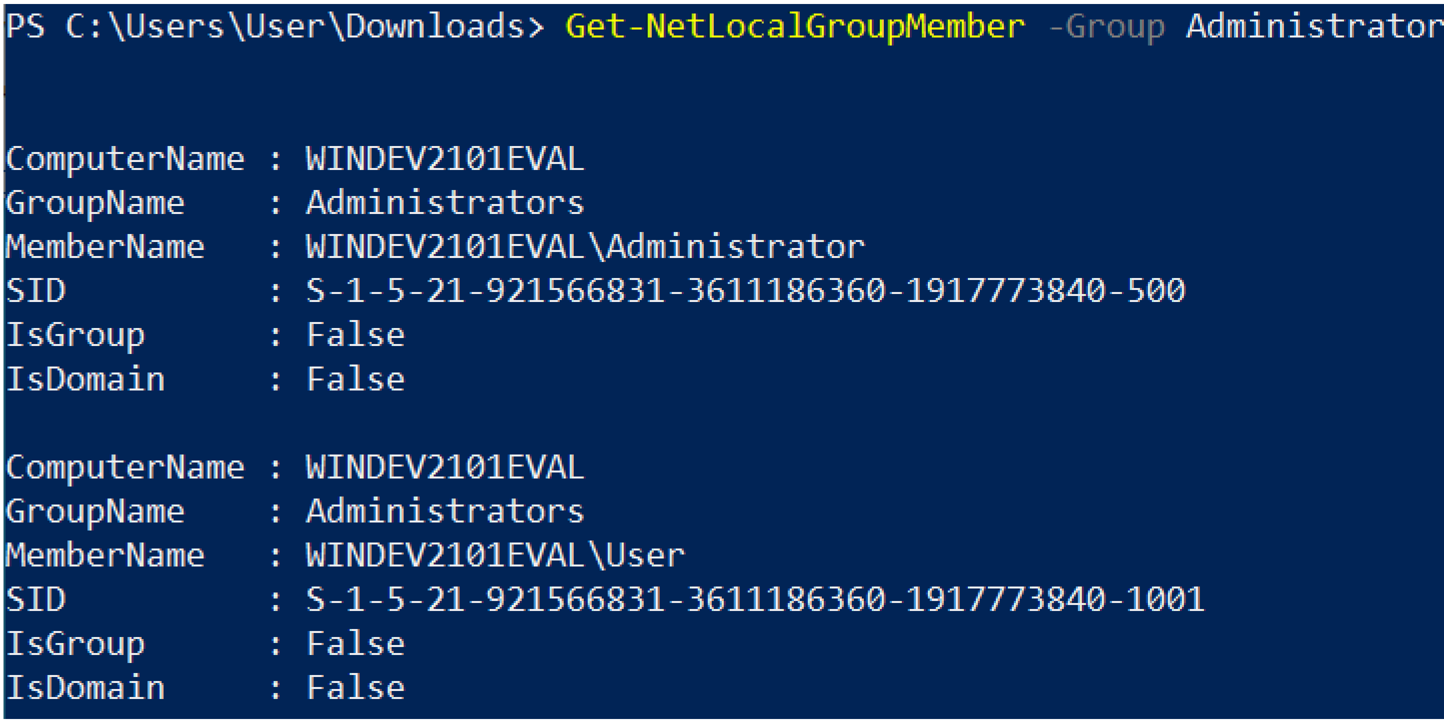
Syntax: Get-NetLocalGroup



```
PS C:\Users\User\Downloads> Get-NetLocalGroup

ComputerName   GroupName                          Comment
------------   ---------                          -------
WINDEV2101EVAL Access Control Assistance Operators Members of this group
WINDEV2101EVAL Administrators                     Administrators have c
WINDEV2101EVAL Backup Operators                   Backup Operators can
WINDEV2101EVAL Cryptographic Operators            Members are authorize
WINDEV2101EVAL Device Owners                      Members of this group
```

The second PowerView command we will be looking at is Get-NetLocalGroupMember; this command will enumerate/list all members of a local group such as users, computers, or service accounts. Find the syntax and output for the command below.

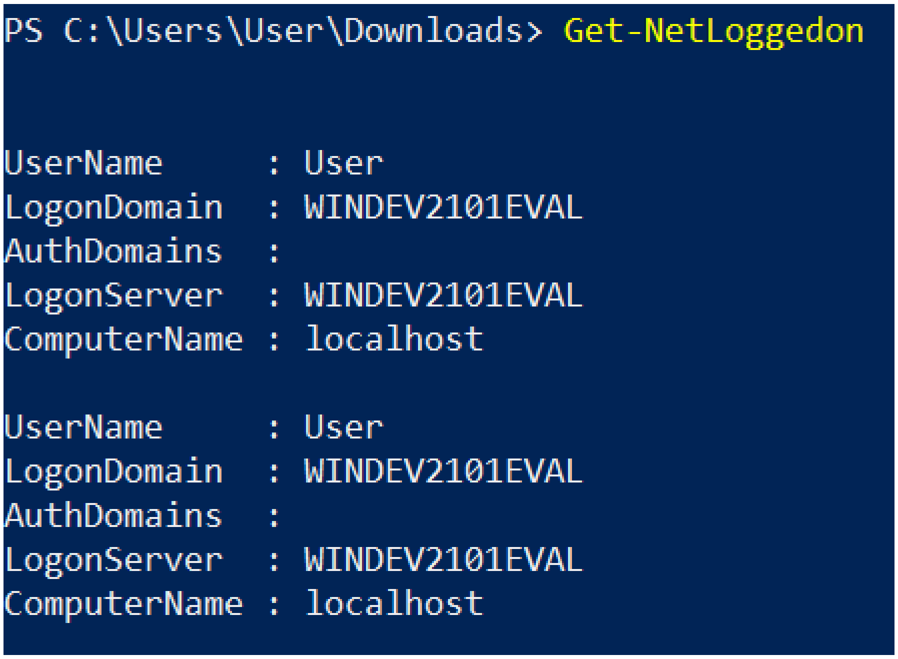
Syntax: Get-NetLocalGroupMember -Group <group>



```
PS C:\Users\User\Downloads> Get-NetLocalGroupMember -Group Administrator


ComputerName : WINDEV2101EVAL
GroupName    : Administrators
MemberName   : WINDEV2101EVAL\Administrator
SID          : S-1-5-21-921566831-3611186360-1917773840-500
IsGroup      : False
IsDomain     : False

ComputerName : WINDEV2101EVAL
GroupName    : Administrators
MemberName   : WINDEV2101EVAL\User
SID          : S-1-5-21-921566831-3611186360-1917773840-1001
IsGroup      : False
IsDomain     : False
```

The third PowerView command we will be looking at is Get-NetLoggedon; this command will enumerate/list all users currently logged onto the local machine/computer. This can be useful to identify what user's not to take over or what users to target in phishing or other attacks depending on your team's methodology and/or goals. Find the syntax and output for the command below.

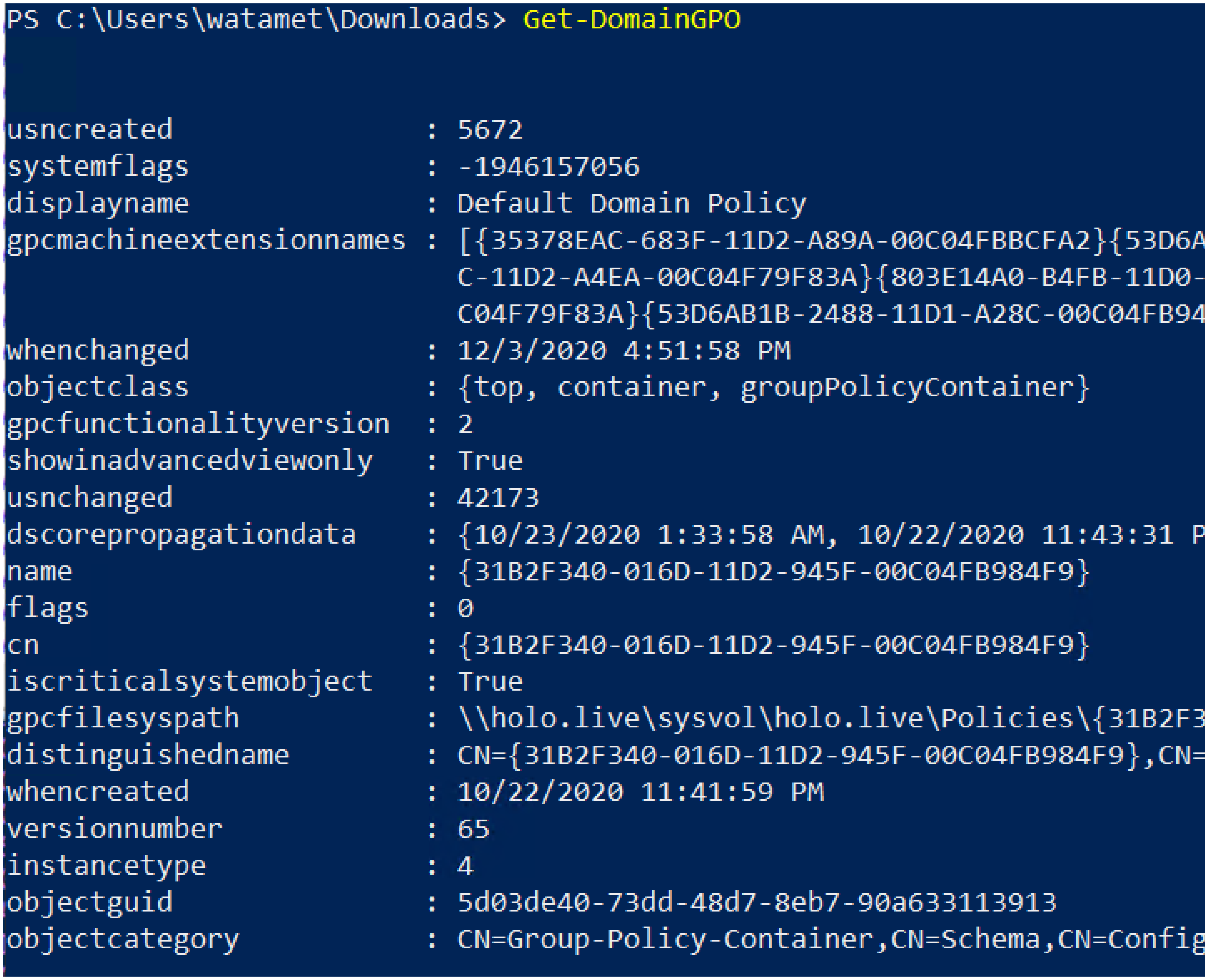
Syntax: Get-NetLoggedon



```
PS C:\Users\User\Downloads> Get-NetLoggedon


UserName     : User
LogonDomain  : WINDEV2101EVAL
AuthDomains  :
LogonServer  : WINDEV2101EVAL
ComputerName : localhost

UserName     : User
LogonDomain  : WINDEV2101EVAL
AuthDomains  :
LogonServer  : WINDEV2101EVAL
ComputerName : localhost
```

The fourth PowerView command we will be looking at is Get-DomainGPO; this command will enumerate/list the active directory domain GPOs installed on the local machine. This can be useful in identifying utilities like AppLocker or other remote services running on the machine/computer. Find the syntax and output for the command below.

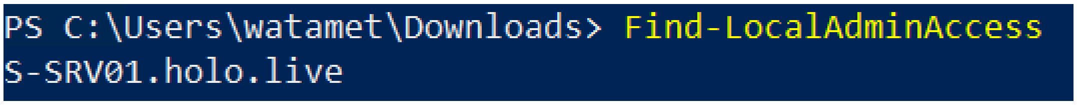
Syntax: Get-DomainGPO



```
PS C:\Users\watamet\Downloads> Get-DomainGPO


usncreated               : 5672
systemflags              : -1946157056
displayname              : Default Domain Policy
gpcmachineextensionnames : [{35378EAC-683F-11D2-A89A-00C04FBBCFA2}{53D6A
                           C-11D2-A4EA-00C04F79F83A}{803E14A0-B4FB-11D0-
                           C04F79F83A}{53D6AB1B-2488-11D1-A28C-00C04FB94
whenchanged              : 12/3/2020 4:51:58 PM
objectclass              : {top, container, groupPolicyContainer}
gpcfunctionalityversion  : 2
showinadvancedviewonly   : True
usnchanged               : 42173
dscorepropagationdata    : {10/23/2020 1:33:58 AM, 10/22/2020 11:43:31 P
name                     : {31B2F340-016D-11D2-945F-00C04FB984F9}
flags                    : 0
cn                       : {31B2F340-016D-11D2-945F-00C04FB984F9}
iscriticalsystemobject   : True
gpcfilesyspath           : \\holo.live\sysvol\holo.live\Policies\{31B2F3
distinguishedname        : CN={31B2F340-016D-11D2-945F-00C04FB984F9},CN=
whencreated              : 10/22/2020 11:41:59 PM
versionnumber            : 65
instancetype             : 4
objectguid               : 5d03de40-73dd-48d7-8eb7-90a633113913
objectcategory           : CN=Group-Policy-Container,CN=Schema,CN=Config
```

The final PowerView command we will be looking at is Find-LocalAdminAccess; this command will check all hosts connected to the domain a machine/computer is a part of and check if the current user or listed user is a local administrator. This can be helpful when targeting a specific user and attempting to move across the domain laterally. This can be used as an alternative to other tools like CME for passing the hash. Find the syntax and output for the command below.

Syntax: Find-LocalAdminAccess



```
PS C:\Users\watamet\Downloads> Find-LocalAdminAccess
S-SRV01.holo.live
```

As with most offensive tooling, Defender detects this script. You will need to follow the methodology given in Task 31-36 to execute this tool and evade detections.

Answer the questions below

Read the above and enumerate PC-FILESRV01 using PowerView.

Task 42

Situational Awareness Using PowerShell

In some environments, endpoint detection and response (EDR) systems may block or restrict the execution of tools like Seatbelt and PowerView. When that happens, we can fall back on native PowerShell commands to perform situational awareness and system/environment enumeration without relying on third-party tools.

PowerShell is a powerful scripting language that comes built into Windows. It includes a wide range of native commands and modules that can give us deep visibility into the system — especially within Active Directory environments.

Scope of This Task

In this section, we’re only scratching the surface of what PowerShell can do. Below is a list of some important PowerShell commands and modules we'll cover:

Get-ScheduledTask

Get-ScheduledTaskInfo

whoami /priv

Get-ADGroup

Get-ADGroupMember

Get-ADPrincipalGroupMembership

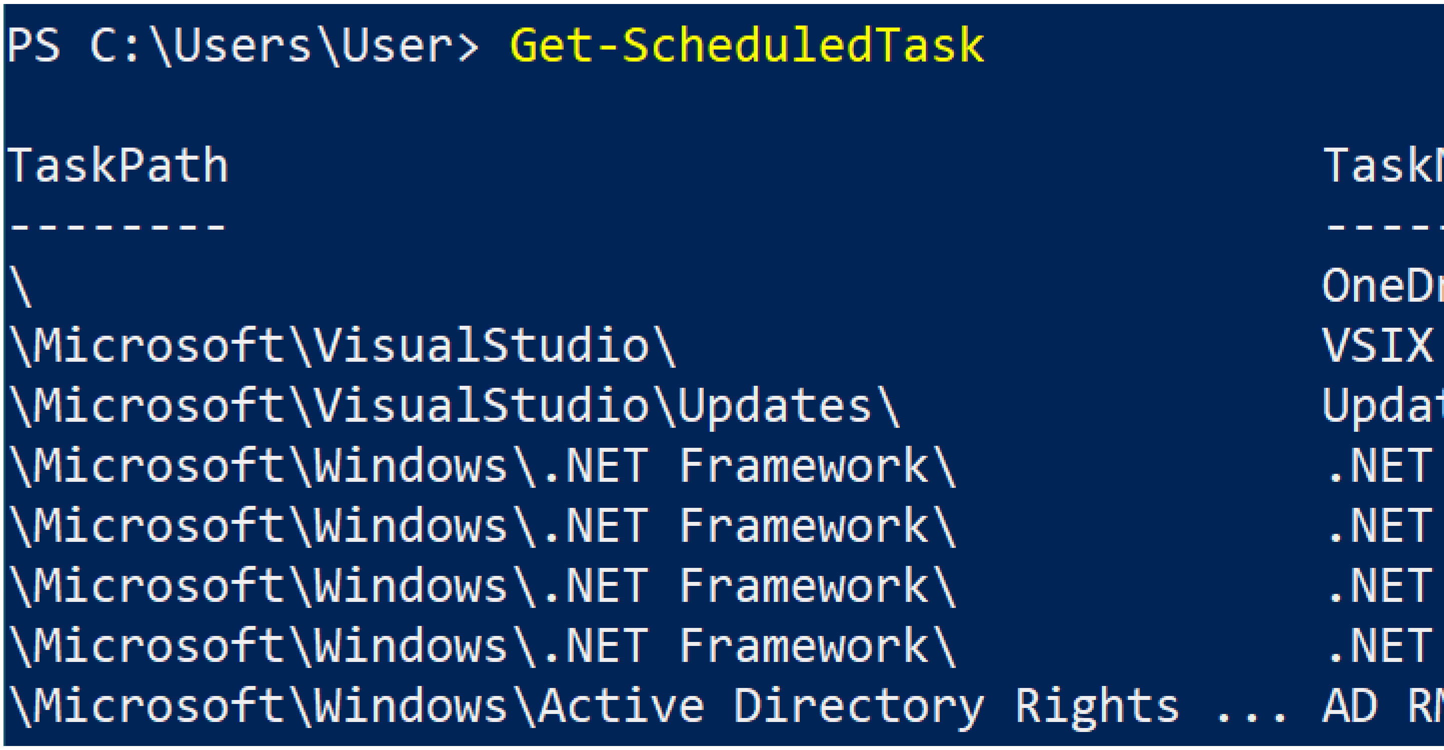
You’ll notice that many of these commands focus on Active Directory. This is intentional — PowerShell’s AD-related modules are designed to give administrators full visibility and control over users, groups, and permissions. We can take advantage of these same capabilities to spot potential misconfigurations or privilege escalation paths.

Step 1: Scheduled Tasks Enumeration

Let’s begin with commands that can reveal scheduled task misconfigurations, a common privilege escalation vector.

```
 Get-ScheduledTask
```

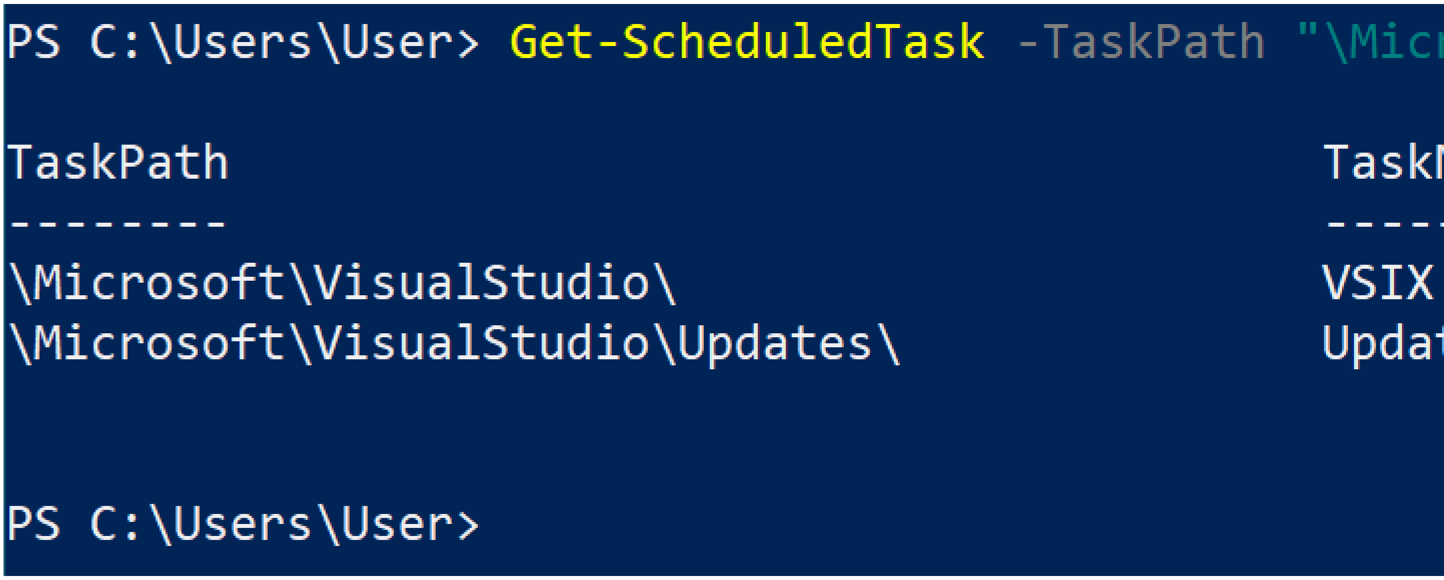
This command lists **all scheduled tasks** on the local machine. Scheduled tasks are often used by system admins for automation, but attackers may abuse them to maintain persistence or escalate privileges.



```
PS C:\Users\User> Get-ScheduledTask

TaskPath                                       TaskN
--------                                       -----
\                                              OneDr
\Microsoft\VisualStudio\                       VSIX
\Microsoft\VisualStudio\Updates\               Updat
\Microsoft\Windows\.NET Framework\             .NET
\Microsoft\Windows\.NET Framework\             .NET
\Microsoft\Windows\.NET Framework\             .NET
\Microsoft\Windows\.NET Framework\             .NET
\Microsoft\Windows\Active Directory Rights ... AD RM
```

You will notice that there is a large number of tasks present; this is because Windows operates at startup with a large number of tasks default on every Windows install. We can use filters and parameters to eliminate some of the unneeded tasks to focus on obscure tasks that we can abuse. Find syntax for filtering below.

Syntax: Get-ScheduledTask -TaskPath "\Users*"



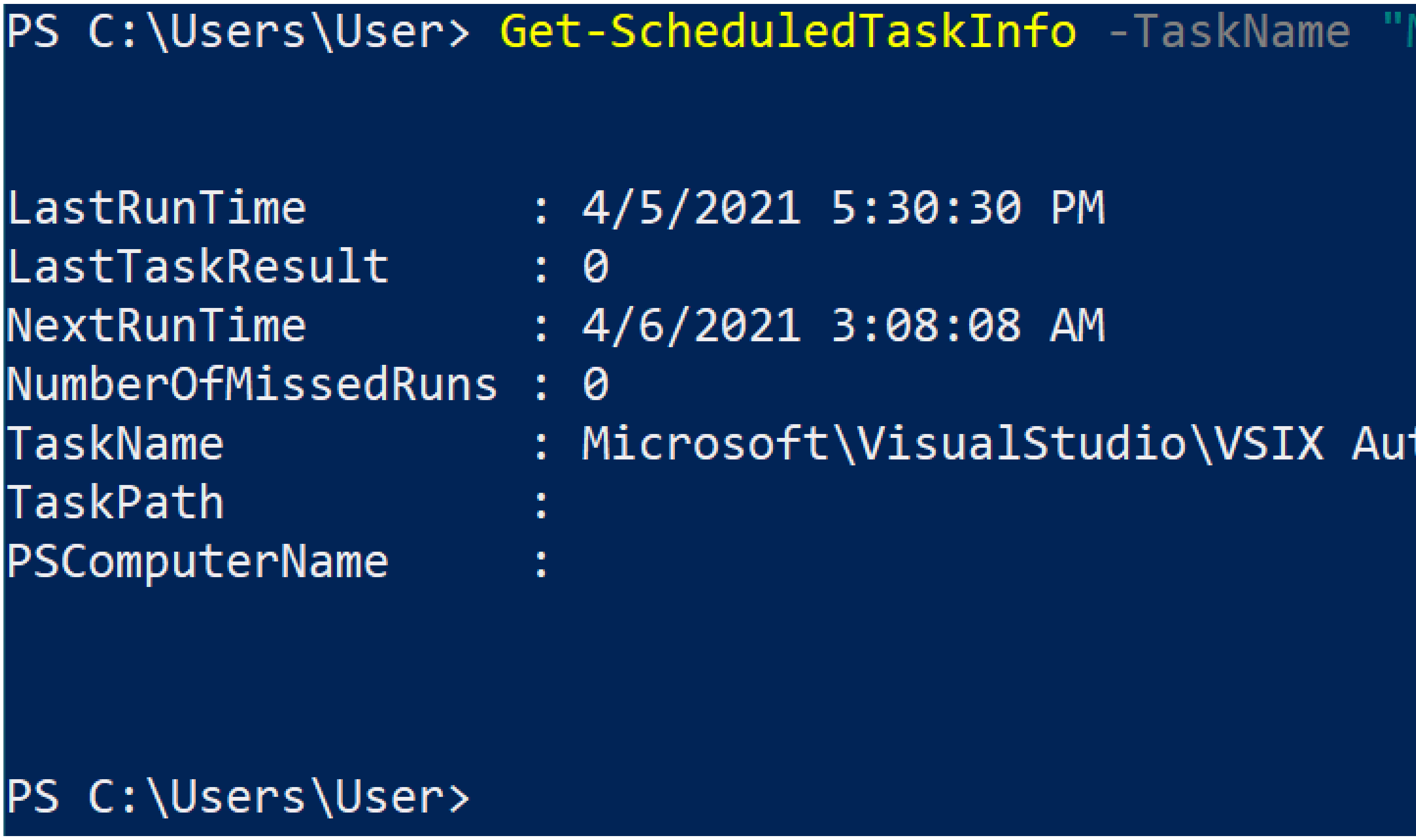
```
PS C:\Users\User> Get-ScheduledTask -TaskPath "\Micr

TaskPath                                       TaskN
--------                                       -----
\Microsoft\VisualStudio\                       VSIX
\Microsoft\VisualStudio\Updates\               Updat


PS C:\Users\User>
```

You can experiment with parameters and inputs to get the most optimal output for system enumeration.

For more information about Get-ScheduledTask,

The second PowerShell command we will be looking at is Get-ScheduledTaskInfo; similar to Get-ScheduledTask, this command will list specific information on specified Tasks allowing the attacker to identify the task and how it could be exploited. Find syntax for the command below.

Syntax: Get-ScheduledTaskInfo -TaskName <Full Path>



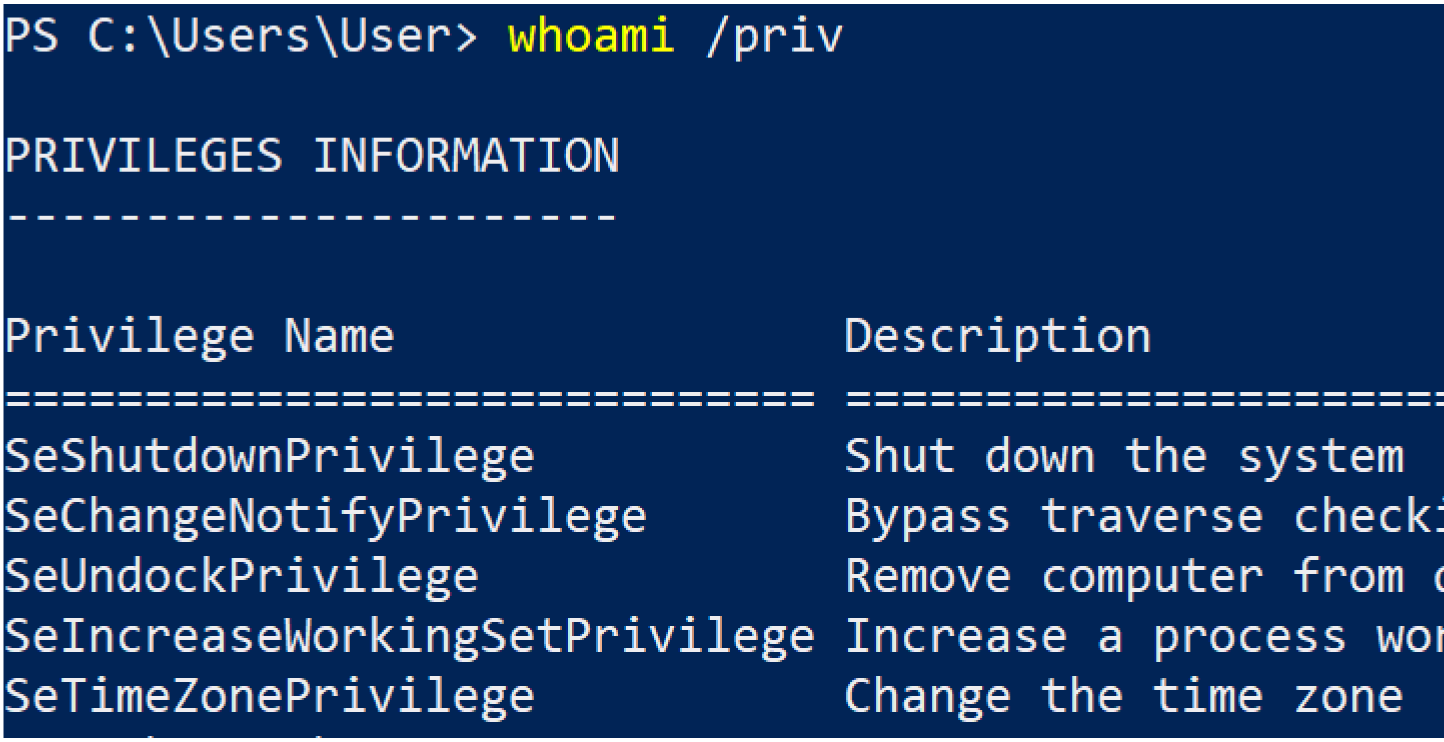
```
PS C:\Users\User> Get-ScheduledTaskInfo -TaskName "M


LastRunTime        : 4/5/2021 5:30:30 PM
LastTaskResult     : 0
NextRunTime        : 4/6/2021 3:08:08 AM
NumberOfMissedRuns : 0
TaskName           : Microsoft\VisualStudio\VSIX Aut
TaskPath           :
PSComputerName     :



PS C:\Users\User>
```

For more information about Get-ScheduledTaskInfo, check out the Microsoft docs, Step 2: Privilege Awareness

whoami /priv

This command is actually a standard Windows command, not PowerShell-specific, but it’s useful when run in a PowerShell terminal.

It lists all privileges currently assigned to your user token — for example, SeDebugPrivilege, SeImpersonatePrivilege, etc. These can help determine whether you already have rights that could be used in an attack.



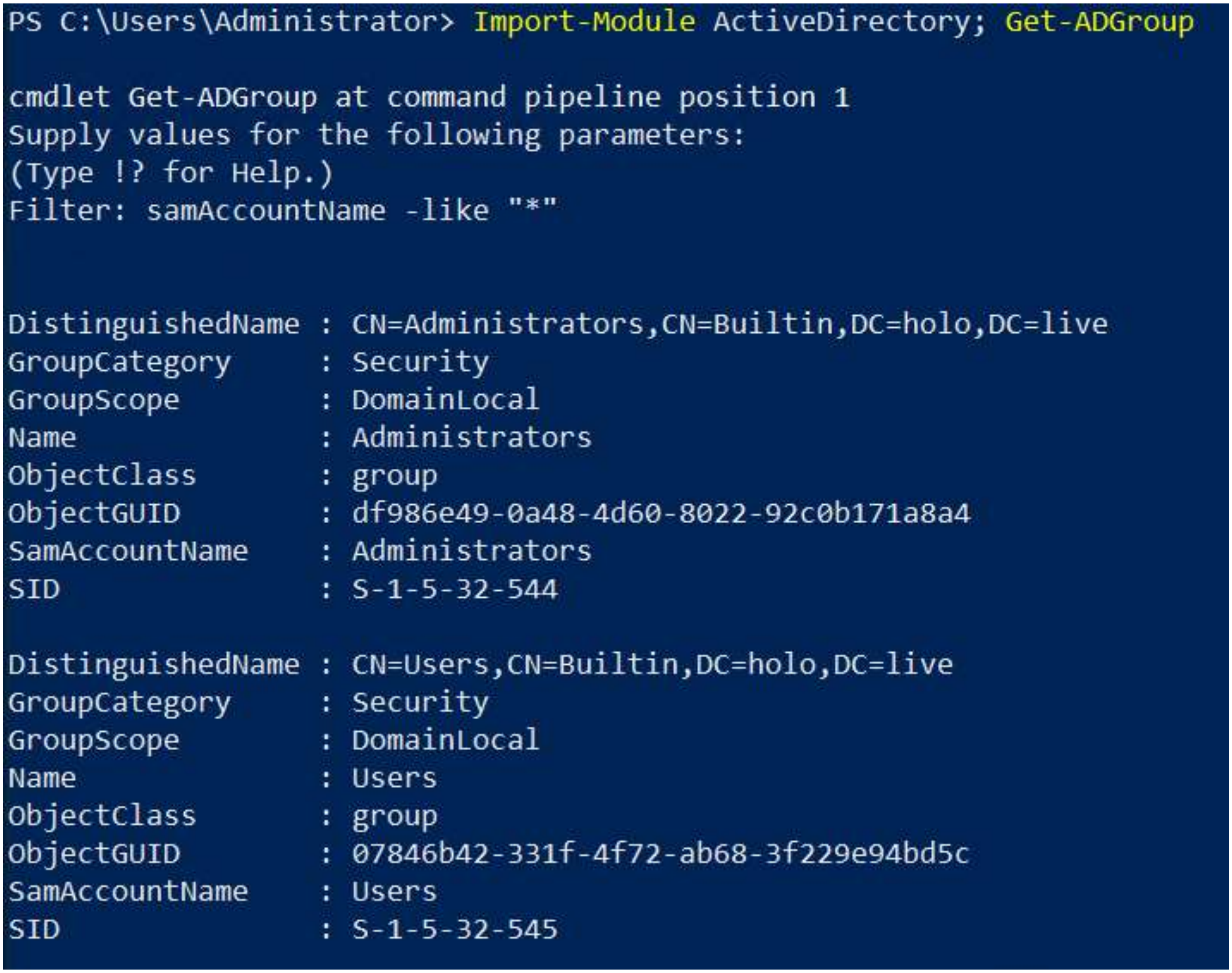
```
PS C:\Users\User> whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description
============================= =======================
SeShutdownPrivilege           Shut down the system
SeChangeNotifyPrivilege       Bypass traverse checki
SeUndockPrivilege             Remove computer from d
SeIncreaseWorkingSetPrivilege Increase a process wor
SeTimeZonePrivilege           Change the time zone
```

The fourth PowerShell command we will be looking at is Get-ADGroup; this module, part of the active directory module package, will allow us to enumerate a user's groups or all groups within the domain. To get the most out of this command, we will already need to enumerate the users present on the machine. Since this command is part of the ActiveDirectory module, you will need first to import the module. Find the syntax for the command below.

Syntax: Import-Module ActiveDirectory; Get-ADGroup

After running the command, you will be prompted with a CLI to apply filters to the command; we recommend filtering by the samAccountName. Find example usage for this filter below.

Syntax: samAccountName -like "*"



```
PS C:\Users\Administrator> Import-Module ActiveDirectory; Get-ADGroup

cmdlet Get-ADGroup at command pipeline position 1
Supply values for the following parameters:
(Type !? for Help.)
Filter: samAccountName -like "*"


DistinguishedName : CN=Administrators,CN=Builtin,DC=holo,DC=live
GroupCategory     : Security
GroupScope        : DomainLocal
Name              : Administrators
ObjectClass       : group
ObjectGUID        : df986e49-0a48-4d60-8022-92c0b171a8a4
SamAccountName    : Administrators
SID               : S-1-5-32-544

DistinguishedName : CN=Users,CN=Builtin,DC=holo,DC=live
GroupCategory     : Security
GroupScope        : DomainLocal
Name              : Users
ObjectClass       : group
ObjectGUID        : 07846b42-331f-4f72-ab68-3f229e94bd5c
SamAccountName    : Users
SID               : S-1-5-32-545
```

To get the most out of this command, you will need to play with the filters and parameters used to get the most efficient output to enumerate the critical information.

For more information about Get-ADGroup, check out the Microsoft

For more information about Get-ADGroupMember,

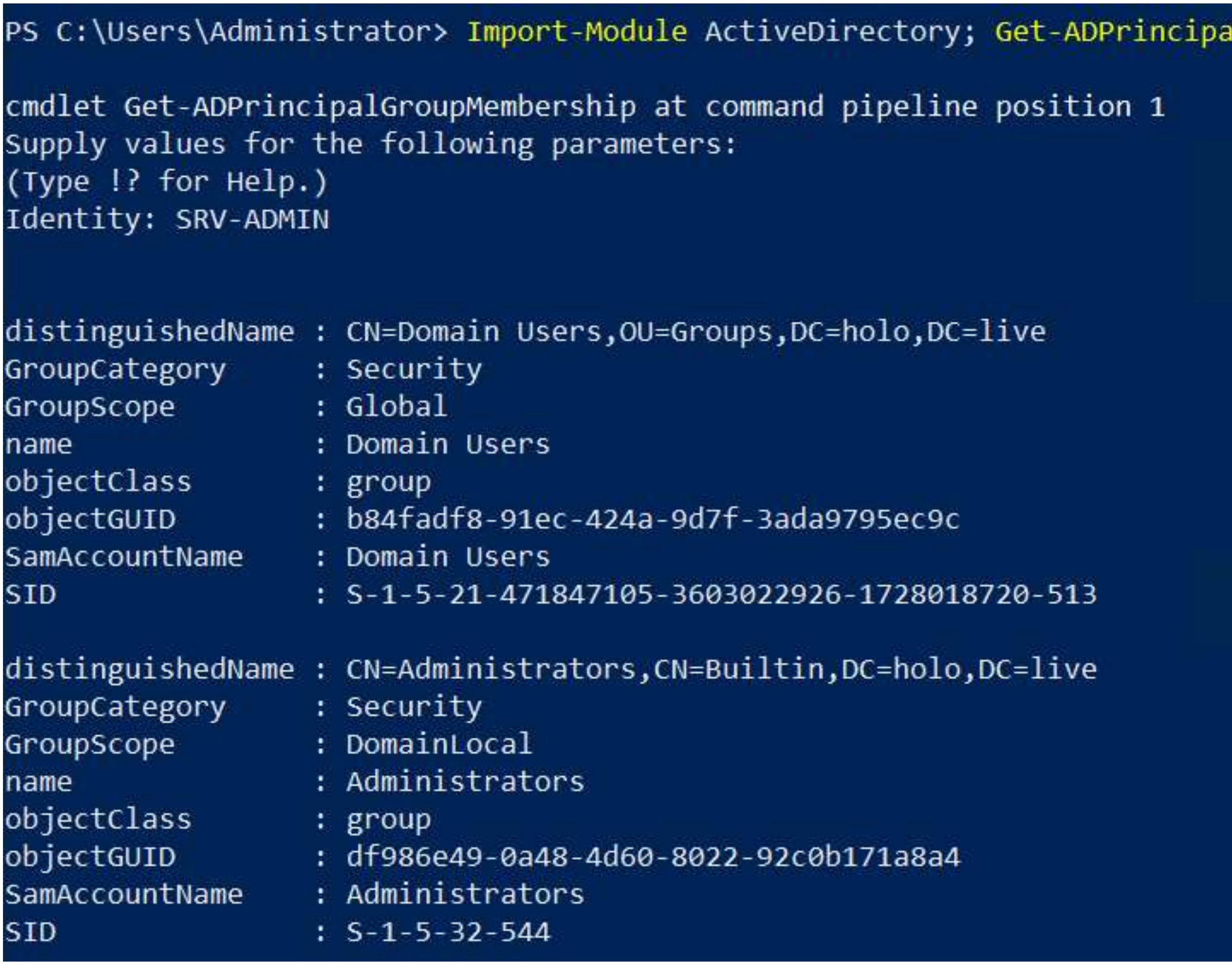
The final PowerShell command we will be looking at is Get-ADPrincipalGroupMembership, similar to Get-ADGroupMember, this command will retrieve the groups a user, computer group, or service account is a member of. In order to get the most out of this command we will need to already have some targeted users enumerated using other commands like Get-ADUser. Since this command is part of the ActiveDirectory module you will need to first import the module. Find the syntax for the command below.

The final PowerShell command we will be looking at is Get-ADPrincipalGroupMembership; similar to Get-ADGroupMember; this command will retrieve the groups a user, computer group, or service

account is a member. To get the most out of this command, we will need to have enumerated target users using other commands like Get-ADUser. Since this command is part of the *ActiveDirectory* module, you will need first to import the module. Find the syntax for the command below.

Syntax: Import-Module ActiveDirectory; Get-ADPrincipalGroupMembership

After running the command, you will be prompted with a CLI to specify the user(s) you want to enumerate.



```
PS C:\Users\Administrator> Import-Module ActiveDirectory; Get-ADPrincipa

cmdlet Get-ADPrincipalGroupMembership at command pipeline position 1
Supply values for the following parameters:
(Type !? for Help.)
Identity: SRV-ADMIN


distinguishedName : CN=Domain Users,OU=Groups,DC=holo,DC=live
GroupCategory     : Security
GroupScope        : Global
name              : Domain Users
objectClass       : group
objectGUID        : b84fadf8-91ec-424a-9d7f-3ada9795ec9c
SamAccountName    : Domain Users
SID               : S-1-5-21-471847105-3603022926-1728018720-513

distinguishedName : CN=Administrators,CN=Builtin,DC=holo,DC=live
GroupCategory     : Security
GroupScope        : DomainLocal
name              : Administrators
objectClass       : group
objectGUID        : df986e49-0a48-4d60-8022-92c0b171a8a4
SamAccountName    : Administrators
SID               : S-1-5-32-544
```

When using PowerShell for offensive operations, you will need to play around with the commands and modules to see what works for you and develop your methodology similar to working with other tools.

Answer the questions below

Read the above and enumerate PC-FILESRV01 using PowerShell.

Completed